FACULDADE DE TECNOLOGIA DE SÃO JOSÉ DOS CAMPOS
FATEC PROFESSOR JESSEN VIDAL

IGOR MACIEL MACHADO

DESENVOLVIMENTO DE MÓDULO PARA O SISTEMA DE SHOP FLOOR ERT

Orientador: Diogo Branquinho

São José dos Campos
2022

SUMÁRIO

1 INTRODUÇÃO

A *Telefonaktiebolaget LM Ericsson* é uma empresa de tecnologia sueca fabricante de equipamentos de telefonia fixa e móvel. Foi fundada em 1876 e é líder mundial no setor de telecomunicação, presente na vida de cerca de dois bilhões de usuários em todo o mundo, com atuação em 175 países e gerando empregos para mais de 90 mil pessoas.

Uma unidade da *Ericsson Telecomunicações S/A* foi construída em 1954 para a produção de aparelhos e terminais telefônicos, a partir de um projeto de Oscar Niemeyer atendendo aos requisitos funcionais da época e construída em área onde havia sido prevista a localização, em 1947, da área esportiva do CTA.

A indústria fez parte da segunda fase da industrialização de São José dos Campos e foi uma das mais próximas da área urbana do município.

Com a mudança para o distrito de Eugênio de Melo a área foi adquirida para a instalação do shopping Center Vale e parcialmente demolida em 1980, perdendo suas características arquitetônicas.

Por sua proximidade com a via Dutra, São José foi escolhida para sediar a primeira fábrica da Ericsson no Brasil, conhecida como “Fábrica dos Telefones”. Tudo porque, no prédio inaugurado em 1955, o grande letreiro indicava “Telefones Ericsson”.

Hoje, as operações da unidade em São José dos Campos consistem na produção e manutenção de equipamentos de infraestrutura de telefonia móvel, com um quadro de funcionários de aproximadamente mil pessoas.

Tendo em vista esse ambiente de produção, as equipes de engenharia e de tecnologia da informação atuam dando suporte à linha de produção, que não cessa desde 1966. E um dos sistemas responsáveis por esse suporte é o ERT, ou *Ericsson Repair Tracker*.

O ERT é um sistema de *shop floor* que atua no controle de entrada e saída de equipamentos de telecomunicações em centros de reparo ao redor do mundo, estando atualmente implantado em 4 países diferentes: Brasil, Malásia, China e Estados Unidos, com alguns outros países para serem adicionados nessa lista. Seu principal objetivo é fazer o controle dos equipamentos e acompanhá-lo durante o processo de avaliação e conserto, desde o momento em que chegam no centro de reparos até o momento de saída.

1.1 Definição do problema

Neste cenário, apresentou-se a necessidade de *business* de saber, com precisão, quanto tempo cada uma das etapas do processo estava consumindo, para que fosse possível ter um controle preciso e acirrado do período de tempo gasto em cada uma das etapas do processo, *e.g.*, quanto tempo era consumido em uma etapa do processo de *troubleshooting*, *unit testing*, *hardware logging*, reparo, etc. Se houvesse um aumento muito grande em determinada etapa, seria mais fácil e rápido a verificação e identificação deste problema.

1.2 Objetivo

O objetivo deste trabalho foi o desenvolvimento de uma *feature* do sistema ERT para cumprir a necessidade de *business* para melhor controle dos processos e equipamentos dos centros de reparo.

2 DESENVOLVIMENTO

Este capítulo apresenta os detalhes de desenvolvimento do módulo feito para o projeto ERT, usando as tecnologias *.net C# 4.8*, *Entity Framework*, *Microsoft Server*, *Java Script* e *SMTP*.

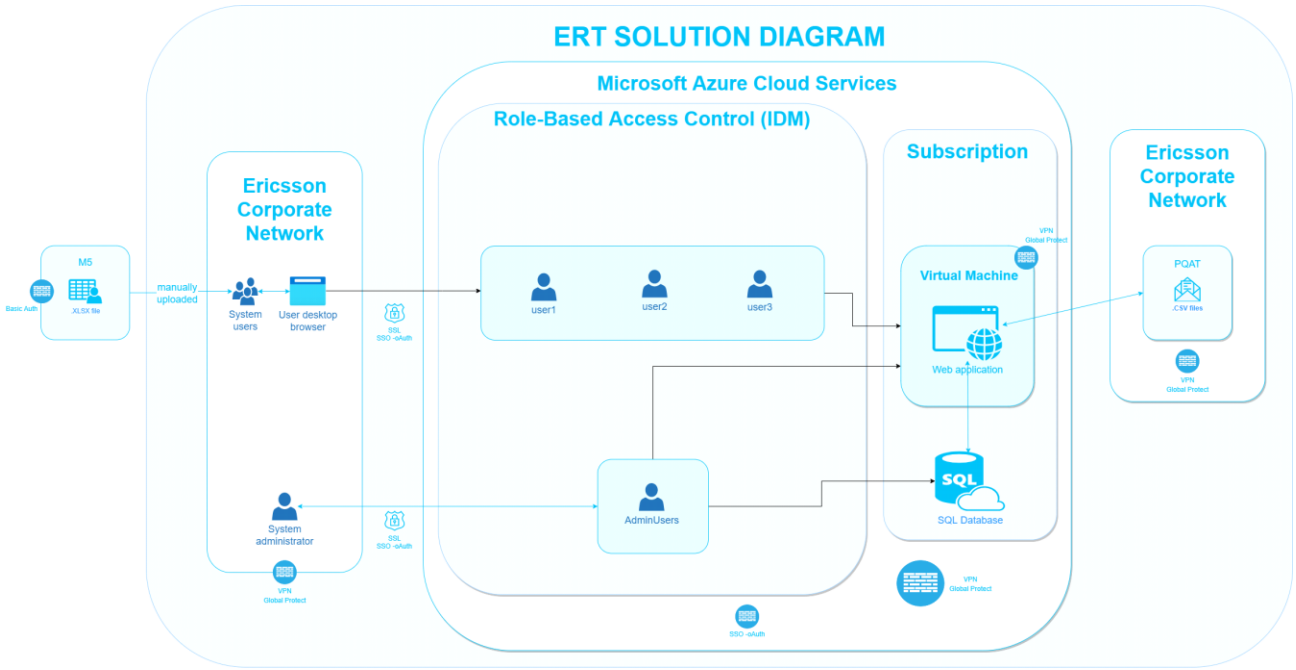
2.1 Arquitetura

A **Figura 1** ilustra a arquitetura geral do sistema. A seguir são apresentados seus elementos:

- **Ericsson Corporate Network -** Rede corporativa interna da *Ericsson* (Intranet).
- **System Users -** Usuários de sistema da rede corporativa.
- **User Desktop browser -** Navegador dos computadores com acesso à rede corporativa.
- **System Administrator -** Usuário com permissões de administrador dentro da rede corporativa.
- **VPN Global Protect -** Serviço de *VPN* de proteção de acesso.
- **SSL -** Certificado aplicado para garantir a conexão segura via *https*.
- **SSO -** *Single Sign-on*, ou usuário único de acesso, que permite que o funcionário use o mesmo login e senha para diferentes sistemas no ambiente corporativo.
- **-oAuth -** Padrão de autenticação que define quais tipos de contas podem ser usados para acessar diferentes serviços, e.g., uma conta do *Google*.
- **User (1, 2 e 3) -** Diferentes usuários do sistema acessando-o de diferentes lugares no mundo.
- **Microsoft Azure Cloud Services -** Serviço de computação em nuvem operado pela *Microsoft* para gerenciamento de aplicativos por meio de data centers.
- **Role-Based Access Control (IDM) -** Serviço de controle de acesso baseado em permissões e perfis de usuários, separados por grupos de acesso.
- **Subscription -** Inscrições dentro do serviço *Azure*.
- **Virtual Machine -** Máquina virtual rodando dentro do ambiente de *cloud* da *Azure*.

- **Web Aplication -** Aplicação *Web*.
- **SQL Database -** Banco de Dados *SQL*.
- **PQAT –** *Product Quality Actual Time*, módulo desenvolvido para o sistema.

Figura 1 - Arquitetura do Sistem



2.2 Detalhes

Esta seção apresenta trechos do código escrito e partes do desenvolvimento do módulo para a aplicação.

A **Figura 2** exibe o método principal para fazer o envio de PQAT através de e-mail. O Arquivo pode ser visualizado antes de ser enviado.

Figura 2 – Início do método principal PQAT

```
public ContentResult PQAT(int pqatId, string source = "")
{
    string today = DateTime.Now.ToString("yyyyMMdd");

    Site site = db.Site.FirstOrDefault();

    var pqatZipFile = GenerateQualityDataFile(pqatId,
        "QEMS" + site.PqatSiteNumber + "-" + today + "-" + today + ".zip",
        "QEMS" + source + "R" + site.PqatSiteNumber + "-" + today + "-" + today + ".txt",
        "QEMS" + source + "S" + site.PqatSiteNumber + "-" + today + "-" + today + ".txt",
        site.PqatSystemId,
        site.Country,
        site.RepairCenterName);
```

O início do método define a data do dia atual e gera o arquivo *pqatZipFile* através do método chamado *GenerateQualityDateFile* e sua parametrização, que é explicado na **Figura 7**.

Figura 3 – Verificações de consistência para geração do arquivo no método PQAT

```
try
{
    if (site.EmailToPqat == null)
    {
        throw new Exception("There is no e-mail registered to send this data to PQAT. " +
            "Please talk to the System Administrator to fill in this field.");
    }

    var email = new EmailViewModel()
    {
        To = site.EmailToPqat,
        Subject = "ERT QDW " + site.PqatSiteNumber,
        Body = "Hi PQAT Team," + Environment.NewLine + "This file was generated by the delivery number " +
            pqatId.ToString() + "." + Environment.NewLine +
            "Best regards," + Environment.NewLine + "ERT",
        IsBodyHtml = false,
        Attachment = new List<Attachment>()
    };

    if (pqatZipFile != null)
    {
        email.Attachment.Add(new Attachment(new MemoryStream(pqatZipFile.FileContents), pqatZipFile.FileDownloadName));
    }

    EmailHelper.SendEmail(email);
}
catch (Exception ex)
{
    return Content("An error occurred while trying to send an email to PQAT. " +
        "Please contact the administrator reporting the problem: " + pqatId.ToString() + ". Error: " + ex.Message);
}
return Content("The email has been sent!");
```

Caso não exista e-mail registrado, o primeiro laço de verificação exibe a mensagem de aviso para que isso seja feito, exibindo a mensagem "*There is no e-mail registered to send this data to PQAT. Please talk to the System Administrator to fill in this field."* através do tratamento de erro *throw new Exception*. Caso exista, é construído o corpo do e-mail para que essa informação seja enviada, com a concatenação das variáveis *To*, *Subject*, *Body*, *IsBodyHtml* e *Attachment* em seus respectivos campos.

Caso a variável responsável por armazenar o arquivo de *PQAT* gerado não esteja vazia, isto é, caso o arquivo tenha sido gerado com sucesso, ele é anexado ao corpo do e-mail para que seja enviado.

Uma vez que isso ocorra, o e-mail é enviado com o uso do da função *SendEmail* do *EmailHelper*, passando a variável e-mail, que foi construída nos laços acima.

Existe um segundo tratamento de erros através da função *try catch return Content*, com a mensagem de erro *"An error occured while trying to send an email to PQAT. Please Contact the administrator and report the problem"*, para tratar erros de ordens genéricas.

Caso tudo ocorra bem, a mensagem retornada ao método é "*The email has been sent!*".

Figura 4 - Geração do arquivo de PQAT e seu retorno em formato *JSON*

```
[HttpPost]
public JsonResult SentPqatDownload(int pqatId)
{
    var pqat = db.Pqats.Find(pqatId);

    if(pqat == null)
    {
        return null;
    }
    if(pqat.RepairProduct.Count() ==0)
    {
        return null;
    }

    var source = pqat.RepairProduct.FirstOrDefault().FileClaimsId != null ? "C" : "H";

    string today = DateTime.Now.ToString("yyyyMMdd");

    Site site = db.Site.FirstOrDefault();

    string zipName = "QEMS" + site.PqatSiteNumber + "-" + today + "-" + today + ".zip";
    string repairFileName = "QEMS" + source + "R" + site.PqatSiteNumber + "-" + today + "-" + today + ".txt";
    string qualityFileName = "QEMS" + source + "S" + site.PqatSiteNumber + "-" + today + "-" + today + ".txt";
    string systemId = site.PqatSystemId;
    string country = site.Country;
    string sLocation = site.RepairCenterName;

    string fullPath = Path.Combine(Server.MapPath("~/FileDownloads/"), zipName);
    var file = GenerateQualityDataFile(pqatId, zipName, repairFileName, qualityFileName, systemId, country, sLocation);
    using (var exportData = new MemoryStream(file.FileContents))
    {
        FileStream f = new FileStream(fullPath, FileMode.Create, FileAccess.Write);
        exportData.WriteTo(f);
        f.Close();
    }
    return Json(new { fileName = zipName, errorMessage = "" });
}
```

Neste método é gerado o arquivo *PQAT* através da chamada *GenerateQualityDataFile*, que é explicado em maiores detalhes na **Figura 7**, e retornado em formato *JSON* em conjunto com metadados relevantes, como, e.g., *country* como variável para o país e *sLocation* como variável para o centro de reparos em que essa operação está sendo feita. Esse método é chamado quando o usuário deseja baixar localmente o arquivo de *PQAT*, para que ele possa ser verificado.

Figura 5 – Verificações de consistência do método *SentPqatDownload*

```
[HttpPost]
public JsonResult SentPqatDownload(int pqatId)
{
    var pqat = db.Pqats.Find(pqatId);

    if(pqat == null)
    {
        return null;
    }
    if(pqat.RepairProduct.Count() ==0)
    {
        return null;
    }
```

Na **Figura 5** é demonstrado as verificações de consistência. Primeiro, a variável *pqat* recebe o objeto *PQAT* encontrado através de seu número de verificação individual(*pqatId*) e, caso esse objeto não seja encontrado, o método retorna valor nulo para sua chamada e encerra sua execução.

Já no segundo laço condicional, é verificado se o produto existe, i.e., se seu *RepairProduct* é diferente de zero.

Figura 6 - Retorno do arquivo de PQAT em JSON do método *sentPQATDownload*.

```
var source = pqat.RepairProduct.FirstOrDefault().FileClaimsId != null ? "C" : "H";

string today = DateTime.Now.ToString("yyyyMMdd");

Site site = db.Site.FirstOrDefault();

string zipName = "QEMS" + site.PqatSiteNumber + "-" + today + "-" + today + ".zip";
string repairFileName = "QEMS" + source + "R" + site.PqatSiteNumber + "-" + today + "-" + today + ".txt";
string qualityFileName = "QEMS" + source + "S" + site.PqatSiteNumber + "-" + today + "-" + today + ".txt";
string systemId = site.PqatSystemId;
string country = site.Country;
string sLocation = site.RepairCenterName;

string fullPath = Path.Combine(Server.MapPath("~/FileDownloads/"), zipName);
var file = GenerateQualityDataFile(pqatId, zipName, repairFileName, qualityFileName, systemId, country, sLocation);
using (var exportData = new MemoryStream(file.FileContents))
{
    FileStream f = new FileStream(fullPath, FileMode.Create, FileAccess.Write);
    exportData.WriteTo(f);
    f.Close();
}
return Json(new { fileName = zipName, errorMessage = "" });
}
```

Em *source* é verificado a origem deste produto, i.e., *RepairProduct*, com a sigla “C” para *claims*, que são produtos que ainda estão na garantia e *“H”* para “Hardware Service”, que são produtos que estão sendo acionados através do suporte por não estarem mais em garantia.

Também é gerado todas as *strings* de informação e metadados do arquivo de *PQAT*. Seu diretório no servidor é definido com a variável *fullPath* e é chamado o método

GenerateQualityDataFile em que o arquivo é gerado com base nas *strings* definidas acima. Através de *MemoryStream*, é separado um espaço de memória específico para a gravação física do arquivo no servidor.

E por fim é retorna o arquivo *JSON* com todas as informações necessárias através de *return*.

Figura 7 – Geração De Facto do arquivo PQAT pelo método *GenerateQualityDataFile*

```
private FileContentResult GenerateQualityDataFile(int pqatId, string zipName, string repairFileName,
    string qualityFileName, string systemId, string country, string sLocation, List<Guid> model = null)
{
    model = model ?? new List<Guid>();
    var site = db.Site.FirstOrDefault();
    var repairData = db.ProductActions
                    .Where(pa => pa.ActionCode != "A130" && pa.ActionCode != "A131")
                    .Where(pa => pa.RepairProduct.EndDate != null)
                    .Where(pa => pa.RepairProduct.PqatId == pqatId || model.Contains(pa.RepairProduct.RepairProductId))
                    .Where(pa => pa.RepairProduct.Portfolio.Product.Services != Models.Services.NPI)
                    .SelectMany(pa => db.ProductActionsComponents
                                        .Where(pac => pac.ProductActionId == pa.ProductActionId)
                                        .DefaultIfEmpty(),
                                    (pa, pac) => new
                                    {
                                        pa,
                                        pac
                                    }
                                )
```

Na **Figura 7** é mostrado em maiores detalhes a geração *De Facto* do arquivo *PQAT*, como é chamado nos métodos anteriores.

Neste trecho é feito a checagem para consistência das regras de negócio, *e.g.*, *Action Codes* diferentes de 130(A130) e 131(A131) e *EndDate* diferente de nulo. Há também um *Join* sendo feito entre as tabelas *ProductActionsComponents* (ou *pac*) e *ProductActions* (*pa*).

Figura 8 – Preenchimento do modelo *PqatExportFile* com informações das tabelas *ProductActionsComponents* e *ProductActions*, continuação da **Figura 7**

```
.Select(res => new PqatExportFile()
{
    ProductActionId = res.pa.ProductActionId,
    SystemId = systemId,
    RecStatus = "C",
    RecType = "CN",
    EmsNumber = res.pa.RepairProduct.OrderNumber,
    Country = res.pa.RepairProduct.IncomingMaterialData.COUNTRY != null ?
        (res.pa.RepairProduct.IncomingMaterialData.COUNTRY.Replace(" ", "").Replace("\t", "") != "" ?
        res.pa.RepairProduct.IncomingMaterialData.COUNTRY.Substring(0, 3) : country) : country,
    CustomerNumber = res.pa.RepairProduct.IncomingMaterialData != null ?
        res.pa.RepairProduct.IncomingMaterialData.WAREHOUSE.Substring(0, 14) : res.pa.RepairProduct.IncomingMaterialClaimsData.Customer.Substring(0, 14),
    Product = res.pa.RepairProduct.Portfolio.ProductNumber,
    RevisionState = res.pa.RepairProduct.Portfolio.RevisionState,
    SerialNumber = res.pa.RepairProduct.SerialNumber,
    ActionCode = res.pa.ActionCode,
    //The best way to do this is changing the structure and save the RepairProductStructureId na ProductActionsComponents
    SProduct = res.pa.SerialNumberAction == res.pa.RepairProduct.SerialNumber ? "" :
               (res.pa.RepairProduct.RepairProductStructure.Any(e => e.SerialNumber == res.pa.SerialNumberAction) ?
                res.pa.RepairProduct.RepairProductStructure.Where(e=>e.SerialNumber ==
                res.pa.SerialNumberAction).FirstOrDefault().ProductNumber.Replace(" ","").Replace("\t", "") : res.pa.OldProductNumber),
    SRevisionState = res.pa.SerialNumberAction == res.pa.RepairProduct.SerialNumber ? "" :
              (res.pa.RepairProduct.RepairProductStructure.Any(e => e.SerialNumber == res.pa.SerialNumberAction) ?
               res.pa.RepairProduct.RepairProductStructure.Where(e => e.SerialNumber ==
               res.pa.SerialNumberAction).FirstOrDefault().RevisionState.Replace(" ", "").Replace("\t", "") : res.pa.OldRevisionState),
    RepairCode = res.pac.RepairCode == null ? "" : res.pac.RepairCode.ToString(),
    MajorFault = res.pac.SolvingAction == null ? " " : (bool)res.pac.SolvingAction ? "Y" : "N",
    SSerialNumber = res.pa.SerialNumberAction != res.pa.RepairProduct.SerialNumber ? res.pa.SerialNumberAction : "",
    Component = res.pa.ActionCode.Equals("A130") || res.pa.ActionCode.Equals("A131") ? null : res.pac.Components.Name,
    CPosition = res.pa.ActionCode.Equals("A130") || res.pa.ActionCode.Equals("A131") ? null : res.pac.Position.Replace(" ", "").Replace("\t", ""),
    RepairDate = res.pa.RepairProduct.EndDate.ToString(),
    SLocation = res.pa.RepairProduct.Portfolio.HwsScreening ? site.ScreeningName : site.RepairCenterName,
    RFin = "Y",
    NSerialNumber = res.pa.NewSubUnit,
    FcoRco = res.pa.RcoNumber != null ? res.pa.RcoNumber.Number + "_" + res.pa.RcoNumber.Revision : res.pa.RcoDocument,
    DescripF = res.pa.PqatDescription,
    NProduct = res.pa.NewProductNumber,
    NRevisionState = res.pa.NewRevisionState
})
.ToList();
```

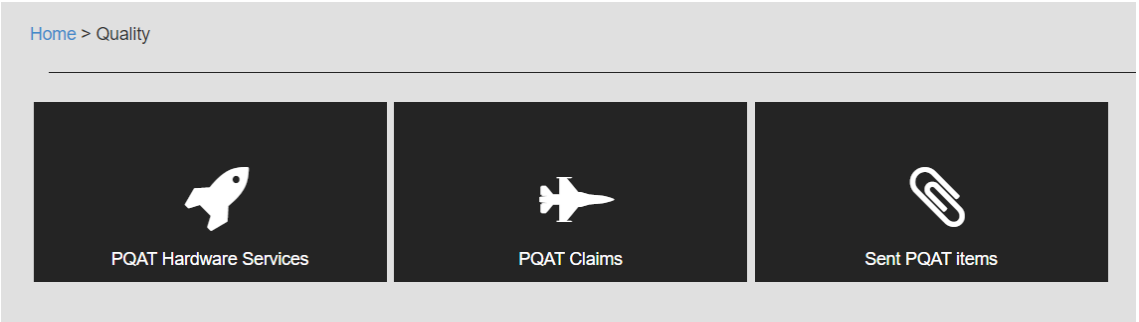
Na **Figura 8**, os dados relevantes para o preenchimento do modelo são coletados das tabelas *ProductAction* e *ProductActionComponents*, trazendo todas *actions* que tenham apontamento de componentes.

Figura 9 - Preenchimento do modelo *PqatExportFile* com informações das tabelas *ProductActionsComponents* e *ProductActions*, continuação da **Figura 8**

```
var list_a130_a131 = db.ProductActions
                .Where(pa => pa.ActionCode == "A130" || pa.ActionCode == "A131")
                .Where(pa => pa.RepairProduct.EndDate != null)
                .Where(pa => pa.RepairProduct.PqatId == pqatId || model.Contains(pa.RepairProduct.RepairProductId))
                .Where(pa => pa.RepairProduct.Portfolio.Product.Services != Models.Services.NPI)
                .Select(res => new PqatExportFile()
                {
                    ProductActionId = res.ProductActionId,
                    SystemId = systemId,
                    RecStatus = "C",
                    RecType = "CN",
                    EmsNumber = res.RepairProduct.OrderNumber,
                    Country = res.RepairProduct.IncomingMaterialData.COUNTRY != null ?
                        (res.RepairProduct.IncomingMaterialData.COUNTRY.Replace(" ", "").Replace("\t", "") != "" ?
                        res.RepairProduct.IncomingMaterialData.COUNTRY.Substring(0, 3) : country) : country,
                    CustomerNumber = res.RepairProduct.IncomingMaterialData != null ?
                    res.RepairProduct.IncomingMaterialData.WAREHOUSE.Substring(0, 14) : res.RepairProduct.IncomingMaterialClaimsData.Customer.Substring(0, 14),
                    Product = res.RepairProduct.Portfolio.ProductNumber,
                    RevisionState = res.RepairProduct.Portfolio.RevisionState,
                    SerialNumber = res.RepairProduct.SerialNumber,
                    ActionCode = res.ActionCode,
                    RepairCode = "",
                    //The best way to do this is changing the structure and save the RepairProductStructureId na ProductActionsComponents
                    SProduct = res.SerialNumberAction != res.RepairProduct.SerialNumber ? res.RepairProduct.RepairProductStructure
                                .Where(rp => rp.SerialNumber.Equals(res.SerialNumberAction))
                                .Select(rps => rps.ProductNumber).FirstOrDefault().Replace(" ", "").Replace("\t", "") : "",
                    SRevisionState = res.SerialNumberAction != res.RepairProduct.SerialNumber ? res.OldRevisionState : "",//em caso de sub unidade pega a revisão do productActi
                    RepairDate = res.RepairProduct.EndDate.ToString(),
                    SLocation = res.RepairProduct.Portfolio.HwsScreening ? site.ScreeningName : site.RepairCenterName,//usar o screening name ou hs name da tabela Site de acord
                    RFin = "Y",
                    NSerialNumber = res.NewSubUnit,
                    FcoRco = res.RcoNumber != null ? res.RcoNumber.Number + "_" + res.RcoNumber.Revision : res.RcoDocument,
                    DescripF = res.PqatDescription,
                    NRevisionState = res.NewRevisionState == null ? (res.SerialNumberAction != res.RepairProduct.SerialNumber ?
                    res.OldRevisionState : res.RepairProduct.Portfolio.RevisionState) : res.NewRevisionState,
                    SSerialNumber = res.SerialNumberAction != res.RepairProduct.SerialNumber ? res.SerialNumberAction : "",
                    NProduct = res.SerialNumberAction != res.RepairProduct.SerialNumber ? res.RepairProduct.RepairProductStructure
                                .Where(rp => rp.SerialNumber.Equals(res.SerialNumberAction))
                                .Select(rps => rps.ProductNumber).FirstOrDefault().Replace(" ", "").Replace("\t", "") : res.RepairProduct.Portfolio.ProductNumber
                })
                .ToList();
```

A **Figura 9** detalha o *select* exclusivo para *Action Codes* 130 e 131. Isto acontece por necessidade de negócios, pois eles precisam de um documento especial, chamado *RCO*, não coberto no escopo deste relatório.

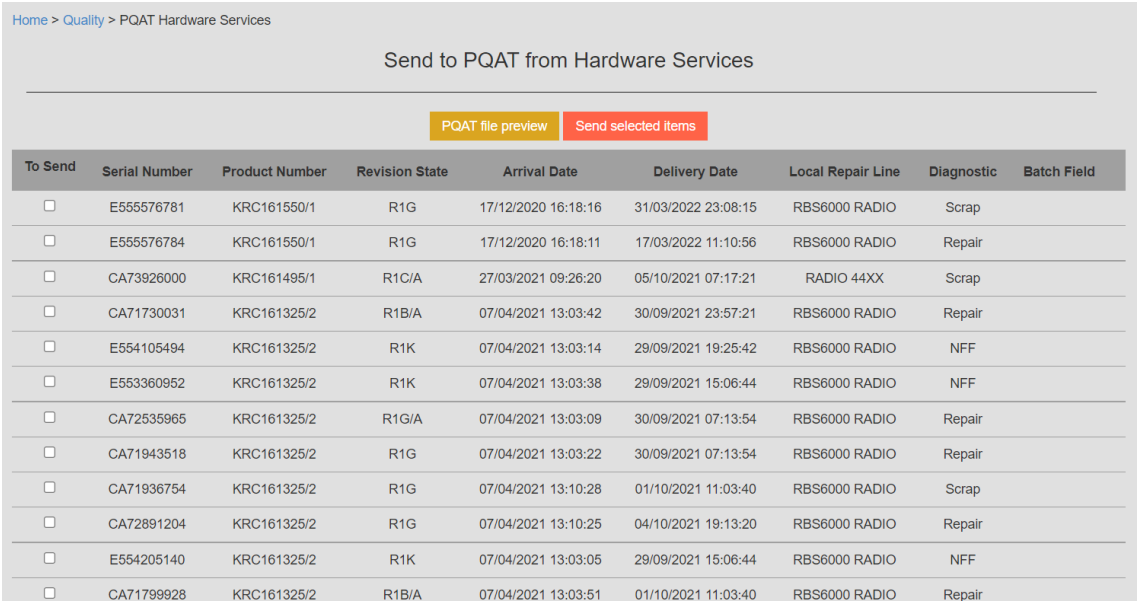
Figura 10 – Menu do Sistema



Menu de qualidade do sistema *ERT*. Na primeira opção, "*PQAT Hardware Services*", é onde o usuário acessa a lista de Produtos que podem gerar os arquivos *PQAT*.

Em "*Sent PQAT Items*" é possível verificar todos os arquivos de PQAT que já foram enviados e baixá-los localmente.

Figura 11 – Tela ao selecionar a opção "*PQAT Hardware Services*"

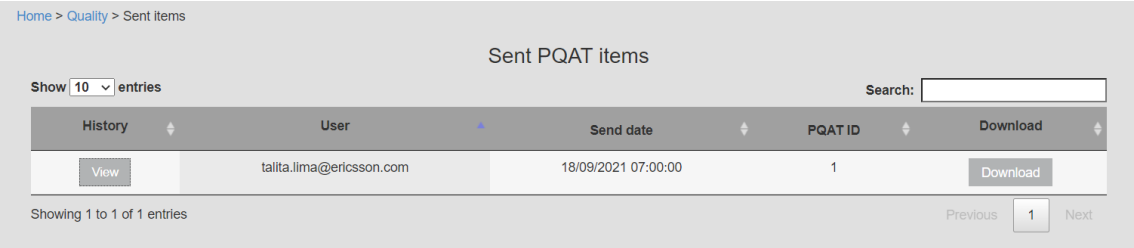


To Send	Serial Number	Product Number	Revision State	Arrival Date	Delivery Date	Local Repair Line	Diagnostic	Batch Field
☐	E555576781	KRC161550/1	R1G	17/12/2020 16:18:16	31/03/2022 23:08:15	RBS6000 RADIO	Scrap	
☐	E555576784	KRC161550/1	R1G	17/12/2020 16:18:11	17/03/2022 11:10:56	RBS6000 RADIO	Repair	
☐	CA73926000	KRC161495/1	R1C/A	27/03/2021 09:26:20	05/10/2021 07:17:21	RADIO 44XX	Scrap	
☐	CA71730031	KRC161325/2	R1B/A	07/04/2021 13:03:42	30/09/2021 23:57:21	RBS6000 RADIO	Repair	
☐	E554105494	KRC161325/2	R1K	07/04/2021 13:03:14	29/09/2021 19:25:42	RBS6000 RADIO	NFF	
☐	E553360952	KRC161325/2	R1K	07/04/2021 13:03:38	29/09/2021 15:06:44	RBS6000 RADIO	NFF	
☐	CA72535965	KRC161325/2	R1G/A	07/04/2021 13:03:09	30/09/2021 07:13:54	RBS6000 RADIO	Repair	
☐	CA71943518	KRC161325/2	R1G	07/04/2021 13:03:22	30/09/2021 07:13:54	RBS6000 RADIO	Repair	
☐	CA71936754	KRC161325/2	R1G	07/04/2021 13:10:28	01/10/2021 11:03:40	RBS6000 RADIO	Scrap	
☐	CA72891204	KRC161325/2	R1G	07/04/2021 13:10:25	04/10/2021 19:13:20	RBS6000 RADIO	Repair	
☐	E554205140	KRC161325/2	R1K	07/04/2021 13:03:05	29/09/2021 15:06:44	RBS6000 RADIO	NFF	
☐	CA71799928	KRC161325/2	R1B/A	07/04/2021 13:03:51	01/10/2021 11:03:40	RBS6000 RADIO	Repair	

Nesta tela, o usuário pode selecionar para quais *ProductNumbers* ele pretende gerar os arquivos ao marcar as *checkboxes* que ficam no lado esquerdo da lista. Em *"PQAT file*

preview", o usuário pode pré-visualizá-los e assegurar-se de que não houveram erros na geração.

Figura 12 – Tela ao selecionar a opção "*Sent PQAT Items*"



Ao selecionar a opção "*Sent PQAT Items*", o usuário tem acesso à tela em que é possível verificar todos os arquivos de *PQAT* que foram enviados, assim como baixá-los novamente.

3 RESULTADOS E DISCUSSÃO

Dentre as tecnologias utilizadas neste trabalho, pode-se destacar o uso de *.NET 4.8*, o *framework* da *Microsoft* para desenvolvimento de sistemas e aplicações, e *C#*, a linguagem multiparadigma de tipagem forte, desenvolvida também pela *Microsoft* como parte da plataforma *.NET*. Além disto, para banco de dados foi usado o próprio proprietário da *Microsoft*, o *Microsoft SQL Server*. Também foi necessário o entendimento da arquitetura do sistema para fosse possível ser feito a comunicação *front end to back end*, usando *JavaScript* e *Angular*. *JavaScript* é uma linguagem de programação interpretada estruturada, de *script* em alto nível e com tipagem dicâmica. Em conjunto com *HTML* e *CSS*, é uma das três principais tecnologias da *World Wide Web*. Para persistência de dados, foi utilizado o *Entity Framework*, que é uma das principais ferramentas para este fim presents na plataforma *.NET*. Ela permite que seja feito o mapeamento dos elementos da base de dados para os elementos da aplicação orientada à objetos. Além disto, o sistema possui arquitetura *MVC*, ou *Model-View-Controller*.

O desenvolvimento se deu concomitantemente com diversas outras *features* paralelas e ao longo de vários meses, sendo progressivamente iterados com base nos resultados e demandas das áreas de engenharia. Tendo em vista o escopo do sistema, foi necessária uma compreensão panorâmica de suas funcionalidades para que, mesmo implementado em diversos *sites* ao redor do planeta, não houvesse impacto em nenhum no *shop floor*.

Uma vez desenvolvido e devidamente testado, foi possível atender a demanda e implementado progressivamente através dos serviços de *Cloud Computing* da *Azure*, onde o sistema funciona.

4 CONCLUSÃO

Maior controle de cada uma das etapas de reparo dos equipamentos de telecomunicação se mostrou essencial em todos os centros de reparo ao redor do mundo. Dessa forma, foi possível maior assertividade na otimização da janela de tempo que as diferentes equipes de técnicos de reparo dispõem, dando aos clientes repostas mais rápidas e proativas.