POLI TÉCNICO GUARDA

Escola Superior de Tecnologia e Gestão

AUTOMAÇÃO DE TICKETS

RELATÓRIO DE ESTÁGIO
PARA OBTENÇÃO DO GRAU DE LICENCIADO(A) EM ENGENHARIA INFORMÁTICA

Rodrigo Luís Santos Lourenço
Novembro / 2023

Escola Superior de Tecnologia e Gestão

AUTOMAÇÃO DE TICKETS

RELATÓRIO DE ESTÁGIO
PARA OBTENÇÃO DO GRAU DE LICENCIADO(A) EM ENGENHARIA INFORMÁTICA

Professor(a) Orientador(a): Professora Maria Clara Silveira

Rodrigo Luís Santos Lourenço

Novembro / 2023

Agradecimentos

Em primeiro lugar, porque a minha família é o principal pilar da minha formação, quero agradecer aos meus pais, irmã, á minha madrinha e aos meus primos por todo o apoio, força, carinho e ajuda que me foram dando ao longo deste percurso.

Quero agradecer a toda a minha equipa de trabalho da ITCenter, por me terem recebido e ajudado da melhor maneira durante o estágio. Quero deixar um especial agradecimento ao João Paulo São Pedro e ao Ângelo Fonseca, por toda a ajuda que me foi dada. Muito obrigado!

A minha orientadora de estágio Professora Maria Clara Silveira, por toda a sua disponibilidade, ajuda e conselhos dados. E a todos os professores do Instituto Politécnico da Guarda.

Por último, quero agradecer a todos os meus amigos pela amizade, apoio e concelhos ao longo desta jornada.

Ficha de Identificação

Aluno:

Nome: Rodrigo Luís Santos Lourenço
Número: 1700254
Licenciatura: Engenharia Informática

Estabelecimento de Ensino:

Instituto Politécnico da Guarda (IPG)
Escola Superior de Tecnologia e Gestão (ESTG)

Entidade Acolhedora do Estágio:

Nome: ITCenter
Morada: Rua Interior do Europarque, 4520-153 Santa Maria da Feira
Contacto Telefónico: 256 370 980
Duração do Estágio: 05/06/2023 - 01/09/2023

Supervisor de Estágio:

Nome: Pedro Pinho
Função: Manager da área de desenvolvimento

Docente Orientador de Estágio:

Nome: Maria Clara Silveira
Grau Académico: Doutoramento

Resumo

O presente relatório descreve o projeto realizado no âmbito da unidade curricular Projeto de Informática, da Licenciatura de Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda. O projeto, em contexto de estágio, foi realizado na empresa ITCenter.

O objetivo deste projeto consiste no desenvolvimento de uma aplicação, para automatizar processos de gestão de tickets. A aplicação deve:

- Permitir a criação de tickets
- Alteração de tickets
- Analisar comentários
- Analisar causas

Para o desenvolvimento deste projeto foi utilizada uma metodologia ágil, bem como as tecnologias: *IntellIJ*, *VScode*.

Palavras-chave

Github, *Java*, *Tickets*, *WorkFlows*.

Abstract

This report describes the project carried out as part of the IT Project course of the IT Engineering degree programme at the School of Technology and Management of the Polytechnic Institute of Guarda. The internship project was carried out at the ITCenter company.

The aim of this major project is to develop an application to automate ticket management processes. The application should

- Allow tickets to be created
- Modify tickets
- Analysing comments
- Analysing causes

An agile methodology was used to develop this project, as well as the following technologies: IntellIJ, VScode.

Keywords

Github, Java, Tickets, WorkFlows.

Índice

Índice de Figuras

Lista de Siglas

UML	Unified Modeling Language
WF	WorkFlow

1. Introdução

O presente relatório descreve o projeto desenvolvido em contexto de estágio na ITCenter, pelo aluno Rodrigo Luís Santos Lourenço, no âmbito da unidade curricular de Projeto de Informática, pertence ao 3º ano da Licenciatura em Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda.

1.1 Caraterização sumária da instituição

Fundada em 2003, a ITCenter está sediada em Portugal e tem também escritórios nos EUA (Nova Iorque), França (Paris) e Reino Unido (Londres.)

A empresa foca-se em encontrar soluções inovadoras para as necessidades dos clientes, e determinação em ter sucesso fez da empresa um líder no sector das TI.

Líder em inovação desde o primeiro dia, a ITCenter é uma empresa de serviços tecnológicos com um conjunto de competências multidisciplinares:

- *cloud infrastructure*
- *cybersecurity*
- *IT operations & Maintenance*
- *multi-vendor reselling*
- *networking*
- *software & apps development*

1.2 Motivação | Enquadramento

Como finalista da licenciatura de engenharia informática, a procura do primeiro trabalho e a entrada no mesmo gera alguma ansiedade e preocupação. Assim um estágio proporciona-nos um ambiente de trabalho diferente do que estamos habituados, adquirimos novos conhecimentos, como podemos ter a oportunidade de continuarmos nessa empresa depois do estágio curricular.

A escolha da empresa ITCenter baseia-se nos anos de experiência que tem, já conta com 19 anos e nas competências em que a mesma trabalha, nomeadamente em *software & apps devolopment e cybersecurity,* a sua sede é em Santa Maria da Feira e esse era o meu único problema devido a ser um pouco longe da minha cidade, mas eles deram-me a hipótese de fazer por teletrabalho.

1.3 Descrição do problema

Este projeto que ainda continua a ser desenvolvido, destina se para o cliente ALTICE que serve para automatizar *tickets*.

A parte do projeto que me foi pedido fazer, tinha como objetivo de analisar os comentários e as causas da abertura dos tickets.

A MEO utiliza uma plataforma de suporte de *tickets* para registar, procurar e gerir os pedidos dos clientes. Essa plataforma pode ser usada por equipas de atendimento ao cliente, técnicos e outros funcionários para fornecer assistência e solucionar problemas.

Os clientes podem abrir *tickets* de suporte de várias maneiras, como ligando para uma central de atendimento da MEO, enviando um e-mail ou preenchendo um formulário online. Os *tickets* podem abordar uma serie de questões, como problemas técnicos, problemas facturamento, alterações de plano, cancelamento de serviço, atendimento do cliente e suporte técnico, mudanças de endereço, problemas de qualidade de serviço, *feedback* e reclamações e duvidas gerais (sobre serviços ou produtos).

1.4 Objetivos

Integrar um projeto que se encontra em produção, mas com desenvolvimento continuo.

Acompanhar os seus colegas de equipa na criação e manutenção de novos *WorkFlow* (WF), na correção de bugs/novas melhorias nos WF já existentes.

Corrigir e implementar melhorias nos *WF* já existentes autonomamente.

1.5 Estrutura do documento

O presente relatório está dividido em 8 capítulos, que descrevem as etapas para a elaboração deste projeto.

O primeiro capítulo é uma introdução ao projeto desenvolvido, objetivos e motivação. No segundo é apresentado o Estado da Arte. No terceiro capítulo é descrita a metodologia que foi aplicada no projeto. O quarto capítulo corresponde a análise de requisitos onde foi feito o planeamento da aplicação. No quinto capítulo é feita a descrição das tecnologias utilizadas durante o desenvolvimento do projeto. No sexto capítulo é feita uma descrição da implementação do mesmo. No sétimo capítulo são apresentados os testes feitos. E no último capítulo, o capítulo oito é apresentada a conclusão.

2. Estado da Arte

Estado da arte pode ser definido como o estado atual de conhecimento ou avanço de um determinado assunto que está a ser tema de estudo. Este estudo é um passo essencial para identificar procurar resolver falhas existentes. Esta parte, do grande projeto em que fui integrado no estágio era dedicada a operadora de telecomunicações, a MEO. A mesma oferece serviços de atendimento ao cliente e suporte técnico para seus clientes. Esses serviços podem incluir a automação de *tickets*, que é uma abordagem comum para gerir solicitações, reclamações e problemas dos clientes.

Em relação a esta aplicação que se esta a desenvolver existem várias evoluções e vantagens que podem melhorar a satisfação do cliente e a qualidade do suporte ao longo do tempo, para não falar que também ajuda a própria empresa de telecomunicações.

Eficiência e velocidade: Antes os processos de suporte dependiam muito de interações manuais, o que fazia com que o processo fosse mais lento e sujeito a erros. Com a automação, a abertura e a resolução de *tickets* podem ser feitos de forma mais rápida e eficiente, reduzindo assim o tempo de resposta e o aumento da produtividade. Como por exemplo, os clientes em vez de terem de ligar ou enviar e-mails para relatar problemas, sistemas automatizados de abertura de *tickets* podem ser implementados, permitindo que eles registem problemas por meio de plataformas online. Outro exemplo simples, reiniciar um dispositivo remotamente, reduzindo assim a dependência da intervenção do ser humano.

Melhoria na Experiência do cliente: simplificar processos, reduzindo assim tempos de espera e oferecendo soluções rápidas e eficazes. Um exemplo disto seria a implementação de sistemas de auto atendimento, que permitissem aos clientes resolverem problemas comuns por conta própria, reduzindo o tempo de espera e proporcionando respostas imediatas.

Acesso facilitado a informações dos clientes: Sistemas que podem integrar informações contextuais dos clientes, como o histórico de chamadas, serviços assinados, problemas anteriores, para oferecer um suporte mais personalizado.

Suporte Virtual: Integração de assistentes virtuais para lidar com consultas simples, fornecendo respostas imediatas e aliviando a carga de trabalho dos agentes para questões mais complexas.

Integração de Sistemas e Processos: a evolução pode envolver uma maior integração entre sistemas internos e externos. Permitindo um fluxo mais suave de informações, desde a entrada dos tickets até ao processo de resolução.

Personalização e Automação: Com base nos dados do cliente, a automação pode se tornar mais personalizada, prevendo necessidades e antecipando problemas antes que surjam, melhorando a experiência do utilizador.

Por exemplo, com base no histórico de interações do cliente, a automação pode oferecer sugestões personalizadas para a solução de problemas. Se um cliente enfrentou problemas semelhantes no passado, a automação pode sugerir soluções com base nas resoluções anteriores.

Outro exemplo, a partir de padrões de uso, a automação pode oferecer promoções personalizadas ou upgrades de serviço que se alinhem com as necessidades do cliente, tornando a oferta mais relevante e atraente.

Comunicação Proativa: Com base no comportamento e nos padrões de uso do cliente, a automação pode prever necessidades futuras. Por exemplo, se um cliente habitualmente consome mais dados no final do mês, a automação pode sugerir atualizações de pacotes ou fornecer dicas para gerir o uso de dados.

Melhoria na Análise e Relatórios: Sistemas podem oferecer analises mais avançadas, permitindo que a empresa identifique tendências, áreas problemáticas e pontos de melhoria no suporte ao cliente.

Dois exemplos disso são, se houver um aumento repentino de *tickets* relacionados a conexões de internet em uma determinada região, isso pode indicar um problema mais amplo que precisa de atenção. O outro exemplo, a analise avançada pode revelar áreas especificas do serviço que geram mais tickets ou têm taxas mais altas de insatisfação. Se os clientes reportam problemas com um determinado modelo de router, isso pode sinalizar a necessidade de atualizações ou suporte apropriado.

3. Metodologia

A ITCenter privilegia a metodologia ágil (*Agile methodologies*) no desenvolvimento dos projetos, proporcionando mais e melhor interação entre a equipa, a promover *feedback* constante entre esta e o cliente ao atender o ciclo desenvolvimento de *software* (Figura 1). Ao longo da semana eram feitas 3 ou 4 reuniões que por norma eram feitas de manhã onde era dito o que tínhamos feito desde a última reunião, onde duvidas eram tiradas no caso de existirem e era onde discutíamos ideias entre todos, o principal objetivo da metodologia ágil é fornecer um processo mais colaborativo, adaptável e orientado a resultados.

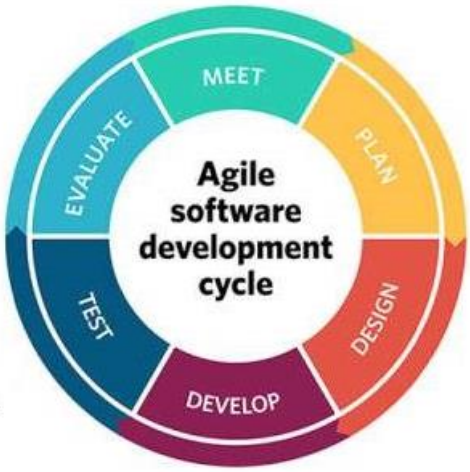


Figura 1 - Diagrama do ciclo de desenvolvimento de software em metodologia Ágil

Os principais valores dessa metodologia são [2]:

- A maior prioridade é satisfazer o cliente através da entrega antecipada e continua de software com valor;
- Aceitar alterações de requisitos, mesmo que seja no final do desenvolvimento. Os processos ágeis aproveitam a mudança em benefício da vantagem competitiva do cliente;
- Entregas de software funcional com frequência, de algumas semanas a alguns meses, com uma preferência de prazos mais curtos;
- A equipa de desenvolvimento e o cliente devem trabalhar juntos, diariamente, durante o decorrer do projeto;

- Desenvolver projetos em torno de indivíduos motivados, dando-lhes o ambiente e o apoio de que necessitam, confiando nos mesmo que irão cumprir os objetivos;
- O método mais eficiente e eficaz de transmitir informações para e dentro de uma equipa de desenvolvimento mediante uma conversa pessoal e direta;
- O software funcional é a principal medida de progresso.
- Os processos ágeis promovem o desenvolvimento sustentável. Os promotores, os programadores e os utilizadores deverão ser capazes de manter um ritmo constante.
- A atenção continua à excelência técnica e ao bom design aumenta a agilidade.
- As melhores arquiteturas, requisitos e projetos emergem de equipas auto-organizadas.

Esta forma de desenvolvimento permite uma rápida identificação ou correção de problemas e, durante o estágio, tivemos a oportunidade de conhecer, no contexto profissional, o *Scrum*. A metodologia ajuda as equipas a trabalham juntas, estimulando os elementos do grupo e aprenderem com as experiências, a se organizarem enquanto resolvem um problema e a refletirem sobre os êxitos e fracassos para um aperfeiçoamento contínuo.

3.1 Scrum

Scrum é uma metodologia de desenvolvimento ágil utilizada no desenvolvimento de *Software*, adaptável, rápida, flexível e eficaz que é projetada para oferecer valor ao cliente durante todo o desenvolvimento do projeto [1].

Conduzidos pelo *Scrum*, semanalmente são realizadas três a quatro reuniões. Uma logo segunda-feira de manhã, outra na quarta-feira de manhã e uma ou duas na sexta-feira, uma de manhã e outra há tarde. Basicamente estas reuniões denominadas por *Daily´s*, era falado o que se tinha feito nas horas/dias anteriores á mesma, alguma dúvida que tivéssemos podíamos expor a mesma nas reuniões, para que nos pudessem esclarecer e para ouvirmos as várias opiniões dos colegas de equipa. Algumas dessas reuniões eram com o cliente o que tornava mais fácil questionar sobre alguma dúvida que tivéssemos ou algum erro que o programa estivesse a dar.

3.2 Formação Complementar

No início do estágio foi-me solicitado para tirar duas formações, uma sobre o Git e a outra sobre o Docker.

Git

O Git é um sistema de controlo de versões amplamente utilizado no desenvolvimento de software, que permite aos developers acompanhar e controlar as alterações feitas em um projeto ao longo do tempo, facilitando a colaboração em equipe e a reversão para versões anteriores [5].
Ao longo deste curso, foi possível aprimorar e relembrar habilidades no uso do Git, durante as aulas foram abordados conceitos fundamentais do Git, como criação de repositórios, gerir branchs, merges, rebase e resolução de conflitos. Além disso o curso também abordou recursos avançados do Git, como *stash* e *tags*. Pude aperfeiçoar os meus conhecimentos no uso do Git e entender como essa ferramenta pode facilitar o desenvolvimento de projetos em equipa.

Docker

Docker é uma plataforma aberta para developers e administradores de sistemas criarem, enviarem e executarem aplicações distribuídas, seja em laptops, Virtual Machine de Data Center ou na *cloud*. É uma plataforma de virtualização de *containers* que permite a criação e a gestão de ambientes isolados para a execução de aplicações. Ela facilita a organização do *software*, incluindo todas as suas dependências, em *containers* leves e portáteis [4].

Através do curso, foi adquirido conhecimento sobre a utilização de ferramentas adicionais que complementam o Docker, como o Docker Compose, que facilita a gestão de aplicações compostas por vários containers. Essa habilidade foi muito útil para criar ambientes de desenvolvimento de software.
Isso permitiu me compreender melhor os benefícios que traz para o desenvolvimento de um *software*.

4. Análise de Requisitos

Neste capítulo será apresentada análise de requisitos.

Foi proposto fazer um diagrama na ferramenta PlantUML para descrever o *Workflow* a desenvolver durante o estágio. Para desenvolver este PlantUML fui-me guiando por exemplos de outros *Workflows* já desenvolvidos anteriormente por membros da minha equipa.

Este diagrama tem como objetivo analisar os comentários e as causas dos tickets.

Na figura 2, para começar temos uma atividade que define ou configura o contexto para a execução, podendo envolver a definição de variáveis, configuração do ambiente, ou outras ações relacionadas ao contexto do sistema.

De seguida passamos para o passo seguinte que é o sistema *k2view*, que é uma empresa de software especializada em soluções para gerir e integrar dados. Eles oferecem uma plataforma de *software* chamada *k2view* que foi desenvolvida para simplificar o acesso e a integração de dados de várias fontes em ambientes empresariais. Neste ponto o sistema realiza a ação de tentar obter comentários, procura na base de dados comentários para os recuperar.

Uma nova atividade é iniciada envolvendo a verificação de uma base de dados, para a decisão seguinte que pergunta e verifica se a lista de comentários esta vazia ou não.

Se a lista de comentários tiver vazia, então avançamos para construir um relatório no sistema Maestro Deo dizendo que já não há mais comentários para tratar ou ver, acabando aqui todo o processo com sucesso.

Se a lista de comentários não estiver vazia, ou seja, a resposta a decisão for não então avançamos para o próximo passo.

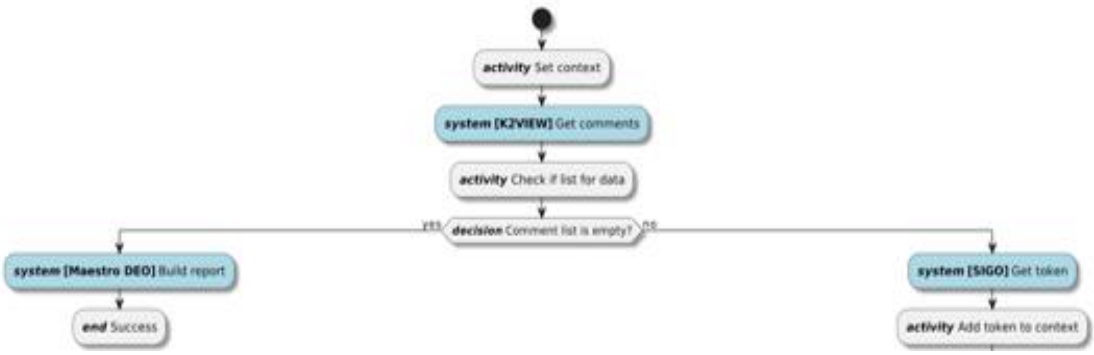


Figura 2 – Modelo UML do WorkFlow 1/3

O próximo passo vai obter um *token* do sistema SIGO para esse e só para esse *ticket*, de seguida na próxima atividade vai adicionar o *token* ao *ticket*, na Figura 3 a próxima atividade é feita uma verificação á resolução dos *tickets* e suas datas.

O sistema *Scanner* que foi criado pela empresa/equipa onde fui integrado, algum tempo antes de começar o estágio. Esse *Scanner* vai verificar o comentário e/ou a causa desse *Ticket*.

Na próxima atividade é feita uma verificação para se saber se o *Ticket* já foi tratado, vai haver uma decisão com duas respostas possíveis, ou sim (já foi tratado) ou não (ainda não foi tratado).

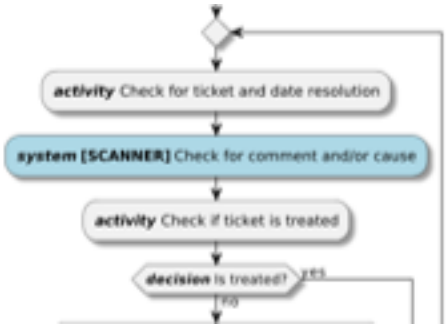


Figura 3 – Modelo UML do WorkFlow 2/3

Se já foi tratado passa logo praticamente para o fim do processo todo como mostra na Figura 4, para o sistema Maestro Deo para a construção do relatório nesse sistema que por sua vez vai diretamente para a atividade que vai verificar o próximo comentário, passando para a próxima decisão que procura se há mais comentários para verificar. No caso de haver volta praticamente para o início do processo, mais propriamente para atividade que faz a verificação da resolução do *ticket* e sua data, no caso de não haver acaba o procedimento todo com sucesso.

Voltando ao ponto anterior, senão foi tratado vai para a decisão que vai verificar se o campo de observações esta vazio ou nulo. Se estiver vazio ou nulo avança logo para a decisão seguinte que verifica se o campo da causa esta vazio ou nulo.

Se o campo de observações não estiver vazio nem nulo Figura 5, continua o processo passando de seguida para o sistema Sigo que vai atualizar o *ticket*. De imediato é atualizado o relatório com as informações que obtivemos até ao mesmo, guardando as observações pelo sistema scanner.

Por fim o *ticket* passa para o sistema Maestro Deo, construindo assim um relatório com toda a informação, passando para a atividade que vai verificar se existe mais algum comentário, é tomada uma decisão sobre a verificação que foi feita na possível existência de mais comentários, se não existir mais nenhum comentário acaba todo o

processo com sucesso, se existir mais algum comentário, volta para a atividade que verifica a resolução do *ticket* e a data fazendo os processos todos novamente ate chegar ao fim sem nenhum comentário.

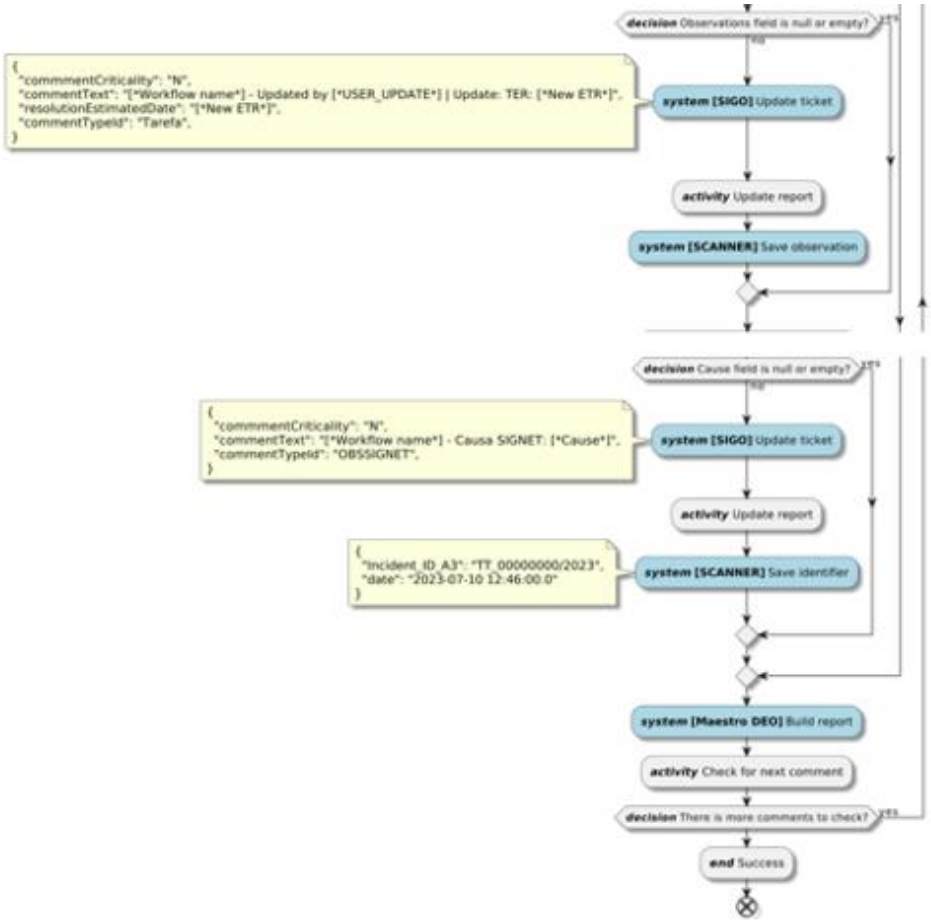


Figura 4 – Modelo UML do WorkFlow 3/3

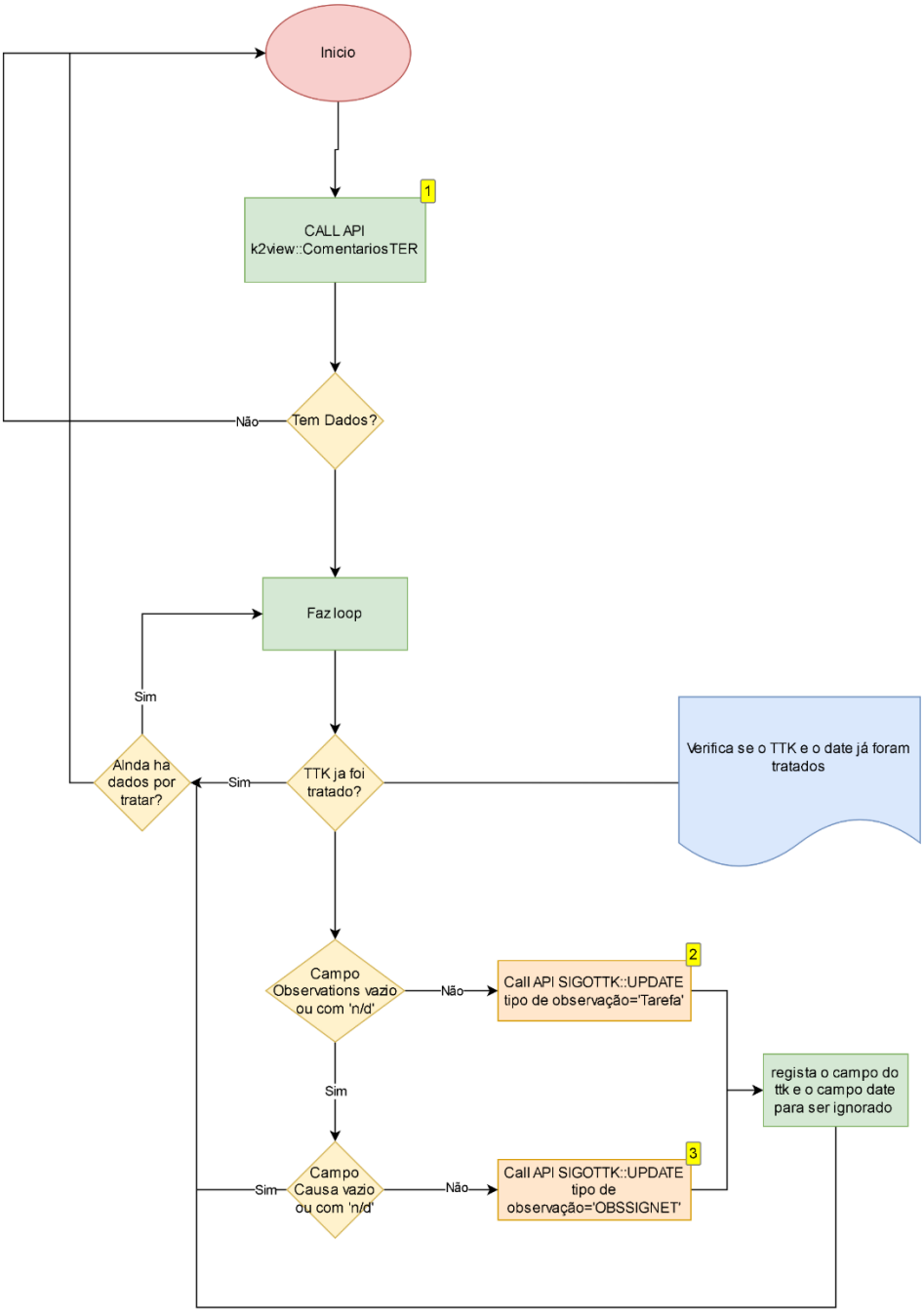


Figura 5 - Diagrama do WorkFlow

5. Tecnologias

Neste capítulo são apresentadas as tecnologias que ajudaram no desenvolvimento do projeto.

Java

Java é uma linguagem orientada a objetos e centrada em rede que pode ser usada como uma plataforma em si. É uma linguagem de programação rápida, segura e confiável para codificar tudo, desde aplicações móveis e *software* empresarial até aplicações de big data e tecnologias do servidor.

Kotlin

Kotlin é uma linguagem de programação moderna e concisa, compatível com java, usada principalmente para o desenvolvimento de aplicações Android, mas também em outras áreas de desenvolvimento de software. Ela foi projetada para ser mais segura, eficiente e produtiva que o Java, oferecendo recursos avançados, como inferência de tipo e extensões de funções. Esta tecnologia foi usada em praticamente todo o projeto que foi desenvolvido.

GitHub

GitHub não é uma linguagem de programação, mas sim uma plataforma de hospedagem de código e colaboração para projetos de desenvolvimento de software. Ela fornece ferramentas para o controlo de versões de código, gestor de projetos, busca de problemas e colaboração entre desenvolvedores, permitindo que equipas trabalhem juntas no desenvolvimento do *software* de forma eficiente. O GitHub foi utilizado para hospedar o código fonte e gerir controlo de versões.

6. Implementação

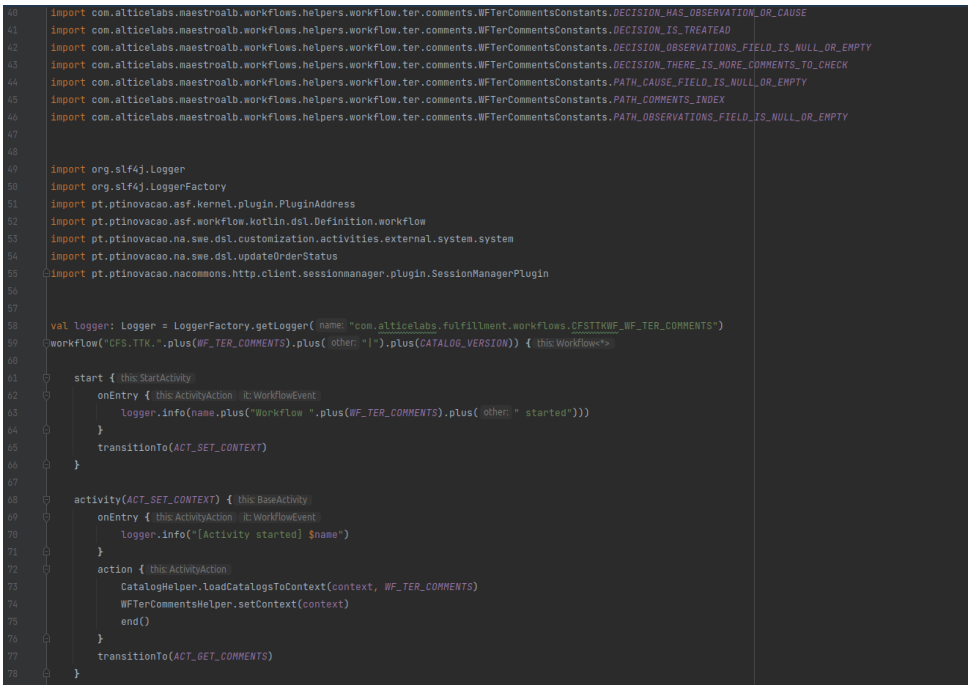
Neste capítulo será apresentado o processo de implementação dos requisitos pedidos pelo cliente.

6.1 Desenvolvimento do Workflow

O *WorkFlow* é das partes mais importantes deste trabalho, é como se fosse a base de tudo. Todos os *workflows* dentro deste grande projeto são desenvolvidos da mesma maneira, ou seja, as *activities* as *decisions* e os *systems* são todos elaborados da mesma maneira, dependendo ou não se têm ações no caso das atividades, e se têm transições ou não no caso das decisões, alterando depois só os métodos em cada *activities*, *decisions*, *systems* de acordo com o *workflow* que se está a tratar.

```
package com.alticelabs.maestroalb.workflows


import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.ACT_ADD_TOKEN_CTX
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.DECISION_LABEL_NO
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.DECISION_LABEL_YES
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.PATH_CUSTOM_CATALOG_TOKEN
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.SYSTEM_MAESTRO_DEO
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.SYSTEM_SCANNER
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.SYSTEM_SIGO
import com.alticelabs.maestroalb.context.helper.getActivityCondition
import com.alticelabs.maestroalb.context.helper.getActivityResponse
import com.alticelabs.maestroalb.context.helper.saveActivityCondition
import com.alticelabs.maestroalb.workflows.helpers.catalog.ref.data.CatalogHelper
import com.alticelabs.maestroalb.workflows.helpers.common.DataReportCommonHelper
import com.alticelabs.maestroalb.workflows.helpers.order.OrderExceptionCode
import com.alticelabs.maestroalb.workflows.helpers.systems.maestrodeo.MaestroDeoHelper
import com.alticelabs.maestroalb.workflows.helpers.systems.sigo.SigoHelper
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsHelper
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsSystemsHelper
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.WF_TER_COMMENTS
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.CATALOG_VERSION
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.ACT_SET_CONTEXT
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.ACT_GET_COMMENTS
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.ACT_CHECK_LIST_FOR_DATA
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.SYSTEM_K2VIEW
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.DECISION_COMMENT_LIST_IS_EMPTY
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.ACT_BUILD_REPORT
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.ACT_GET_SIGO_TOKEN
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.ACT_SUCCESS
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.ACT_UPDATE_REPORT
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.ACT_CHECK_FOR_COMMENT_AND_CAUSE
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.ACT_CHECK_FOR_NEXT_COMMENT
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.ACT_CHECK_FOR_TICKET_AND_DATE_RESOLUTION
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.ACT_CHECK_IF_TICKET_IS_TREATED
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.ACT_SAVE_IDENTIFIER
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.ACT_UPDATE_SIGO_TICKET_CAUSE
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.ACT_UPDATE_SIGO_TICKET_OBSERVATION
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.DECISION_CAUSE_FIELD_IS_NULL_OR_EMPTY
```

Figura 6 - WorkFlow 1/9



```
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.DECISION_HAS_OBSERVATION_OR_CAUSE
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.DECISION_IS_TREATEAD
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.DECISION_OBSERVATIONS_FIELD_IS_NULL_OR_EMPTY
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.DECISION_THERE_IS_MORE_COMMENTS_TO_CHECK
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.PATH_CAUSE_FIELD_IS_NULL_OR_EMPTY
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.PATH_COMMENTS_INDEX
import com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments.WFTerCommentsConstants.PATH_OBSERVATIONS_FIELD_IS_NULL_OR_EMPTY


import org.slf4j.Logger
import org.slf4j.LoggerFactory
import pt.ptinovacao.asf.kernel.plugin.PluginAddress
import pt.ptinovacao.asf.workflow.kotlin.dsl.Definition.workflow
import pt.ptinovacao.na.swe.dsl.customization.activities.external.system.system
import pt.ptinovacao.na.swe.dsl.updateOrderStatus
import pt.ptinovacao.nacommons.http.client.sessionmanager.plugin.SessionManagerPlugin


val logger: Logger = LoggerFactory.getLogger( name: "com.alticelabs.fulfillment.workflows.CFSTTKWF_WF_TER_COMMENTS")
workflow("CFS.TTK.".plus(WF_TER_COMMENTS).plus( other: "|").plus(CATALOG_VERSION)) { this: Workflow<*>

    start { this: StartActivity
        onEntry { this: ActivityAction  it: WorkflowEvent
            logger.info(name.plus("Workflow ".plus(WF_TER_COMMENTS).plus( other: " started")))
        }
        transitionTo(ACT_SET_CONTEXT)
    }

    activity(ACT_SET_CONTEXT) { this: BaseActivity
        onEntry { this: ActivityAction  it: WorkflowEvent
            logger.info("[Activity started] $name")
        }
        action { this: ActivityAction
            CatalogHelper.loadCatalogsToContext(context, WF_TER_COMMENTS)
            WFTerCommentsHelper.setContext(context)
            end()
        }
        transitionTo(ACT_GET_COMMENTS)
    }
```

Figura 7 - WorkFlow 2/9

Na Figura 7, começa o desenvolvimento do *workflow* iniciando na *activity* (*ACT_SET_CONTEXT*), que faz um carregamento de um catálogo, em seguida o contexto vai ser definido usando a função *WFTerCommentsHelper*, passando para a próxima transição (*GET_COMMENTS*).

Na Figura 8, começamos com um *systems* que a primeira coisa que faz é invocar a operação *"get"* no sistema *k2view* usando o endereço de plugin do *SessionManagerPlugin*, de seguida regista uma mensagem informativa dizendo que já foi invocado, chamando a função para obter os comentários. A atividade a seguir obtém a resposta da atividade anterior do contexto, verificando se a lista de comentários esta vazia é guardada a condição no contexto, indicando se a lista de comentários esta vazia ou não. Nesta Figura 8, por último temos uma decisão composta por duas transições, uma caso a condição seja verdadeira e outra caso seja falsa.

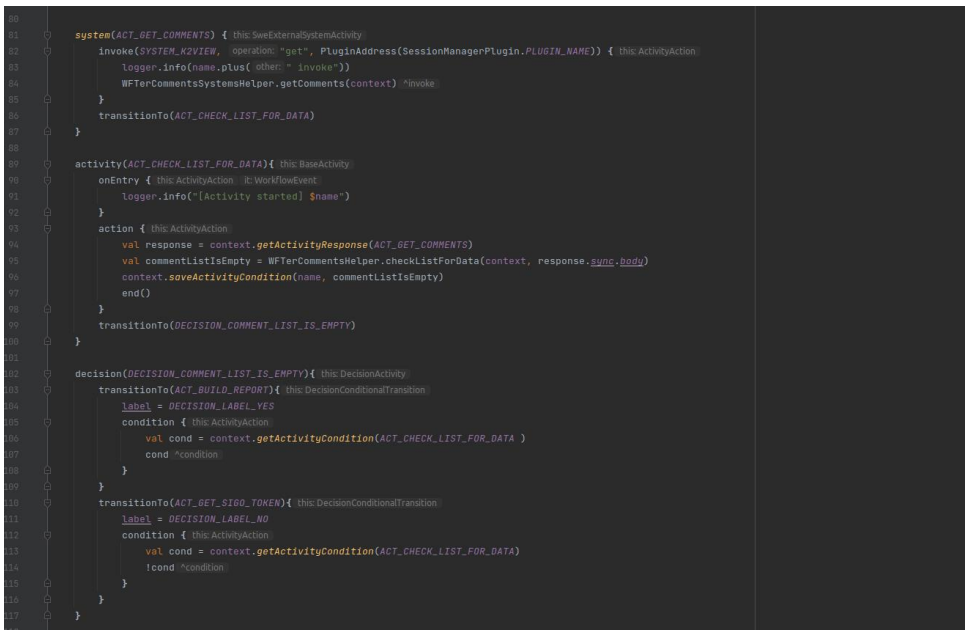
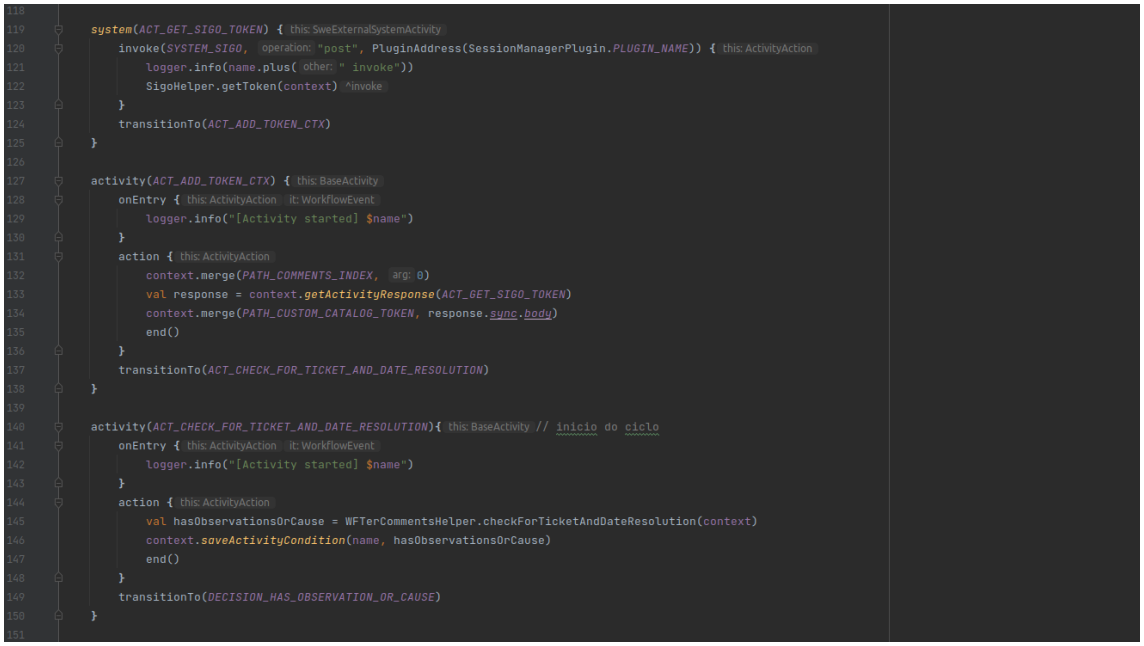


Figura 8 - WorkFlow 3/9

Na Figura 9, temos outro sistema que onde invoca uma operação *"post"* no sistema SIGO, chamando uma função de *SigoHelper* para obter um token.

Na próxima atividade (*ACT_ADD_TOKEN_CTX*) vai concatenar o valor 0 no contexto sob a chave *PATH_COMMENTS_INDEX*, o mesmo vai acontecer com o *body*, obtendo também a resposta da atividade anterior.

A última atividade (**ACT_CHECK_FOR_TICKET_AND_DATE_RESOLUTION**) verifica se há alguma observação ou causa usando a função *WFTerCommentsHelper*, guardando de seguida a condição da atividade no contexto, indicando se há observações ou causas.



```
system(ACT_GET_SIGO_TOKEN) { this: SweExternalSystemActivity
    invoke(SYSTEM_SIGO,  operation: "post", PluginAddress(SessionManagerPlugin.PLUGIN_NAME)) { this: ActivityAction
        logger.info(name.plus( other: " invoke"))
        SigoHelper.getToken(context) ^invoke
    }
    transitionTo(ACT_ADD_TOKEN_CTX)
}

activity(ACT_ADD_TOKEN_CTX) { this: BaseActivity
    onEntry { this: ActivityAction  it: WorkflowEvent
        logger.info("[Activity started] $name")
    }
    action { this: ActivityAction
        context.merge(PATH_COMMENTS_INDEX,  arg: 0)
        val response = context.getActivityResponse(ACT_GET_SIGO_TOKEN)
        context.merge(PATH_CUSTOM_CATALOG_TOKEN, response.sync.body)
        end()
    }
    transitionTo(ACT_CHECK_FOR_TICKET_AND_DATE_RESOLUTION)
}

activity(ACT_CHECK_FOR_TICKET_AND_DATE_RESOLUTION){ this: BaseActivity // inicio do ciclo
    onEntry { this: ActivityAction  it: WorkflowEvent
        logger.info("[Activity started] $name")
    }
    action { this: ActivityAction
        val hasObservationsOrCause = WFTerCommentsHelper.checkForTicketAndDateResolution(context)
        context.saveActivityCondition(name, hasObservationsOrCause)
        end()
    }
    transitionTo(DECISION_HAS_OBSERVATION_OR_CAUSE)
}
```

Figura 9 - WorkFlow 4/9

Na figura 10 na primeira decisão (*DECISION_HAS_OBSERVATION_OR_CAUSE*) obtém simplesmente as condições da atividade (*ACT_CHECK_FOR_TICKET_AND_DATE_RESOLUTION*) retornando a condição e a negação da condição. Caso a condição seja verdadeira, o fluxo segue para a atividade (*ACT_CHECK_FOR_COMMENT_AND_CAUSE*), caso contrário segue para a (*ACT_BUILD_REPORT*).

No sistema, esta a ser feita uma chamada do *SYSTEM_SIGO* para obter comentários, de seguida utiliza um *Helper* (*WFTerCommentsSystemHelper*)para verificar os comentários e causas.

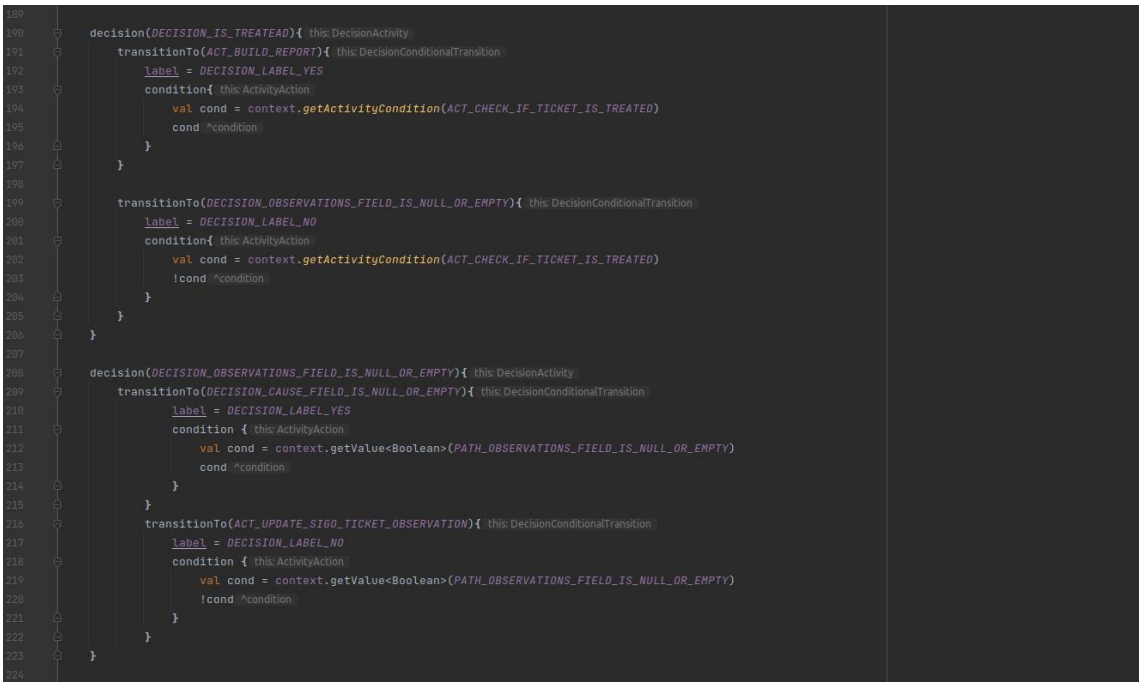
Por último, na atividade (*ACT_CHECK_IF_TICKET_IS_TREATED*), obtém se a resposta da atividade anterior (*ACT_CHECK_FOR_COMMENT_AND_CAUSE*), verifica se o *ticket* está tratado usando a função *checkIfTicketIsTreated*, e de seguida é guardada a condição, indicando se o *ticket* está tratado ou não.

```
decision(DECISION_HAS_OBSERVATION_OR_CAUSE){ this: DecisionActivity
    transitionTo(ACT_CHECK_FOR_COMMENT_AND_CAUSE){ this: DecisionConditionalTransition
        label = DECISION_LABEL_YES
        condition { this: ActivityAction
            val cond = context.getActivityCondition(ACT_CHECK_FOR_TICKET_AND_DATE_RESOLUTION)
            cond ^condition
        }
    }
    transitionTo(ACT_BUILD_REPORT){ this: DecisionConditionalTransition
        label = DECISION_LABEL_NO
        condition { this: ActivityAction
            val cond = context.getActivityCondition(ACT_CHECK_FOR_TICKET_AND_DATE_RESOLUTION)
            !cond ^condition
        }
    }
}

system(ACT_CHECK_FOR_COMMENT_AND_CAUSE){ this: SweExternalSystemActivity
    invoke(SYSTEM_SCANNER, operation: "get", PluginAddress(SessionManagerPlugin.PLUGIN_NAME)) { this: ActivityAction
        logger.info(name.plus( other: " invoke"))
        WFTerCommentsSystemsHelper.checkForCommentAndCause(context) ^invoke
    }
    transitionTo(ACT_CHECK_IF_TICKET_IS_TREATED)
}

activity(ACT_CHECK_IF_TICKET_IS_TREATED){ this: BaseActivity
    onEntry { this: ActivityAction it: WorkflowEvent
        logger.info("[Activity started] $name")
    }
    action { this: ActivityAction
        val response = context.getActivityResponse(ACT_CHECK_FOR_COMMENT_AND_CAUSE)
        val isTreated = WFTerCommentsHelper.checkIfTicketIsTreated(context, response.sync.body)
        context.saveActivityCondition(name, isTreated)
        end()
    }
    transitionTo(DECISION_IS_TREATEAD)
}
```

Figura 10 - WorkFlow 5/9

Na Figura 11, na primeira decisão obtém a condição da atividade (*ACT_CHECK_IF_TICKET_IS_TREATED*) retornando a condição e negação da condição respetivamente.

Na outra condição é obtido o valor booleano do contexto, retornando a condição e negação da condição respetivamente.



```
decision(DECISION_IS_TREATEAD){ this: DecisionActivity
    transitionTo(ACT_BUILD_REPORT){ this: DecisionConditionalTransition
        label = DECISION_LABEL_YES
        condition{ this: ActivityAction
            val cond = context.getActivityCondition(ACT_CHECK_IF_TICKET_IS_TREATED)
            cond ^condition
        }
    }

    transitionTo(DECISION_OBSERVATIONS_FIELD_IS_NULL_OR_EMPTY){ this: DecisionConditionalTransition
        label = DECISION_LABEL_NO
        condition{ this: ActivityAction
            val cond = context.getActivityCondition(ACT_CHECK_IF_TICKET_IS_TREATED)
            !cond ^condition
        }
    }
}

decision(DECISION_OBSERVATIONS_FIELD_IS_NULL_OR_EMPTY){ this: DecisionActivity
    transitionTo(DECISION_CAUSE_FIELD_IS_NULL_OR_EMPTY){ this: DecisionConditionalTransition
            label = DECISION_LABEL_YES
            condition { this: ActivityAction
                val cond = context.getValue<Boolean>(PATH_OBSERVATIONS_FIELD_IS_NULL_OR_EMPTY)
                cond ^condition
            }
        }
        transitionTo(ACT_UPDATE_SIGO_TICKET_OBSERVATION){ this: DecisionConditionalTransition
            label = DECISION_LABEL_NO
            condition { this: ActivityAction
                val cond = context.getValue<Boolean>(PATH_OBSERVATIONS_FIELD_IS_NULL_OR_EMPTY)
                !cond ^condition
            }
        }
}
```

Figura 11 - WorkFlow 6/9

Na figura 12, o sistema chama a função *updateTicketObservation* para atualizar as observações do *Ticket*, transitando de seguida para a próxima atividade.

Nesta atividade é chamada a função *setActionDataReport* para configurar dados do relatório, transitando para a decisão.

Na decisão (*DECISION_CAUSE_FIELD_IS_NULL_OR_EMPTY*), obtemos o valor booleano de (*PATH_CAUSE_FIELD_IS_NUL_OR_EMPTY)* retornando a condição e a negação da condição respetivamente, dependo da decisão. Caso a decisão seja *"yes"* segue para a atividade (*ACT_BUILD_REPORT*), caso contrário segue para a atividade (*ACT_UPDATE_SIGO_TICKET_CAUSE*).

```
224
225     system(ACT_UPDATE_SIGO_TICKET_OBSERVATION) { this: SweExternalSystemActivity
226         invoke(SYSTEM_SIGO,  operation: "post", PluginAddress(SessionManagerPlugin.PLUGIN_NAME)) { this: ActivityAction
227             logger.info(name.plus( other: " invoke"))
228             WFTerCommentsSystemsHelper.updateTicketObservation(context) ^invoke
229         }
230         transitionTo(ACT_UPDATE_REPORT)
231     }
232
233     activity(ACT_UPDATE_REPORT){ this: BaseActivity
234         onEntry { this: ActivityAction  it: WorkflowEvent
235             logger.info("[Activity started] $name")
236         }
237         action { this: ActivityAction
238             DataReportCommonHelper.setActionsDataReport(context, ACT_UPDATE_SIGO_TICKET_OBSERVATION)
239             end()
240         }
241         transitionTo(DECISION_CAUSE_FIELD_IS_NULL_OR_EMPTY)
242     }
243
244     decision(DECISION_CAUSE_FIELD_IS_NULL_OR_EMPTY){ this: DecisionActivity
245         transitionTo(ACT_BUILD_REPORT){ this: DecisionConditionalTransition
246             label = DECISION_LABEL_YES
247             condition { this: ActivityAction
248                 val cond = context.getValue<Boolean>(PATH_CAUSE_FIELD_IS_NULL_OR_EMPTY)
249                 cond ^condition
250             }
251         }
252         transitionTo(ACT_UPDATE_SIGO_TICKET_CAUSE){ this: DecisionConditionalTransition
253             label = DECISION_LABEL_NO
254             condition { this: ActivityAction
255                 val cond = context.getValue<Boolean>(PATH_CAUSE_FIELD_IS_NULL_OR_EMPTY)
256                 !cond ^condition
257             }
258         }
259     }
260
```

Figura 12 - WorkFlow 7/9

Na Figura 13, o sistema (*ACT_UPDATE_SIGO_TICKET_CAUSE*) chama a função *updateTicketCause* para atualizar as observações do *Ticket*, transitando de seguida para próxima atividade.

Na atividade é chamada a função *setActionDataReport* para configurar dados do relatório, transitando para o sistema.

O sistema chama a função *saveIdentifier* para guardar um identificador, transitando para o próximo sistema.

O último sistema, chama a função *sendReport* para enviar um relatório, transitando para a próxima atividade.

```
260
261     system(ACT_UPDATE_SIGO_TICKET_CAUSE) { this: SweExternalSystemActivity
262         invoke(SYSTEM_SIGO,  operation: "post", PluginAddress(SessionManagerPlugin.PLUGIN_NAME)) { this: ActivityAction
263             logger.info(name.plus( other: " invoke"))
264             WFTerCommentsSystemsHelper.updateTicketCause(context) ^invoke
265         }
266         transitionTo(ACT_UPDATE_REPORT)
267     }
268
269     activity(ACT_UPDATE_REPORT){ this: BaseActivity
270         onEntry { this: ActivityAction  it: WorkflowEvent
271             logger.info("[Activity started] $name")
272         }
273         action { this: ActivityAction
274             DataReportCommonHelper.setActionsDataReport(context, ACT_UPDATE_SIGO_TICKET_CAUSE)
275             end()
276         }
277         transitionTo(ACT_SAVE_IDENTIFIER)
278     }
279
280     system(ACT_SAVE_IDENTIFIER){ this: SweExternalSystemActivity
281         invoke(SYSTEM_SCANNER,  operation: "get", PluginAddress(SessionManagerPlugin.PLUGIN_NAME)) { this: ActivityAction
282             logger.info(name.plus( other: " invoke"))
283             WFTerCommentsSystemsHelper.saveIdentifier(context) ^invoke
284         }
285         transitionTo(ACT_BUILD_REPORT)
286     }
287
288     system(ACT_BUILD_REPORT) { this: SweExternalSystemActivity
289         invoke(SYSTEM_MAESTRO_DEO,  operation: "post", PluginAddress(SessionManagerPlugin.PLUGIN_NAME)) { this: ActivityAction
290             logger.info(name.plus( other: " invoke"))
291             MaestroDeoHelper.sendReport(context, WF_TER_COMMENTS) ^invoke
292         }
293         transitionTo(ACT_CHECK_FOR_NEXT_COMMENT)
294     }
295
```

Figura 13 - WorkFlow 8/9

Nesta figura 14, a atividade chama a função *checkForMoreComments* para verificar se há mais comentários por ver, guardando de seguida a condição da atividade, transitando para a próxima decisão que vai perguntar se há mais comentários para verificar.

Se a decisão for "*yes*" passa para a atividade que vai verificar a resolução do *ticket* e sua data, se for "*no*", acaba assim todo o processo.

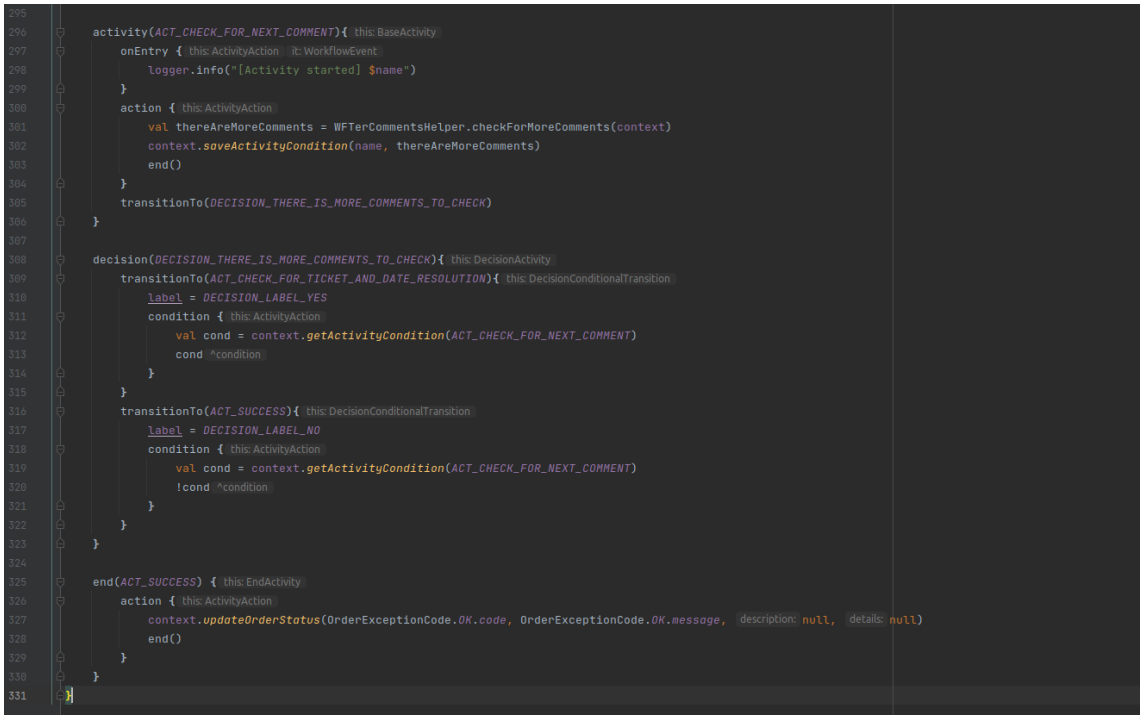


Figura 14 - WorkFlow 9/9

6.2 Desenvolvimento do CommentsHelper

O *CommentsHelper* é uma classe onde vai ser implementados todos os métodos a seguir.

Na figura 15, foi criado um método chamado '*setContext*'. Com o *context.merge* vamos guardar informação sobre a causa se está nula ou vazia, no *Path_Cause_Field_Is_Null_Or_Empty*. Começando com a resposta em falso, para garantir que o caminho tenha algo dentro dele, ou seja, não está vazio. O mesmo se repete com o outro *Path* das *Observations*.

```
public static void setContext(WorkflowContextWrapper context){
    setBaseContext(context);
    context.merge(PATH_CAUSE_FIELD_IS_NULL_OR_EMPTY, arg: false);
    context.merge(PATH_OBSERVATIONS_FIELD_IS_NULL_OR_EMPTY, arg: false);
}
```

Figura 15 – Método setContext

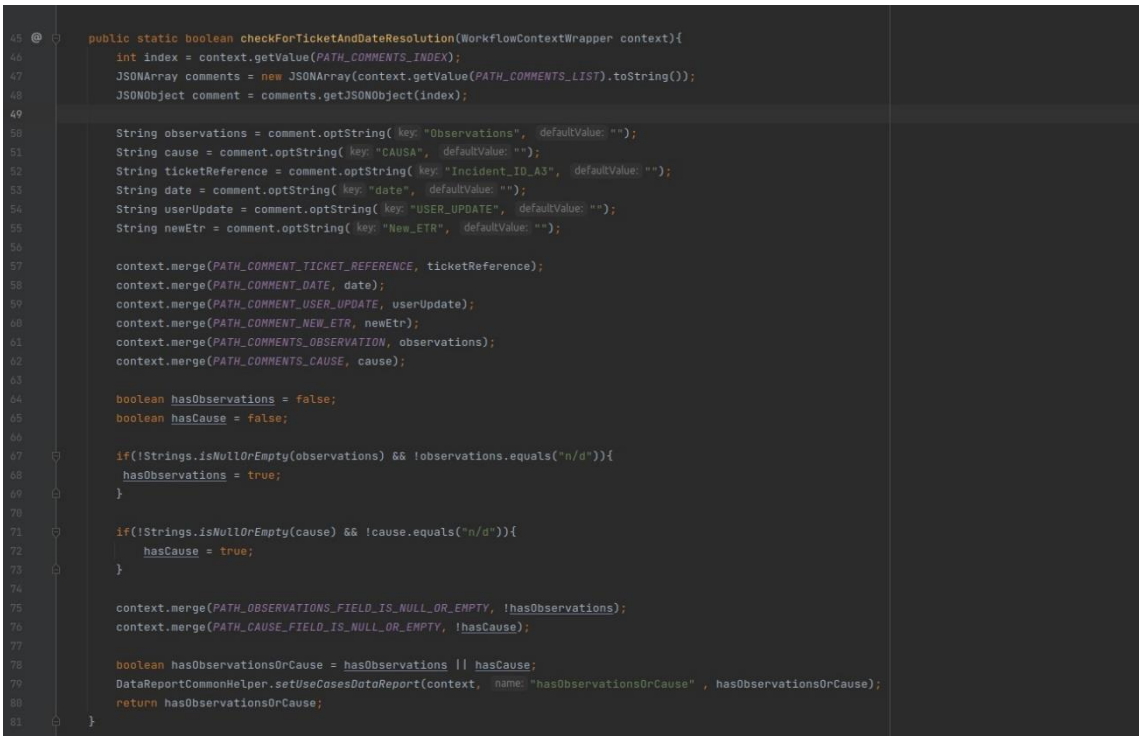
Na figura 16, sabemos se chegarmos a este passo vamos obter a resposta, e a resposta é um *json array*. Cria se um *json array*, para começar a trabalhar os dados para dar a resposta que se quer, neste caso se a lista esta vazia ou não. Este método foi feito para saber se há algo dentro dos comentários. Por último foi feita uma condição para guardar os comentários.

```
32  public static boolean checkListForData(WorkflowContextWrapper context, String response){
33
34      JSONArray comments = new JSONArray(response);
35      boolean commentListIsEmpty = comments.isEmpty();
36
37      if(!commentListIsEmpty){
38          context.merge(PATH_COMMENTS_LIST, comments.toString());
39      }
40
41      DataReportCommonHelper.setUseCasesDataReport(context, name: "commentListIsEmpty" , commentListIsEmpty);
42      return commentListIsEmpty;
43  }
44
```

Figura 16 - Método checkListForData

Neste método criado Figura 17, que tem a função de verificar a resolução dos *tickets* e suas datas. Se o campo de observação vier *null* ou vazio ou com n/d, não se deve atualizar o *ticket*. Se houver conteúdo devera ser feito um pedido de *update* ao Sigo. Estamos assumindo que as observações e a causa vêm vazias, depois cria se um método para verificar se a *String observations* não está vazia, nula ou com n/d. Se esta condição for verdadeira então a variável é definida como verdadeira, o mesmo foi criado para a *String Cause*.

Este método verifica e processa informações associadas a um comentário, atualiza o relatório e retorna um valor indicando se há observações, causa ou ambas.



```
public static boolean checkForTicketAndDateResolution(WorkflowContextWrapper context){
    int index = context.getValue(PATH_COMMENTS_INDEX);
    JSONArray comments = new JSONArray(context.getValue(PATH_COMMENTS_LIST).toString());
    JSONObject comment = comments.getJSONObject(index);

    String observations = comment.optString( key: "Observations",  defaultValue: "");
    String cause = comment.optString( key: "CAUSA",  defaultValue: "");
    String ticketReference = comment.optString( key: "Incident_ID_A3",  defaultValue: "");
    String date = comment.optString( key: "date",  defaultValue: "");
    String userUpdate = comment.optString( key: "USER_UPDATE",  defaultValue: "");
    String newEtr = comment.optString( key: "New_ETR",  defaultValue: "");

    context.merge(PATH_COMMENT_TICKET_REFERENCE, ticketReference);
    context.merge(PATH_COMMENT_DATE, date);
    context.merge(PATH_COMMENT_USER_UPDATE, userUpdate);
    context.merge(PATH_COMMENT_NEW_ETR, newEtr);
    context.merge(PATH_COMMENTS_OBSERVATION, observations);
    context.merge(PATH_COMMENTS_CAUSE, cause);

    boolean hasObservations = false;
    boolean hasCause = false;

    if(!Strings.isNullOrEmpty(observations) && !observations.equals("n/d")){
     hasObservations = true;
    }

    if(!Strings.isNullOrEmpty(cause) && !cause.equals("n/d")){
        hasCause = true;
    }

    context.merge(PATH_OBSERVATIONS_FIELD_IS_NULL_OR_EMPTY, !hasObservations);
    context.merge(PATH_CAUSE_FIELD_IS_NULL_OR_EMPTY, !hasCause);

    boolean hasObservationsOrCause = hasObservations || hasCause;
    DataReportCommonHelper.setUseCasesDataReport(context,  name: "hasObservationsOrCause" , hasObservationsOrCause);
    return hasObservationsOrCause;
}
```

Figura 17 - Método checkForTicketAndDateResolution

Neste método Figura 18, analisa se a resposta que esta no *Json* para verificar se o ticket já foi tratado, extrai o valor booleano associado á chave "*exists*", e de seguida atualiza o relatório.

```
public static boolean checkIfTicketIsTreated(WorkflowContextWrapper context, String response){
    JSONObject res = new JSONObject(response);
    boolean isTreated = res.optBoolean( key: "exists");

    DataReportCommonHelper.setUseCasesDataReport(context,  name: "isTreated" , isTreated);
    return isTreated;
}
```

Figura 18 - Método checkIfTicketIsTreated

Neste último método Figura 19, vai se processar uma lista de comentários, para verificar se existe mais algum comentário disponível para ser tratado. O contexto é atualizado com o novo *index* e informa sobre a existência de mais comentários por meio de um relatório de dados.

```
public static boolean checkForMoreComments(WorkflowContextWrapper context){
    int index = context.getValue(PATH_COMMENTS_INDEX);
    index++;
    context.merge(PATH_COMMENTS_INDEX, index);
    JSONArray comments = new JSONArray(context.getValue(PATH_COMMENTS_LIST).toString());
    boolean thereAreMoreComments = index < comments.length();

    DataReportCommonHelper.setUseCasesDataReport(context,  name: "thereAreMoreComments" , thereAreMoreComments);
    return thereAreMoreComments;
}
```

Figura 19 – checkForMoreComments


```
{
  "Incident_ID_A3": "TT_00453018/2023",
  "New_ETR": "2021-01-01 20:00:00.0",
  "Observations": "observation 1",
  "date": "2021-01-01 20:00:00.0",
  "USER_UPDATE": "user1",
  "SIGNET": "false",
  "Comentario": "comentario 1",
  "CAUSA": "n/d"
}
```


Figura 20 - Exemplo do Ficheiro Json

6.3 Desenvolvimento das Constants

Neste tópico vão ser apresentadas todas as constantes que foi necessário implementar no desenvolvimento do projeto. Na Figura 21 vão ser apresentados os *Paths*, na Figura 22 as *Activities* e na Figura 23 as *Decisions*.

Na figura 21, são criadas *Strings* que representa o caminho de cada *path*, dependendo da fase de analise onde o *workflow* se situa.

```
package com.alticelabs.maestroalb.workflows.helpers.workflow.ter.comments;

import com.alticelabs.maestroalb.constants.WFCommonConstants;

49 usages
public class WFTerCommentsConstants extends WFCommonConstants {
    no usages
    private WFTerCommentsConstants () {
    }
    //---- paths----------------------------------------------------------------------------------------------------
    15 usages
    public static final String WF_TER_COMMENTS = BASE_PATH+"/terComments";
    6 usages
    public static final String PATH_CAUSE_FIELD_IS_NULL_OR_EMPTY = WF_TER_COMMENTS+"/causeFieldIsNullOrEmpty";
    6 usages
    public static final String PATH_OBSERVATIONS_FIELD_IS_NULL_OR_EMPTY = WF_TER_COMMENTS+"/observationsFieldIsNullOrEmpty";
    6 usages
    public static final String PATH_COMMENTS_INDEX = WF_TER_COMMENTS+"/commentsIndex";
    4 usages
    public static final String PATH_COMMENTS_LIST = WF_TER_COMMENTS+"/commentsList";
    7 usages
    public static final String PATH_COMMENT_TICKET_REFERENCE = WF_TER_COMMENTS+"/commentTicketReference";
    5 usages
    public static final String PATH_COMMENT_DATE = WF_TER_COMMENTS+"/commentDate";
    4 usages
    public static final String PATH_COMMENT_USER_UPDATE = WF_TER_COMMENTS+"/commentUserUpdate";
    4 usages
    public static final String PATH_COMMENT_NEW_ETR = WF_TER_COMMENTS+"/commentNewEtr";
    4 usages
    public static final String PATH_COMMENTS_OBSERVATION = WF_TER_COMMENTS+"/commentsObservation";
    4 usages
    public static final String PATH_COMMENTS_CAUSE = WF_TER_COMMENTS+"/commentsCause";

```

Figura 21 – Paths

Na figura 22, a *Strings* criadas representam a descrição de cada atividade ou sistema, dependendo do contexto.

```
    //---- activities----------------------------------------------------------------------------------------------
    4 usages
    public static final String ACT_GET_COMMENTS = "Get comments";
    5 usages
    public static final String ACT_CHECK_LIST_FOR_DATA = "Check list for data";
    6 usages
    public static final String ACT_CHECK_FOR_TICKET_AND_DATE_RESOLUTION = "Check for ticket and date resolution";
    4 usages
    public static final String ACT_CHECK_FOR_COMMENT_AND_CAUSE = "Check for comment and cause";
    5 usages
    public static final String ACT_CHECK_IF_TICKET_IS_TREATED = "Check if ticket is treated";
    5 usages
    public static final String ACT_UPDATE_REPORT = "Update report";
    5 usages
    public static final String ACT_CHECK_FOR_NEXT_COMMENT = "Check for next comment";
    4 usages
    public static final String ACT_UPDATE_SIGO_TICKET_OBSERVATION = "Update sigo ticket observation";
    4 usages
    public static final String ACT_UPDATE_SIGO_TICKET_CAUSE = "Update sigo ticket cause";
    3 usages
    public static final String ACT_SAVE_IDENTIFIER = "Save identifier";
```

Figura 22 – Activities

Na figura 23, são declaradas *Strings* que representam a descrição de cada decisão.

Figura 23 - Decisions

6.4 Desenvolvimento do SystemHelper

Na figura 24 foi declarado um método chamado *getComments*. É criado um objeto chamado *getCommentsRequestBuilder* usando um *helper* chamado *k2viewHelper* para obter comentários ou incidentes.

```
public static SessionManagerPluginRequest getComments(WorkflowContextWrapper context){
    RequestBuilder getCommentsRequestBuilder = K2viewHelper.getIncidents(context);
    return new SessionManagerPluginRequest(getCommentsRequestBuilder.build());
}
```

Figura 24 - Método getComments

Este método Figura 25, verifica comentários e causas relacionadas a um *ticket* ou incidente em uma data especifica.

```
public static SessionManagerPluginRequest checkForCommentAndCause(WorkflowContextWrapper context){
    String ticketReference = context.getValue(PATH_COMMENT_TICKET_REFERENCE);
    String date = context.getValue(PATH_COMMENT_DATE);

    RequestBuilder checkForCommentAndCauseRequestBuilder = ScannerHelper.checkTicketDateInfo(context, ticketReference, date);
    return new SessionManagerPluginRequest(checkForCommentAndCauseRequestBuilder.build());
}
```

Figura 25 - Método checkForCommenAndCause

Na Figura 26 o método da classe *ScannerHelper*, está a configurar um *RequestBuilder* para realizar uma solicitação http relacionada à verificação de informações do *ticket* e da data em uma URL especifica. Primeiro começa por criar um *RequestBuilder*, a seguir obtêm se a base URL usando um método da classe *ScannerRequestHelper*. Cria se um objecto *JSON* para conter os dados do ticket e da data, de seguida configura o *RequestBuilder* com a URL e o conteúdo que está contido no *JSON*.

```
public static RequestBuilder checkTicketDateInfo(WorkflowContextWrapper context, String ticketReference,
                                                  String date) {
  RequestBuilder getRequestBuilder = ScannerRequest.getRequestBuilder();

  String baseUrl = ScannerRequestHelper.getBaseUrl(context) + "/check-ticket-date-info";
  JSONObject content = new JSONObject();
  content.put("ticket", ticketReference);
  content.put("date", date);

  try {
    getRequestBuilder
        .url(baseUrl)
        .content(content.toString());
  } catch (Exception e) {
    LOGGER.error("Error while generating request. Exception: " + e.getMessage(), e);
  }
  return getRequestBuilder;
}
```

Figura 26 – Método checkTicketDateInfo

No método da Figura 27, vamos obter o valor o *ticketReference* e a sua data dos seus caminhos associados respetivamente "*PATH_CONTEXT_TICKET_REFERENCE*" e "*PATH_COMMENT_DATE*". De seguida é criado um *RequestBuilder* usando um método da classe *ScannerHelper* apresentado na figura 23, retornando uma instância com base no *RequestBuilder*, guardando assim os dados.

```
public static SessionManagerPluginRequest saveIdentifier(WorkflowContextWrapper context){
    String ticketReference = context.getValue(PATH_COMMENT_TICKET_REFERENCE);
    String date = context.getValue(PATH_COMMENT_DATE);

    RequestBuilder checkForCommentAndCauseRequestBuilder = ScannerHelper.saveTicketDateInfo(context, ticketReference, date);
    return new SessionManagerPluginRequest(checkForCommentAndCauseRequestBuilder.build());
}
```

Figura 27 - Método saveIdentifier

Na Figura 28 o método da classe *ScannerHelper*, é um método que está a configurar um *RequestBuilder* para realizar uma solicitação http relacionada ao registo de informações do ticket e da data em uma URL especifica. Primeiro começa por criar um *RequestBuilder*, a seguir obtém se a base URL usando um método da classe *ScannerRequestHelper*. Cria se um objeto *JSON* para conter os dados do *ticket* e da data, de seguida configura o *RequestBuilder* com a URL e o conteúdo que está contido no *JSON*.

Figura 28 - Método saveTicketDataInfo

Na figura 29, obtêm se a base URL através do método '*getBaseUrl*', para de seguida se fazer a construção da URL completa, concatenando a base URL com "/*wsTer*". O tipo de pedido é um *get* para o *wsTER*, do *endpoint* que se chama Co.

Figura 29 - Método getIncidents

Este método da Figura 30, está a configurar um *RequestBuilder* para realizar uma solicitação HTTP, incluindo a autenticação com um *token* no cabeçalho "*Authorization*". Primeiro é inicializado uma variável *token*, a seguir tenta obter um *token* usando o método da classe *k2viewRequestHelper*, retornando uma instância configurada.

Ou seja, o *RequestBuilder* apresentado já tem o *authorization* e o *accept*, isto é, já tem grande parte dos dados que necessitamos por isso na Figura 27, só precisamos de inserir o que precisamos para o *endpoint* que é "/*wsTER*".

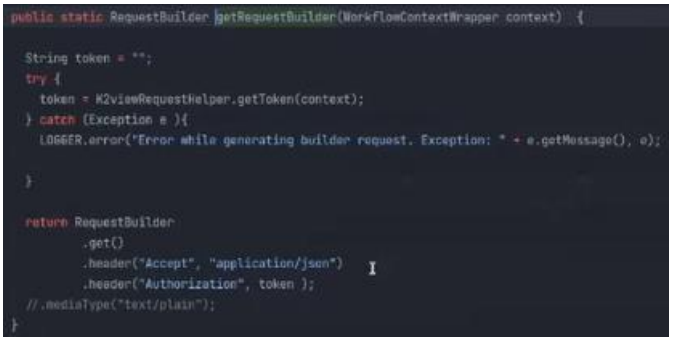


Figura 30 - Método getRequestBuilder

O método criado "*updateTicketObservation*" Figura 31, cria uma solicitação para atualizar uma observação em um *ticket*.

Em primeiro lugar obtêm se a data e hora atuais no fuso horário especificado, de seguida formata se a data e hora atual conforme o padrão especificado. Obtém se os valores relevantes do contexto, como *ticketReference*, *userUpdate*, *newEtr* e *observation*.

São criados uns *arrays* e objetos *Json* para representar os comentários e o conteúdo da solicitação. Uma *String* de texto e construída para o comentário com base nos valores do contexto, configurando informações relevantes para o comentário, como criticidade, texto, proprietário, tipo e data. Com base nos *arrays* e objetos *Json* criados é configurado o conteúdo da solicitação.

```
1 usage   Joao Paulo
public static SessionManagerPluginRequest updateTicketObservation(WorkflowContextWrapper context){
    LocalDateTime now = LocalDateTime.now(ZoneId.of(ZONE_ID));
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern( s: "yyyy-MM-dd HH:mm:ss.SSSSSS");
    String ticketReference = context.getValue(PATH_COMMENT_TICKET_REFERENCE);
    String userUpdate = context.getValue(PATH_COMMENT_USER_UPDATE);
    String newEtr = context.getValue(PATH_COMMENT_NEW_ETR);
    String observation = context.getValue(PATH_COMMENTS_OBSERVATION);
    JSONArray comments = new JSONArray();
    JSONObject comment = new JSONObject();
    String text = WF_TER_COMMENTS + " - Updated by " + userUpdate + "|Update: " + observation + "| New TER: " + newEtr;
    comment.put("commentCriticality", "N");
    comment.put("commentText", text);
    comment.put("commentOwner", "MaestroALB");
    comment.put("commentTypeId", "Tarefa");
    comment.put("commentDate", now.format(formatter));
    comments.put(comment);
    JSONObject content = new JSONObject();
    content.put("causeId", "");
    content.put("ciName", "");
    content.put("ciTypeId", "");
    content.put("resolutionEstimatedDate", newEtr);
    content.put("comments", comments);

    RequestBuilder updateTicketRequestBuilder = SigoHelper.updateTicketRequestBuilder(context, ticketReference, content);
    return new SessionManagerPluginRequest(updateTicketRequestBuilder.build());
}
```

Figura 31 - Método updateTicketObservation

O método criado na Figura 32 "*updateTicketObservation*", cria uma solicitação para atualizar a causa em um *ticket*, adicionado um comentário relacionado à causa.

Em primeiro lugar obtêm se a data e hora atuais no fuso horário especificado, de seguida formata se a data e hora atual conforme o padrão especificado. Obtém se os valores relevantes do contexto, como *ticketReference*, e a causa associada.

São criados uns *arrays* e objetos *Json* para representar os comentários e o conteúdo da solicitação. Uma String de texto e construída para o comentário com base nos valores do contexto, configurando informações relevantes para o comentário, como criticidade, texto, proprietário, tipo e data. Com base nos *arrays* e objetos *Json* criados é configurado o conteúdo da solicitação.

```
1 usage   Joao Paulo
public static SessionManagerPluginRequest updateTicketCause(WorkflowContextWrapper context){
    LocalDateTime now = LocalDateTime.now(ZoneId.of(ZONE_ID));
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern( s: "yyyy-MM-dd HH:mm:ss.SSSSSS");
    String ticketReference = context.getValue(PATH_COMMENT_TICKET_REFERENCE);
    String cause = context.getValue(PATH_COMMENTS_CAUSE);
    JSONArray comments = new JSONArray();
    JSONObject comment = new JSONObject();
    String text = WF_TER_COMMENTS + " - Causa SIGNET: " + cause;
    comment.put("commentCriticality", "N");
    comment.put("commentText", text);
    comment.put("commentOwner", "MaestroALB");
    comment.put("commentTypeId", "OBSSIGNET");
    comment.put("commentDate", now.format(formatter));
    comments.put(comment);
    JSONObject content = new JSONObject();
    content.put("causeId", "");
    content.put("ciName", "");
    content.put("ciTypeId", "");
    content.put("comments", comments);

    RequestBuilder updateTicketRequestBuilder = SigoHelper.updateTicketRequestBuilder(context, ticketReference, content);
    return new SessionManagerPluginRequest(updateTicketRequestBuilder.build());
}

}
```

Figura 32 - Método updateTicketCause

7. Verificação e Validação

Os testes de *software* são processos para avaliar a funcionalidade da aplicação ou de uma parte dessa aplicação com o intuito de descobrir se o software está a funcionar, se cumpre os requisitos como o pretendido ou não.

Para testar o código feito foram feitos testes unitários. O teste unitário é uma prática de desenvolvimento de *software* na qual, partes especificas do código são testadas de forma isolada para garantirem que funcionem conforme o esperado.

Para a realização destes testes foi utilizado o *Fulfillment One*.

O *workflow* era acionado por um endpoint, que se refere a um URL específico que indica um local onde uma *API* pode ser acessada, onde era enviada uma *service order* com a indicação do *Workflow* que se ia testar.

Service order é um documento ou registo usado para registar e acompanhar o trabalho a ser realizado. Uma servisse order contem informações relevantes para um serviço ou tarefa especifica, como: detalhes do cliente, descrição do serviço, datas e prazos, facturamento e instruções e notas adicionais.

No decorrer do *workflow* invocávamos outros *endpoints* para testes que muitas vezes simulávamos as respostas para testar os vários "caminhos/pontos" dos *workflows*, podendo assim verificar o processo todo do *workflow*.

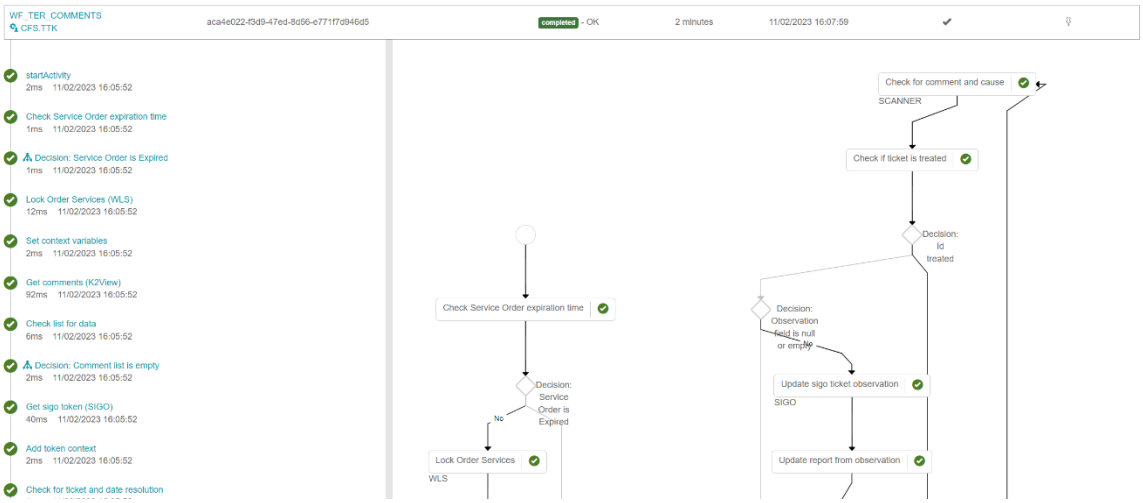


Figura 33 - Teste 1/3

Na figura 33, o início do teste é verificado a hora de expiração da *service order*, com uma decisão a seguir com um sim ou não, para verificar se já expirou ou não. Se a resposta for “sim”, acaba de imediato todo o processo, o que não aconteceu neste caso, que a resposta foi um “não”. Sendo a resposta um “não”, vão se definir as variáveis do contexto obtendo assim os comentários como mostra na Figura 34. De seguida é feita a verificação da lista de dados passando para a decisão que verifica se a lista de comentários está vazia ou não. Neste caso a resposta foi “não”, como mostra a Figura 34. De seguida obtém-se o token pelo sistema SIGO e logo de seguida é adicionado, Figura 35.

O próximo passo é feito uma verificação a resolução dos *tickets* e suas datas passando para a decisão que vai verificar se tem observações ou causas, onde obtivemos a resposta sim, como mostra na Figura 35.

Voltando a Figura 33, o Scanner vai verificar o comentário e a causa e verifica de seguida se já foi tratado ou não. Neste caso já tinha sido tratado, o que faz com que siga logo para a criação de um relatório, Figura 34.

É feita uma verificação para uma possível existência de mais observações, onde vamos ter uma decisão logo a seguir para responder se existem mais observações ou não. Neste caso a resposta foi não, como mostra na Figura 35, fazendo assim com que todo o processo acabe e fique finalizado a analise este teste.

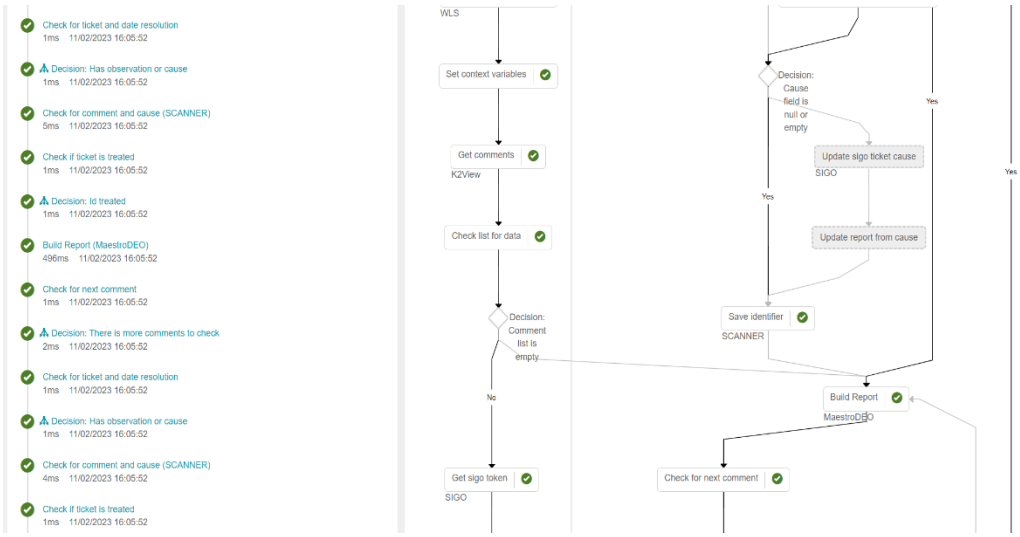


Figura 34 - Teste 2/3

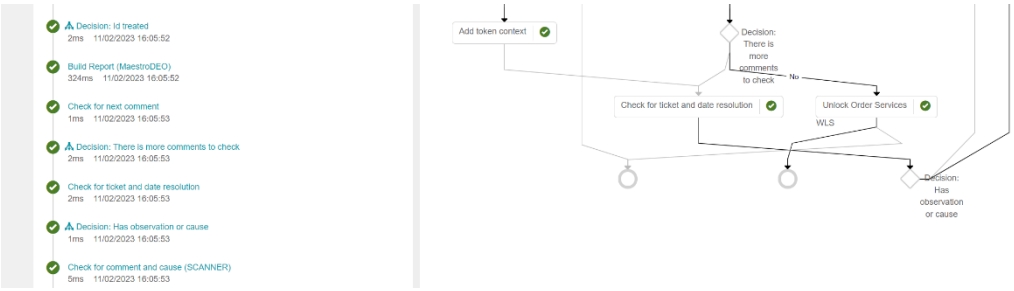


Figura 35 – Teste 3/3

8. Conclusão

A automação de *tickets* na empresa de telecomunicações MEO vai representar uma evolução significativa no suporte do cliente, otimizado processos e aumentando a eficiência operacional.

Ao longo deste trabalho, exploramos como a MEO implementou avançadas soluções de automação para a gestão de *tickets*, desde a abertura até a sua resolução, proporcionando uma experiência mais ágil aos clientes.

A concretização deste projeto espera revelar melhorias notáveis na eficiência e velocidade do suporte, tentando reduzindo tempos de resposta e minimizando intervenções manuais.

A MEO pretende demonstrar um compromisso claro com a melhoria continua da experiência do cliente, simplificando processos, procurando dar respostas mais rápidas e eficazes, identificando áreas de aperfeiçoamento por meio de analises detalhadas e relatórios.

A minha parte da aplicação foi desenvolvida com sucesso, tendo assim cumprido todos os requisitos com sucesso.

Bibliografia

[1] “Ágil, Desenvolvimento,” [Online]. Available: http://www.desenvolvimentoagil.com.br/scrum/.

[2] “ZENDESK, “Entenda quais são os 4 valores e os 12 principios ágeis do Manifesto Ágil para desenvvolvimento de projetos”,” 2022. [Online]. Available: https://www.zendesk.com.br/blog/principios-ageis/.

[3] “Compose, Use Docker,” [Online]. Available: https://docs.docker.com/get-started/08_using_compose/.

[4] “DevOps, Docker for the Absolute Beginner - Hands On -,” [Online]. Available: https://www.udemy.com/course/learn-docker/#overview.

[5] “Git Complete: The definitive, step-by-step guide to Git,” [Online]. Available: https://www.udemy.com/course/git-complete/#overview.

[6] “Introduction, Git and GitHub,” [Online]. Available: https://www.w3schools.com/git/git_intro.asp?remote=github.

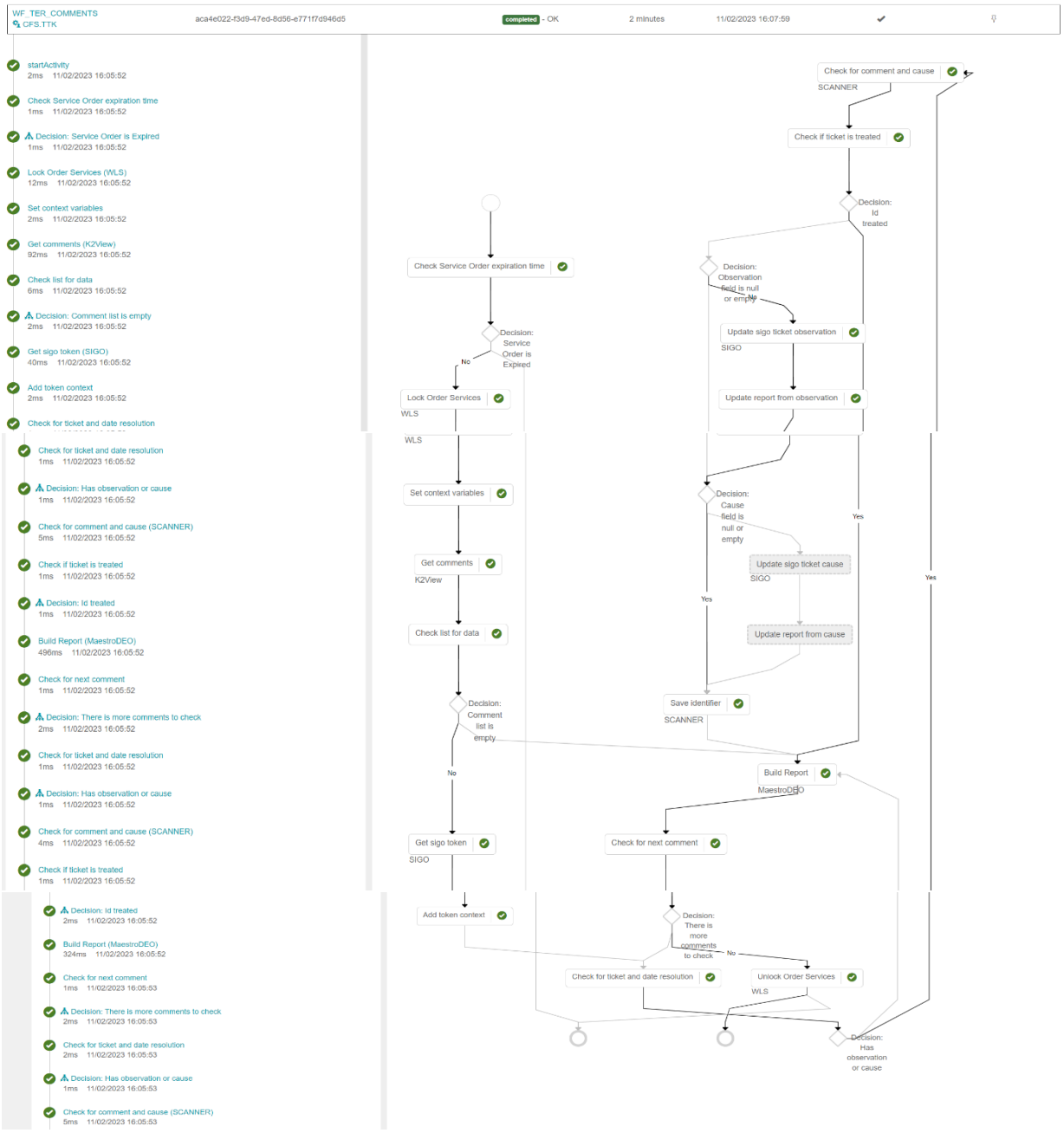
[7] “Introduction, Java,” [Online]. Available: https://www.w3schools.com/java/java_intro.asp.

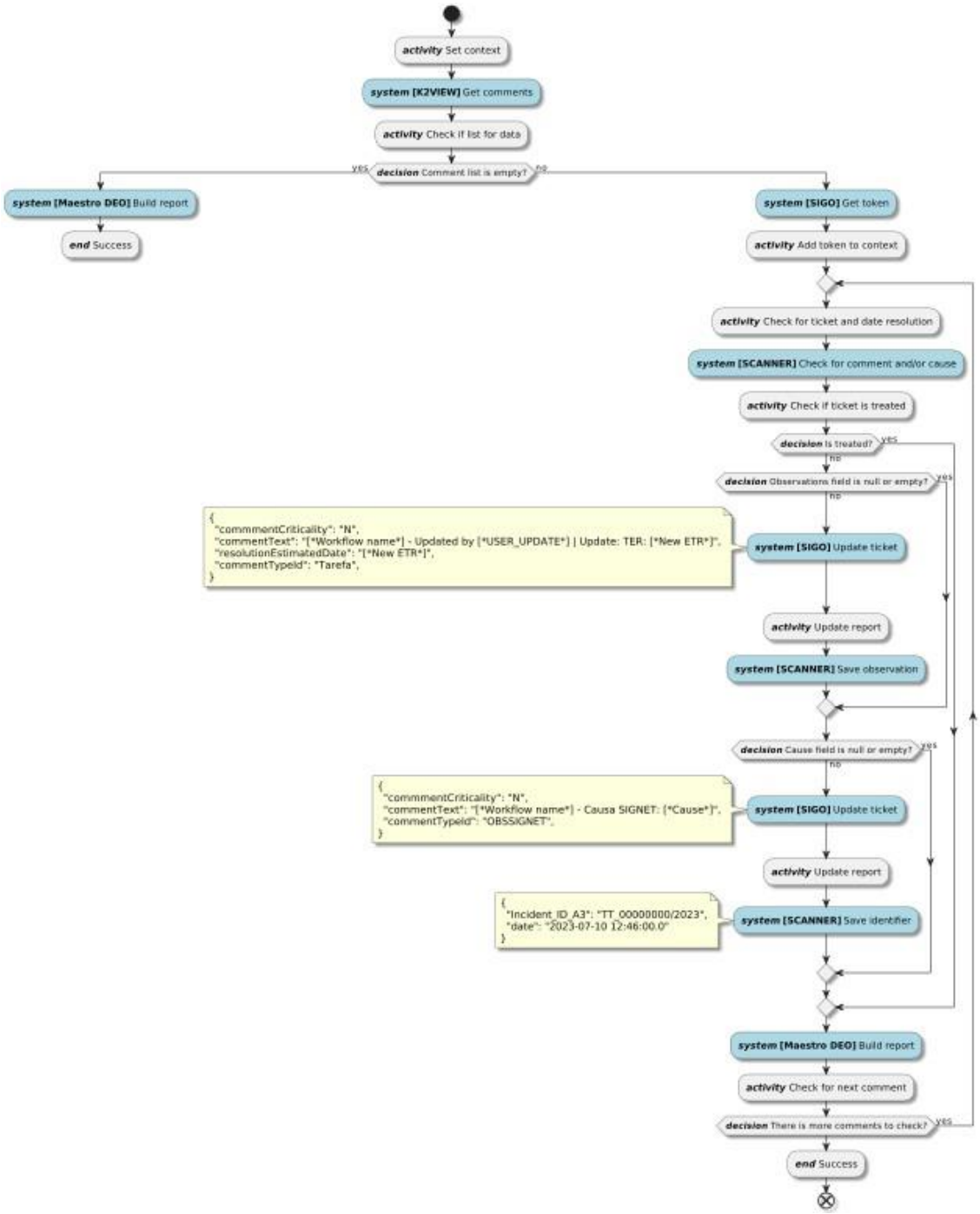
[8] “Introduction, Kotlin,” [Online]. Available: https://www.w3schools.com/kotlin/kotlin_intro.php.

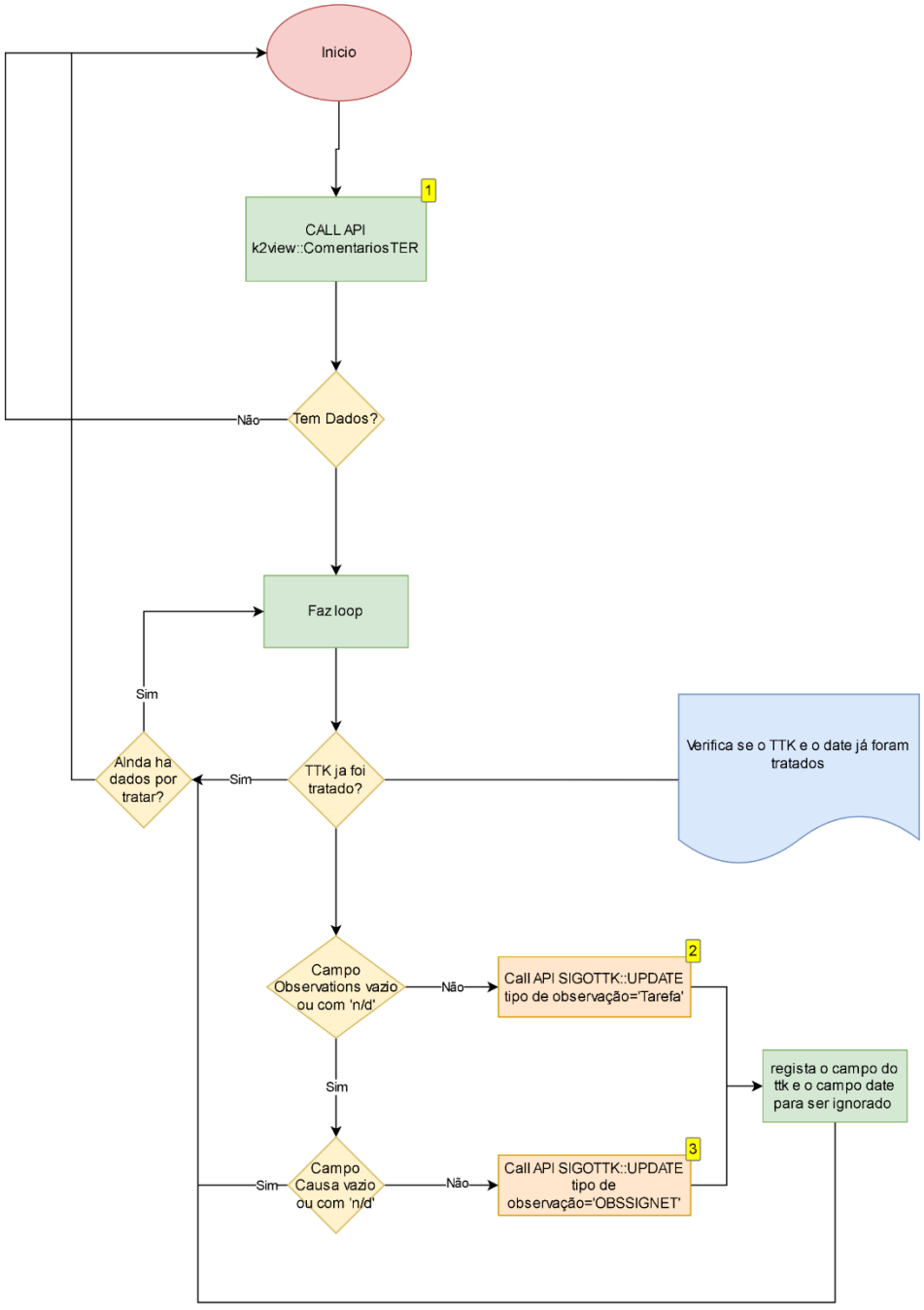
[9] “workflow?, O que é,” [Online]. Available: https://www.growunder.com/pt/blog/dicas/297-o-que-e-workflow-como-fazer-e-porque.

Anexos

1 Diagramas




activity Set context
system [K2VIEW] Get comments
activity Check if list for data
decision Comment list is empty?
yes
no
system [Maestro DEO] Build report
end Success
system [SIGO] Get token
activity Add token to context
activity Check for ticket and date resolution
system [SCANNER] Check for comment and/or cause
activity Check if ticket is treated
decision Is treated?
decision Observations field is null or empty?
{
"commmentCriticality": "N",
"commentText": "[*Workflow name*] - Updated by [*USER_UPDATE*] | Update: TER: [*New ETR*]",
"resolutionEstimatedDate": "[*New ETR*]",
"commentTypeId": "Tarefa",
}
system [SIGO] Update ticket
activity Update report
system [SCANNER] Save observation
decision Cause field is null or empty?
{
"commmentCriticality": "N",
"commentText": "[*Workflow name*] - Causa SIGNET: [*Cause*]",
"commentTypeId": "OBSSIGNET",
}
system [SIGO] Update ticket
activity Update report
{
"Incident_ID_A3": "TT_00000000/2023",
"date": "2023-07-10 12:46:00.0"
}
system [SCANNER] Save identifier
system [Maestro DEO] Build report
activity Check for next comment
decision There is more comments to check?
end Success


Inicio
1
CALL API
k2view::ComentariosTER
Não
Tem Dados?
Faz loop
Sim
Ainda ha dados por tratar?
Sim
TTK ja foi tratado?
Verifica se o TTK e o date já foram tratados
Campo Observations vazio ou com 'n/d'
Não
2
Call API SIGOTTK::UPDATE tipo de observação='Tarefa'
Sim
regista o campo do ttk e o campo date para ser ignorado
Campo Causa vazio ou com 'n/d'
Sim
Não
3
Call API SIGOTTK::UPDATE tipo de observação='OBSSIGNET'