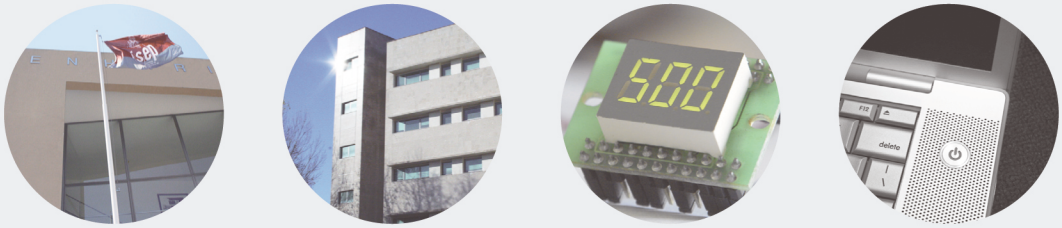
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



Gestão e Atualização Automática de Firmware para Câmaras de Videovigilância em Shop Floor

LUÍS MIGUEL PINTO LISBOA
novembro de 2021

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Departamento de Engenharia Electrotécnica
Rua Dr. António Bernardino de Almeida, 431, P-4200-072 Porto

Gestão e Atualização Automática de Firmware para Câmaras de Videovigilância em Shop Floor

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização em Telecomunicações

Luís Miguel Pinto Lisboa

Orientação: Prof. Maria Benedita Campos Neves Malheiro

Ano Letivo: 2020-2021

Resumo

A Bosch Building Technologies tinha um problema com a atualização do *firmware* das câmaras de videovigilância IP produzidas. Por um lado, a sistemática depuração do código e incorporação de novas funcionalidades nas câmaras obriga à frequente atualização do *firmware*, essenciais ao melhoramento e diferenciação dos produtos no mercado. Por outro lado, a atualização do *firmware*, devido a incompatibilidades entre versões, obriga a reconfigurar o *software* de teste das câmaras, implicando paragens de produção e custos acrescidos. Para contornar este problema, a empresa tinha implementado um sistema manual de atualização de *firmware* nos postos de embalagem. Este processo, dada a diversidade de versões e famílias de produtos, era lento e indutor de erros.

Para colmatar este problema, foi desenvolvido no âmbito deste projeto um novo sistema composto por uma *Application Programming Interface* (API) do tipo *Representational State Transfer* (REST), um serviço de *back-office*, um módulo de atualização de *firmware* e uma estação de carregamento de *firmware*. A API verifica diariamente as versões de *firmware* disponibilizadas na Bosch *download store* para cada produto e descarrega as novas versões para o servidor da Bosch Ovar. O serviço de *back-office* gere a solução desenvolvida e permite a consulta dos dados de cada produto e das várias versões de *firmware*. O módulo de atualização de *firmware* comunica com a API e atualiza o *firmware* das câmaras de vigilância, tendo sido integrado nos postos de embalagem e na estação de carregamento de *firmware*. Apenas a API acede à base de dados de produção, sendo os pedidos provenientes *back-office*, estação de carregamento e postos de embalagem processados pela API. Desta forma, melhora-se a segurança e evita-se a sobrecarga no sistema. Por omissão, o sistema é atualizado todos os dias à meia noite. Contudo, é possível forçar a atualização do sistema a qualquer momento através do *back-office*. Com este projeto, a Bosch Security Systems melhora a qualidade dos produtos que envia para o mercado, reduzindo atrasos e erros. O módulo de carregamento de *firmware* nas câmaras de videovigilância e a estação de carregamento de *firmware* foram desenvolvido recorrendo à mesma linguagem de programação do posto de embalagem, o LabVIEW. O restante código foi de-

senvolvido em Python. O sistema está em produção, apresentando fiabilidade e adicionando valor à empresa.

Abstract

Bosch Building Technologies had a problem with the firmware updating of the produced IP video surveillance cameras. On the one hand, the systematic debugging of the code and incorporation of new features makes it necessary to frequently update the firmware, essential to product improvement and market differentiation. On the other hand, firmware updates require the reconfiguration of the camera test software due to incompatibilities between versions. This results in production stops and increased costs. To get around this problem, the company implemented a manual firmware update system carried out at the packaging stations. This process was time consuming and prone to errors given the high diversity of firmware versions and product families.

To solve this problem, a new system was develop within this project encompassing a REST API, a back-office service, a firmware update module and a firmware upload station. The API checks daily the firmware versions available at the Bosch download store for each product and downloads all new versions to the Bosch Ovar server. The back-office manages the service developed and checks the data of each product against the various firmware versions. The firmware update module communicates with the REST API and updates the firmware of the surveillance cameras. This software has been integrated into the packaging and firmware upload stations. To improve security and avoid system overload, only the API has access to the production database, meaning that all requests from the back-office, firmware upload station and packaging stations request are executed by the REST API. While the system updates every day at midnight by default, it is possible to do it at any other time through the back-office. With this project, Bosch Security Systems improves the quality of the products it sends to the market, reducing delays and errors. The firmware upload module and the firmware upload station were developed in the same programming language as the packaging station, LabVIEW. The remaining software was developed in Python. The system is currently in production, proving to be a reliable service and adding value to the company.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Excertos de Código

Lista de Acrónimos

Acrónimo	Descrição	Página
API	Application Programming Interface	2
CSS	Cascading Style Sheets	3
UML	Unified Modeling Language	4
PLC	Programmable logic controller	5
REST	Representional State Transfer	6
RPC	Remote Procedure Call	6
HTTP	Hypertext Transfer Protocol	6
JSON	JavaScript Object Notation	6
XML	Extensible Markup Language	6
WEB	World Wide Web	6
SOAP	Simple Object Access Protocol	7
WSDL	Web Services Description Language	7
URI	Uniform Resource Identifier	8
UDDI	Universal Description, Discovery, and Integration	9
PHP	Hypertext Preprocessor	12
HTML	HyperText Markup Language	12
ORM	Object-relational mapping	17
PCBA	Printed Circuit Board Assembly	21
POE	Power over Ethernet	33
EER	Enhanced Entity-Relationship	41
JWT	JSON Web Tokens	47
ITM	Information and Technology for Manufacture	80
VI	Virtual Instrument	85
IP	Internet Protocol	88

Agradecimentos

A realização desta dissertação contou com o apoio e incentivo de várias pessoas, sem os quais não teria sido realizada.

À Professora Maria Benedita Campos Neves Malheiro, pela orientação, apoio e total disponibilidade durante todo este projeto.

Ao Aníbal Calçada, Group Lider da Engenharia de Testes da Bosch Security Systems, cuja co-orientação foi essencial para o desenvolvimento deste projeto.

A todos os colaboradores da Bosch Security Systems, por todo o contributo dado.

A todos os meus amigos, mas em especial ao Orlando Oliveira, Fernando Gomes, Pedro Silva, Nuno Ferreira e João Barrocas, pelo incentivo e apoio nesta jornada.

E por último, tendo plena consciência que nada disto seria possível de alcançar sem elas, dirijo um agradecimento especial à minha companheira e à minha filha, que foram o meu pilar e a minha força nesta longa caminhada.

Capítulo 1

Introdução

Esta dissertação de Mestrado em Engenharia Eletrotécnica e de Computadores, Área de Especialização em Telecomunicações, do Departamento de Engenharia Eletrotécnica do Instituto Superior de Engenharia do Porto descreve o desenvolvimento de um serviço Web para gestão automática de firmware a colocar em câmaras de videovigilância no final do processo produtivo.

1.1 Problema

A Bosch Security Systems - Sistemas de Segurança S.A. é uma unidade fabril pertencente à divisão Bosch Building Technologies cuja a atividade principal da empresa é produção de produtos de segurança como câmaras de videovigilância, sistemas de comunicações, sistemas de deteção de incêndio e intrusão, produzindo também uma panóplia de produtos pertencentes a outras divisões da Bosch, como a divisão de Termo tecnologia.

A Bosch Security Systems tem um problema com a atualização do *firmware* das câmaras de vigilância IP que produz. Por um lado, a sistemática depuração do código e incorporação de novas funcionalidades obriga à frequente necessidade de atualização de *firmware*, essenciais ao melhoramento e diferenciação dos produtos no mercado. Por outro lado, a atualização de *firmware*, devido a incompatibilidades entre versões, obriga a reconfigurar o *software* de teste, implicando paragens de produção e custos acrescidos. Para contornar este problema, o posto de embalagem passou também a fazer a atualização de *firmware*. Esta alteração ajudou a melhorar o processo produtivo. Assim, os testes funcionais ou finais, que continuam a ser efetuados com a versão de *firmware* validada durante a fase de industrialização, conseguem avaliar o produto sem problemas acrescidos. Estes testes destinam-se a verificar as funcionalidades/caraterísticas e não o *firmware*

do produto, permitindo passar a atualização do *firmware* para o final do processo produtivo sem afetar a qualidade do produto. Desta forma garante-se que o cliente final recebe o produto com a versão mais recente de *firmware* no mercado.

No entanto, o facto da gestão da atualização do *firmware* ser manual e morosa leva a que os postos de embalagem nem sempre possuam a versão mais recente de *firmware*. Quando há novas versões de *firmware* é necessário notificar o responsável dos postos de embalagem acerca dos produtos afetados. Com esta informação, o responsável desloca-se a cada posto de embalagem e introduz manualmente as alterações para cada código afetado. Dada a grande diversidade de câmaras produzidas na unidade fabril de Ovar (mais de 250 códigos diferentes em 12 linhas de produção), é possível antever a fragilidade do processo. É humanamente impossível gerir tantos produtos, com diversas particularidades e exceções, sem cometer erros ou gerar atrasos. Após a atualização do *firmware* nos postos de embalagem, falta notificar o departamento da qualidade para assegurar que, quando for efetuada a inspeção por amostragem a algum produto, a versão de *firmware* instalada seja conhecida.

1.2 Motivação

Este projeto foi aceite por parte do autor, pois para além de aliciante, é um projeto que vai resolver um problema de longa data desta organização e irá proporcionar a aprendizagem de novas linguagens de programação bem como abrir portas para mais projetos de melhoria dentro da organização.

1.3 Objetivos

O objetivo principal é o desenvolvimento de um serviço para automatizar este processo, fazendo com que não haja atrasos na introdução de novas versões de *firmware*, bem como não se introduza a versão errada, ou seja, não se cometam erros, tornado todo este processo mais ágil e cómodo quer para a produção e qualidade quer para a engenharia de testes. Em conjunto com o objetivo principal é também requisito fazer um *back-office* para gestão do serviço, para consulta de toda a informação referente a este tópico e atualizar o posto de embalagem, de forma que este seja compatível com a arquitetura de sistema proposta. Foi também um requisito criar uma estação de carregamento de *firmware* para as linhas cuja cadência de produtos é elevada. Para atingir o objetivo foi necessário dividir estes tópicos que resultam nos seguintes requisitos:

- Desenvolvimento de uma API REST com a finalidade de permitir a automatização de todo o processo, de fazer a verificação e descarregar novas versões de *firmware* existentes no Bosch Download Store (Alemanha);

- Criação de uma interface Web (*back-office*) para gestão do serviço e consulta de informação dos produtos e respetivas versões de *firmware*.
- Desenvolvimento de *software* para carregamento de *Firmware* e comunicação com a API REST, para ser implementado nos Postos de Embalagem e Estações de Carregamento de *Firmware*.
- Desenvolvimento da Estação de Carregamento de *firmware* (*Firmware upload Station*).
- Tecnologias a Utilizar:
  - Python, HTML5, CSS, JavaScript, JSON, Microsoft SQL Server e LabVIEW.

1.4 Planificação

De seguida são apresentadas as tarefas realizadas no âmbito deste projeto.

1. Dissertação - Introdução, estado da arte, descrição do problema e proposta de solução
2. Desenvolvimento API REST
3. Desenvolvimento *Back-office*
4. Criação do Virtual Instrument (VI) para Carregamento de *Firmware*
5. Desenvolvimento da Estação de Carregamento de *Firmware*
6. Dissertação - Desenvolvimento da solução, testes e resultados e conclusão

1.5 Estrutura do Documento

Esta Dissertação é constituída por seis capítulos. No capítulo 1 é apresentada a introdução ao projeto, a contextualização, os objetivos a atingir e os requisitos do mesmo. É também exposta a calendarização.

No Capítulo 2, é apresentado um estudo sobre as tecnologias utilizadas para o desenvolvimento deste projeto, bem como exposto um caso de uso similar.

De seguida, no Capítulo 3, é apresentada a descrição do problema, nomeadamente descreve-se a situação antes do desenvolvimento deste projeto e aborda-se a necessidade da implementação do mesmo. É feita a abordagem à proposta de solução, com a exposição dos vários *software* e serviços que são necessários desenvolver.

No Capítulo 4 é descrita a solução implementada. Ao longo deste capítulo são detalhados os passos do desenvolvimento da API REST, do *back-office*, recorrendo a fluxogramas e a diagramas de sequência UML. É também detalhado o desenvolvimento do VI de Carregamento de *Firmware* nas câmaras de videovigilância e o desenvolvimento da Estação de Carregamento de *Firmware*.

No Capítulo 5 são descritos os testes funcionais e de desempenho, assim como os resultados obtidos.

Por último, no Capítulo 6 são apresentadas as conclusões e expostas possíveis sugestões de melhoria para serem implementadas no projeto desenvolvido.

Capítulo 2

Estado da Arte

Este capítulo contém a descrição das tecnologias e serviços Web mais importantes utilizados no desenvolvimento deste projeto, bem como o caso de estudo de uma empresa que enfrentou um desafio semelhante.

A Bosch decidiu desenvolver um projeto de raiz dada: (*i*) a especificidade do problema, uma vez que não foram encontradas soluções comerciais que satisfizessem todos os requisitos identificados; (*ii*) o custo de subcontratação de serviços a empresas externas; e a (*iii*) a possibilidade de aquisição e retenção de novo conhecimento, permitindo a adição de novos produtos, funcionalidades e a resolução de problemas sem recorrer a terceiros.

Dada a especificidade do problema, a pesquisa de produtos e projetos congéneres realizada encontrou apenas um caso semelhante. Trata-se do sistema desenvolvido pela AUVESY, que implementa a gestão automática e controlo de versões de *firmware* para os *Programmable Logic Controllers* (PLC) utilizados no *shop floor* de uma fábrica alemã. A fábrica efetuava a gestão manual de todos os PLC utilizados na produção, nomeadamente todas as atualizações, versões de hardware e de *firmware* instalados em cada PLC. Este processo manual de atualização era sequencial, levando muito tempo e gerando muitos erros, maioritariamente de origem humana. O *versiondog* do sistema de gestão de dados da AUVESY foi implementado e usado para controlo de versão e *backup* dos PLC utilizados no *shop floor*. O sistema tornou-se uma ferramenta de manutenção indispensável. Dependendo da frequência com que as alterações são feitas, as máquinas de produção podem agendar um *backup* diário ou semanal, sendo comparadas as versões instaladas com as versões armazenadas no servidor central (comparação *online-offline*).

Com este *software* tornou-se fácil verificar as máquinas com necessidade de ser atualizadas [4]. Neste caso, a atualização não é feita em produtos, mas nos PLC

que controlam as máquinas das várias linhas de produção. Segundo a AUVESY, a solução garante a documentação e rastreabilidade de todas as alterações feitas em dispositivos do tipo PLC, *Computer Numeric Control*, *Human-Machine Interface*, robôs e sistemas *Supervisory Control and Data Acquisition*, bem como permite realizar a gestão central de projetos ou programas.

2.1 Serviços Web

Em termos técnicos, a palavra serviço é utilizada quando se faz referência a funções de *software* que executam tarefas tais como, dar acesso a ficheiros/documentos (como por exemplo, texto, imagens, vídeos, ficheiros áudio) ou apenas a execução de funções genéricas como fazer o *login* ou registo. Um serviço *Web* é um conjunto de métodos que podem ser acedidos por *software* que utilizem tecnologias *Web*. É utilizado para transferir dados através de protocolos de comunicação entre diferentes plataformas, ou seja, é independente das linguagens de programação utilizadas pelas plataformas.

Um serviço *Web* disponibiliza um conjunto de funcionalidades para outros sistemas através de uma interface de comunicação bem definida, e deve funcionar em paralelo com os vários serviços que estão a correr, quer do lado do cliente quer do lado do servidor. Os detalhes do serviço são transparentes para o consumidor, ou seja, a comunicação funciona como uma caixa negra, pois os dados são encapsulados e a informação que interessa é acedida através de mensagens, nas quais os contratos definem as operações que os serviços podem oferecer. A implementação de um serviço *Web* compreende um ou mais conjuntos de aplicações que apresentam interoperabilidade, que de uma forma geral funcionam da seguinte forma:

1. O cliente faz um pedido ao serviço;
2. O serviço recebe o pedido, processa a informação e envia a resposta para o cliente;
3. O cliente recebe a resposta e utiliza os dados dessa resposta.

Os serviços *Web* podem adotar um de dois tipos de interface padrão: *Representional State Transfer* (REST) e *Remote Procedure Call* (RPC).

A especificação REST surgiu com o objetivo de simplificar o acesso a serviços *Web*. O REST baseia-se no protocolo *Hypertext Transfer Protocol* (HTTP) e permite utilizar vários formatos de representação de dados, sendo os mais comuns o *JavaScript Object Notation* (JSON) e *Extensible Markup Language* (XML).

A tecnologia RPC baseia-se exclusivamente em XML, quer para a troca de mensagens quer para a descrição dos serviços, servindo-se do protocolo HTTP

para transportar os dados. A informação é trocada em formato XML no corpo das mensagens HTTP, recorrendo ao *Simple Object Access Protocol* (SOAP). Os serviços são auto-descritos em XML, usando a *Web Service Description Language* (WSDL). Assim, cada serviço *Web* RPC é descrito através de um documento WSDL, que descreve a localização do serviço *Web*, as funções/operações disponíveis e a informação necessária para que a comunicação seja possível.

Através deste tipo de mecanismo, é fácil partilhar informação via *Web*, agilizando processos de negócio e permitindo a sua utilização em diferentes arquiteturas [5].

2.1.1 Representional State Transfer

REST especifica um tipo de interface para desenvolvimento de serviços *web* que teve origem na tese de doutoramento de Roy Fielding, um dos autores do HTTP 1.0 e 1.1. Esta arquitetura fornece *standards* para comunicação de dados entre vários tipos de sistemas.

Os serviços *Web* que estão em conformidade com o estilo REST são chamados de serviços da *Web* RESTfull. Os seguintes princípios devem ser seguidos pelos serviços REST:

- **Cliente-Servidor** - A separação das responsabilidades é o princípio por trás do cliente-servidor. Ao separar as preocupações de interface de utilizador (UI) do armazenamento de dados, é possível melhorar a portabilidade através de múltiplas plataformas de UI, simplificar os componentes do servidor, mas principalmente, permitir a evolução de forma independente uma vez que não há dependência entre os lados cliente/servidor.

- **Interface Uniforme** - A caraterística principal que diferencia o estilo REST dos demais é uma interface uniforme entre os componentes cliente e servidor. Como o cliente e servidor partilham esta interface, deve existir um "contrato" bem definido para comunicação entre ambos. Há quatros princípios que devem ser seguidos para obter uma interface uniforme: Identificação dos Recursos, Representação dos recursos, Mensagens auto-descritivas e Hypermedia (HATEOAS).

- **Stateless** - Cada pedido do cliente ao servidor deve conter todas as informações necessárias para este entender o pedido, não sendo da responsabilidade do servidor armazenar qualquer tipo de informação. O estado da sessão é, portanto, mantido inteiramente no cliente. Este princípio acaba por criar um elevado tráfego de dados, que pode ser balanceado pela utilização adequada do *cache*.

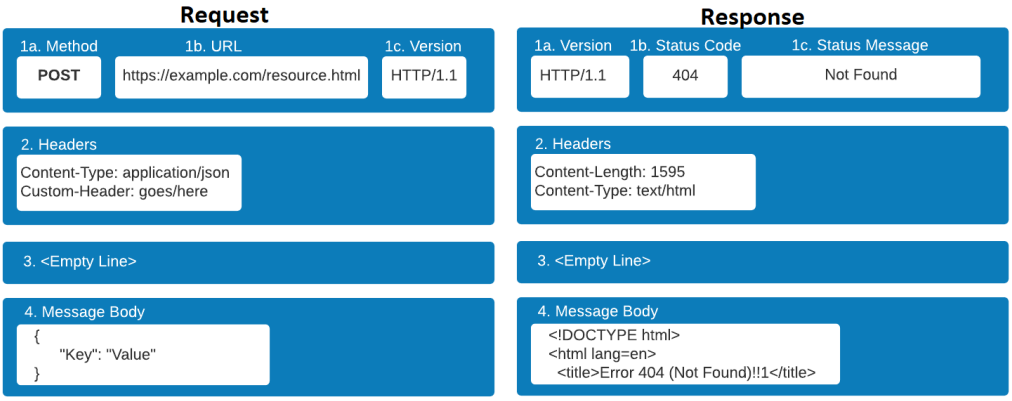


Figura 2.1: Pedido e Resposta - Pedido HTTP [1]

- **Cache** - O *cache* ajuda a melhorar a performance, a escalabilidade e eficiência uma vez que reduz o tempo de resposta médio quando comparado entre uma série de interações cliente-servidor. As diretivas de *cache* são controladas pelo servidor através do cabeçalho HTTP (HTTP Header). O servidor deve informar o cliente se os pedidos podem ser *cached* ou não.

- **Sistema de Camadas** - O estilo de sistema em camadas permite que uma arquitetura seja composta por camadas hierárquicas, restringindo o comportamento do componente de forma que cada componente não possa ”ver”mais do que a camada com que está a interagir.

Os pedidos são centrados em torno de representações de recursos, como um documento ou um URI. Estas representações são normalmente enviadas/recebidas por uma interface padrão, como HTTP. O cliente faz pedidos ao servidor apenas tendo o conhecimento do URI.

Na Figura 2.1, é possível verificar um pedido REST. As aplicações baseadas nesta arquitetura estão em conformidade com os princípios já abordados. O REST utiliza um conjunto standard de métodos como GET, POST, PUT, DELETE e todas as outras capacidades existentes no protocolo HTTP. Resumindo alguns benefícios do REST, como é baseado em operações standard do HTTP, utiliza palavras com um significado específico, como PUT ou DELETE. Desta forma é possível evitar a ambiguidade. Os recursos também são associados aos URI específicos, fazendo com que se tenha mais flexibilidade. Com o REST, a informação gerada e consumida está separada da tecnologia de desenvolvimento das aplicações, o que facilita ambas, quer a informação gerada quer a consumida. Resumidamente, o estilo REST é escalável, simples e fácil de utilizar [6]. Um *RESTful Web Service* pode também ser conhecido por *RESTful API*.

2.1.2 Remote Procedure Call

As interfaces do tipo Remote Procedure Call (RPC) constituem uma interface padrão para serviços *Web*.

Os serviços *Web* RPC têm as seguintes caraterísticas:

- Têm interfaces que se autodescrevem em documentos XML independentes de plataformas. WSDL é o padrão utilizado para descrever os serviços.
- Comunicam através de mensagens definidas através de esquemas XML (também chamado por XSD). A comunicação entre fornecedores de serviços e os consumidores de serviços acontece em ambiente heterogéneo.
- São mantidos através de um registo que funciona como uma listagem de diretórios. As aplicações podem procurar serviços no registo e chamar esses serviços. O Universal Description, Definition and Integration (UDDI) é uma especificação do serviço de registo de serviços *Web*.

2.1.2.1 SOAP - Simple Object Access Protocol

O protocolo SOAP é baseado em XML. As mensagens do SOAP são encapsuladas no corpo das mensagens HTTP. O SOAP é:

1. Extensível, pois suporta objetos desde os mais básicos (int, string, etc.) até aos mais complexos (arrays, clusters, etc.);
2. Independente da plataforma, modelo de programação e linguagem de programação;
3. Contém mecanismos de deteção de erros.

A estrutura de uma mensagem SOAP consiste num envelope que contém cabeçalho e corpo. A identificação destes componentes é feita através dos elementos XML `envelope`, `header` e `body`.

A estrutura de uma mensagem SOAP pode ser visualizada na Figura 2.2 O elemento *envelope* é a raiz de uma mensagem SOAP, correspondendo à descrição da mensagem. Pode conter também declarações de espaços, de nomes e vários atributos relativos à forma como os dados XML são representados, recorrendo ao `namespace encondig style`. O cabeçalho é opcional e tem como função estender as funcionalidades das mensagens SOAP. Desta forma é possível adicionar novas funcionalidades à mensagem, tais como autenticação, transações e encriptação, sem ser necessário alterar as especificações do SOAP, permitindo deste modo manter a interoperabilidade de sistemas. Caso exista cabeçalho na mensagem, este deve ser o primeiro elemento do envelope. O elemento `body` é um

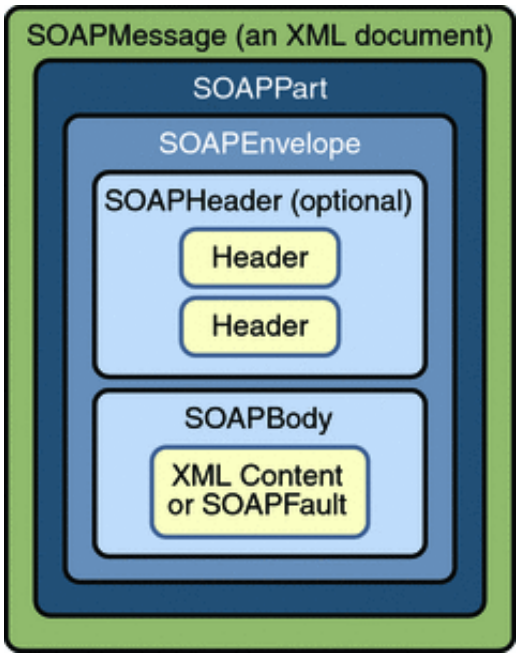


Figura 2.2: Estrutura da Mensagem SOAP [2]

elemento obrigatório de qualquer mensagem SOAP e nele encontramos a informação propriamente dita. Dentro do corpo de uma mensagem pode-se encontrar desde simples chamadas a métodos, até folhas de estilos, ou qualquer outro tipo de informação, em formato XML, que se pretenda transferir através de uma mensagem. O conteúdo da mensagem é representado através de elementos do `body` de acordo com o `encodingStyle` escolhido. Como opcional do `body`, pode utilizar-se o elemento `fault`, que serve para transportar mensagens de erros que se vão acumulando pelos vários nós que processam a mensagem até ao destino [5].

2.1.2.2 Web Service Description Language

A especificação Web WSDL é uma linguagem baseada em XML que descreve de forma padronizada e independente de plataforma como e onde os serviços podem ser conectados e utilizados através da rede [7]. Um documento WSDL é composto por três elementos XML:

1. `types` - descreve os tipos de dados suportados pelo serviço em questão;
2. `message` - especifica os padrões de entrada e saída de dados dos serviços *Web*;
3. `operation` - é o campo que permite a especificação das assinaturas dos métodos disponibilizados;

4. `port type` - um conjunto abstrato de operações suportadas por um ou mais *endpoints*;

5. `binding` - define o protocolo e formato de dados para as mensagens definidas num determinado tipo de porto;

6. `port` - é um *end point*, *i.e.* representa a combinação de um `binding` e um endereço de rede;

7. `service` - é um conjunto de portas, onde cada elemento `port` se relaciona com um elemento `binding`, indicando qual a interface e protocolo de comunicação que estão a ser utilizados.

2.1.2.3 Universal Description, Definition and Integration

O protocolo UDDI é um *standard* aprovado pela OASIS e um membro-chave da camada de serviços *Web*. Este define um método padrão para publicar e descobrir os componentes de *software* de rede baseados numa arquitetura orientada a serviços (SOA).

O UDDI especifica mecanismos para a publicação, descoberta e integração de serviços. Tais mecanismos definem as estruturas de dados necessárias para a sua descrição e classificação, bem como uma interface baseada no protocolo SOAP que permite o acesso às informações. O serviço UDDI é muito idêntico à utilização de uma lista telefónica para encontrar um número. O protocolo UDDI apresenta três diretórios[8], representados sob a forma de esquemas XML:

1. Páginas Brancas - contêm identificadores sobre o contacto técnico do serviço oferecido;

2. Páginas Amarelas - contêm informações genéricas sobre os tipos e localização dos serviços disponíveis;

3. Páginas Verdes - contêm informações técnicas sobre um determinado serviço *Web*.

A finalidade funcional de um registo UDDI é a representação de dados e metadados sobre serviços *Web*. Um registo, para uso na rede pública ou na rede interna de uma organização, oferece um mecanismo baseado em padrões para classificar, catalogar e gerir serviços *Web*, de modo a que possam ser descobertos e consumidos por outras aplicações ou serviços. Consequentemente, o *standard* do UDDI, especifica protocolos para aceder a um registo de serviços *Web*, métodos para controlar o acesso ao registo e um mecanismo para distribuir ou delegar registos a outros registos. Por outras palavras, fornece um meio para encontrar, chamar e gerir os metadados do serviço.

1. `modelo de dados UDDI` - A especificação UDDI define os tipos de dados principais que incluem uma descrição da função de negócios do serviço, informações sobre o editor do serviço, os detalhes técnicos e API do serviço. Estes dados são definidos em vários esquemas XML, que juntos formam um modelo de informações de base e estrutura de interação de registos UDDI.

2. os fornecedores dos serviços inscrevem-se e registam os seus serviços *Web* no serviço de registos UDDI, classificando-os de acordo com categorias pré-definidas e carregando os correspondentes ficheiros WSDL. Os eventuais clientes interrogam o serviço de registos UDDI, usando as categorias pré-definidas, escolhem o serviço que pretendem consumir, descarregam o respetivo WSDL, e, de forma quase automática e com base no ficheiro WSDL, conseguem interagir diretamente com o serviço.

2.2 Tecnologias do Lado do Cliente

Os padrões *Web*, segundo a definição W3C, são um conjunto de diretrizes, recomendações ou notas, de carácter técnico criados pelo World Wide Web Consortium (W3C) e destinados a orientar os programadores para o uso de boas práticas na criação de páginas *Web*, acedidas através de qualquer tipo de dispositivo. Quando se fala em normas, na prática tratam-se de três componentes independentes: linguagens de anotação *Hypertext Transfer Protocol* (HTML) e XML), linguagens de apresentação (CSS) e linguagens de *scripting* (Javascript, PHP).

2.2.1 HTML5

O HTML5 é a última versão do HTML e um dos seus principais objetivos é facilitar a manipulação dos elementos, possibilitando alterar as caraterísticas dos objetos de forma não invasiva, fazendo com que fique transparente para o utilizador final. Diferente das versões anteriores, o HTML5 fornece ferramentas para o CSS e o Javascript fazerem o seu trabalho da melhor forma possível, ou seja, sem que estes causem mau funcionamento à página *Web*. Algumas *tags* foram modificadas, outras criadas e algumas descontinuadas. As versões anteriores do HTML não eram padronizadas para criação de secções comuns e específicas como rodapé, cabeçalho, barra lateral, menus, etc. Nesta versão não é necessário escrever muito código para ter todas as funcionalidades das versões anteriores, aumentando a interatividade sem a necessidade de instalação de *plug-ins*, que em alguns casos, causa perda de performance.

2.2.2 JSON

JavaScript Object Notation é uma formatação leve para troca de dados. O JSON para além de ser um formato muito leve é também muito simples de interpretar e gerar, quer por seres humanos quer por máquinas. É em formato texto e completamente independente das linguagens de programação, uma vez que usa convenções que são familiares às várias, como o C, C++, Java, JavaScript, Perl, Python, etc. A ideia utilizada pelo JSON para representar informações é tremendamente simples: para cada valor representado, atribui-se um nome (ou rótulo) que descreve o seu significado. Esta sintaxe é derivada da forma utilizada pelo JavaScript para representar informações.

2.2.3 CSS

O *Cascading Style Sheets* (CSS) descreve como os elementos HTML são exibidos numa página *Web*. Com CSS o trabalho de criar páginas agradáveis é mais reduzido pois este permite controlar o *layout* de várias páginas *Web* de uma só vez. Todos os estilos são armazenados num ficheiro CSS, que pode ser utilizado pelas várias páginas de um *front-end*.

2.2.4 JavaScript

O JavaScript é uma linguagem de programação que é executada do lado cliente, ou seja, é processada pelo próprio *Web Browser*. Com o JavaScript é possível criar efeitos especiais para incorporar nas páginas HTML, para além de proporcionar uma maior interatividade com utilizadores. O JavaScript é uma linguagem orientada a objetos tratando todos os elementos da página como objetos distintos, facilitando a tarefa da programação. É uma poderosa linguagem que deve ser dominada por quem deseja criar páginas *Web* dinâmicas e interativas. Com a evolução desta linguagem surgiram algumas bibliotecas, como por exemplo, o *JQuery*.

O *JQuery* é a biblioteca mais conhecida de JavaScript e fornece uma variação de JavaScript com uma sintaxe mais amigável, facilitando e simplificando a criação de aplicações. Esta biblioteca é tão popular que em muitas situações já se substitui o JavaScript nativo por *JQuery*, como por exemplo, o *framework* Bootstrap.

2.3 Linguagens, Bibliotecas e Ferramentas

2.3.1 LabVIEW

O Laboratory Virtual Instrument Engineering Workbench (LabVIEW) é uma plataforma e ambiente de desenvolvimento da National Instruments para desenvolver

aplicações de aquisição de dados, testes, instrumentação e controlo. Apresenta uma linguagem gráfica de programação baseada em instrumentos virtuais, que a torna muito atrativa.

Um Virtual Instrument (VI) é um elemento de programação do LabVIEW. Um VI consiste num painel frontal, num diagrama de blocos e num ícone que representa o programa. O painel frontal é utilizado para mostrar os controlos e indicadores utilizados no programa, e o diagrama de blocos contém o código desenvolvido do VI. O ícone, que é a representação visual do VI, tem conectores para entradas e saídas do programa. O painel frontal é responsável pela funções de entrada e saída e o diagrama de blocos executa o código desenvolvido do VI. Múltiplos VI podem ser utilizados para criar uma aplicação de larga escala, pois normalmente este tipo de aplicações têm centenas de VI a correr. Um VI pode ser utilizado como interface ou como uma sub-rotina/função numa aplicação [9].

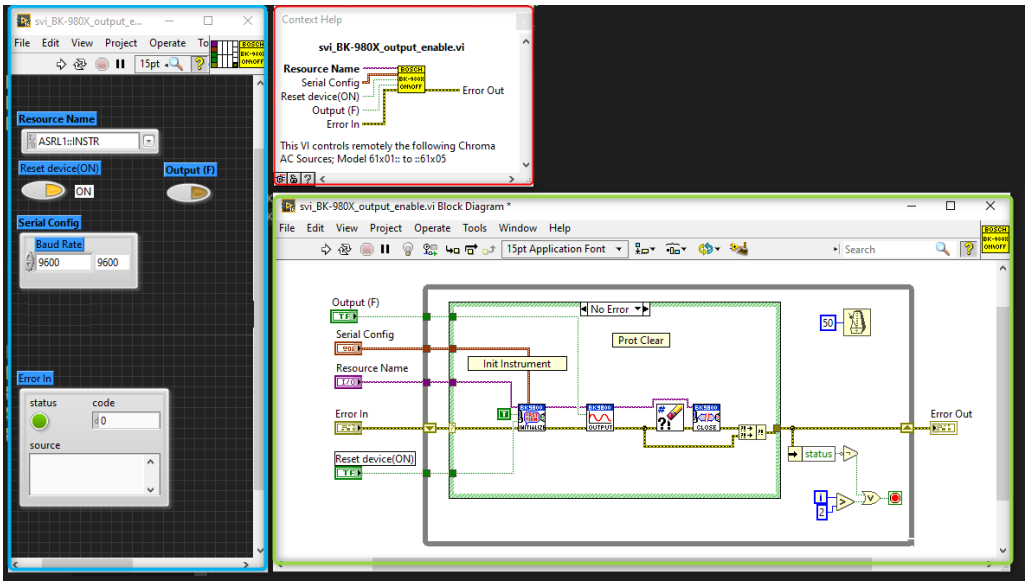


Figura 2.3: Exemplo de um VI - Labview

Na zona azul da Figura 2.3 é possível visualizar o painel frontal do VI com os vários controlos. Na zona a verde, pode consultar-se o diagrama de blocos. Neste diagrama é possível verificar que o VI principal está a executar alguns VI. Estes VI são as funções que são necessárias chamar para atingir um objetivo. Pode também consultar-se o que para além dos VI, também é desenvolvido código como comparações lógicas, contadores, etc. Na zona delimitada a vermelho, estão as entradas e saídas do VI e o respetivo ícone. Os ícones a negrito, são os que são obrigatórios ligar quando este VI é executado dentro de outro VI. Todos os controlos e indicadores colocados no painel frontal ficam disponíveis no diagrama de blocos de forma imediata, para se poderem utilizar.

2.3.2 Python

Python é uma linguagem de programação *open source* criada por Guido Van Rossum em 1991. Os objetivos base desta linguagem foram a produtividade e legibilidade, por outras palavras, esta linguagem foi criada para permitir a criação de código que fosse fácil de manter ao longo da vida das aplicações. Isto é possível devido à sua sintaxe. Ao contrário de outras linguagens, o Python já tem incluído de raiz uma grande quantidade de extensões, permitindo assim o desenvolvimento de aplicações de uma forma mais rápida. A versatilidade do Python pode ser utilizada para operações simples, como a criação de *scripts* para leitura de ficheiros, bem como desenvolver sites de organizações com uma elevada transferência e gestão de dados. As principais caraterísticas desta linguagem são:

- Produtividade;
- Portabilidade;
- Multi-plataforma;
- Manutenção;
- Qualidade do *software*;
- Ambientes virtuais.

Todas estas caraterísticas fazem do Python uma linguagem versátil e poderosa, com cada vez mais utilizadores a preferirem esta linguagem a outras.[10] Uma vez que é uma das linguagens mais utilizadas no mundo, existe um suporte elevado em torno desta linguagem, o que faz com que existam cada vez mais bibliotecas que facilitam o desenvolvimento de aplicações, pois as bibliotecas já fornecem classes que podem executar as mais variadas tarefas, como o OpenCV, que já tem classes que se podem utilizar rapidamente e de forma ”intuitiva”para análise de imagem. Neste caso, os serviços vão ser desenvolvidos com recurso a bibliotecas como o Flask e o SqlAlchemy.

2.3.3 Flask

O Flask, para a maioria dos *standards* é um pequeno *framework*, sendo designado *micro-framework*. Este facto não quer dizer que seja menos capaz do que os *frameworks* como o Django. Pelo contrário, é um *micro-framework* desenhado para ser extensível, isto é, fornece aos utilizadores um *core* muito sólido, com todos os serviços básicos e inúmeras extensões. Uma vez que se podem escolher todos os pacotes de extensões, no final de contas, acaba-se por ter um sistema específico, apenas com o que queremos, não utilizando extensões ou recursos desnecessários. O Flask tem duas grandes dependências, a depuração e roteamento e o WSGI,

enquanto o suporte para os *templates* é fornecido pelo Jinja 2 [11]. Ambos são da autoria dos *core developers* do Flask. [12] Este micro-*framework* tem como principais caraterísticas:

- Servidor de desenvolvimento e depuração;
- Suporte para testes unitários;
- Tratamento de pedidos RESTful
- Jinja 2 para criar *templates*. É um sistema de *templates* moderno e de fácil compreensão, quer para programadores como para designers;
- Suporte para *secure cookies* (sessões do lado do cliente);
- 100 % compatível com WSGI 1.0.1. O WerkZeug é uma biblioteca para desenvolvimento de aplicações WSGI. O *Web Server Gateway Interface* é a especificação universal de como se deve criar uma interface entre uma aplicação ou *framework* em Python e um *Web Server*. Possui a implementação básica deste padrão para lidar com pedidos e respostas, tem controlo de *cache*, *cookies*, status HTTP, roteamento/*routing* de URL e também tem uma poderosa ferramenta de depuração. A última versão do WSGI também é conhecido por PEP3333 [13];
- Baseado em unicode;
- Amplamente documentado; [14];
- Good Intentions: além do código ter alta qualidade a nível de requisitos de legibilidade, este segue as premissas do Zen do Python e dentro dessas boas intenções o facto de ser um micro-*framework* permite ter a liberdade na forma de estruturar as aplicações. Com recurso aos Blueprints, é possível reaproveitar código já desenvolvido muito facilmente, permitindo desta forma que as aplicações evoluam sem problemas.

Como o FLASK é um micro-framework, não tem suporte nativo para aceder a bases de dados, validação de formulários, autenticação de utilizadores ou qualquer outro tipo de tarefa de nível superior.[15]. Os serviços referidos e muitos mais (serviços importantes) necessários, estão disponíveis e precisam de estar ligados a outros através de várias extensões do Python. No caso do Framework Django, já não acontece assim, uma vez que tem muitas mais opções a este nível do que tem disponível para desenvolvimento *Web*, uma vez que é específico para este tipo de finalidade.

Os prós do Flask:

- Desempenho;
- Flexibilidade;
- Portabilidade;
- Simplicidade;
- Robustez;
- Liberdade.

Os contras do Flask:

- Não é compatível com aplicações assíncronas;
- Suporte e documentação limitada;
- Falta de base de dados / ORM / forms;
- Limitado a nível de recursos.

2.3.4 SQLAlchemy ORM

Com recurso ao SQLAlchemy, quando se desenvolve aplicações, esta biblioteca permite ter acesso direto às tabelas da base de dados. Para isso basta combinar Python com as *queries* em SQL. Este tipo de solução funciona, mas torna-se difícil de manter quando a dimensão da aplicação aumenta, com a introdução de *queries* mais complexas, ou quando é necessário migrar o motor de base de dados por várias razões. Outra dificuldade encontrada prende-se com o facto de que em Python os dados são colocados dentro de objetos e nas bases de dados estão guardados em tabelas, logo o programador necessita de fazer conversões entre os dois formatos de dados. O que também pode ser complexo. Para resolver estes problemas foi criado O SQLAlchemy ORM, que é semelhante a muitos outros *object-relational mapping* (ORM) que se podem encontrar noutras linguagens de programação. O SQLAlchemy ORM mapeia as várias tabelas da base de dados em classes, as linhas da base de dados em instâncias de objetos e as colunas em instâncias de atributos.[16]

Na Figura 2.4 é possível visualizar a tabela de base de dados.

Na Figura 2.5 pode visualizar-se a classe criada em Python com recurso ao SqlAlchemy para mapear a tabela da Figura 2.4. Cada coluna da base de dados corresponde a um atributo da classe `DB_BCS_TransferStatus`. Quando é executado um *query* para obter dados das várias entradas da tabela, em que cada entrada corresponde a uma linha, para cada atributo os dados serão retornados como objetos desse atributo.

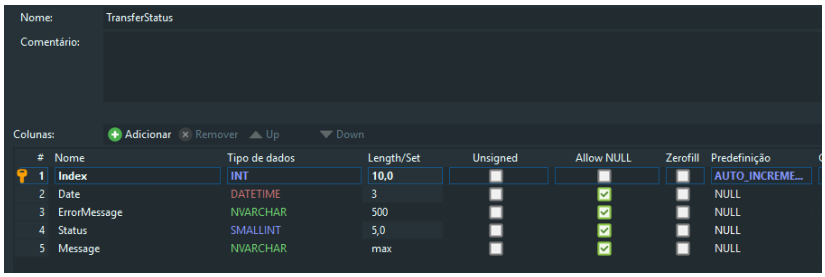


Figura 2.4: Exemplo - Tabela base de Dados

```
class DB_BCS_TransferStatus(Base):
    __tablename__ = "TransferStatus"
    index = Column(Integer, primary_key=True)
    Date = Column(DATETIME(255))
    ErrorMessage = Column(String(500))
    Status = Column(SMALLINT())
    Message = Column(String(max))
    __table_args__ = {"schema": "OvrP_CS"}
```

Figura 2.5: Exemplo - Classe que mapeia a tabela da base de Dados

2.4 Servidor de Bases de Dados Relacionais

Uma vez que a base de dados existente na Bosch Security Systems é a Microsoft SQL Server e não é possível criar e fazer alterações a tabelas, devido a razões de segurança, antes de criar a estrutura neste motor de base de dados, foi decidido fazer o desenvolvimento de todo o sistema com recurso ao MySql e posteriormente migrar a base de dados para a base de dados de Produção da Bosch Security Systems.

2.4.1 Microsoft SQL Server 2016

O SQL Server também conhecido com MSSQL quer dizer Microsoft SQL Server. Foi desenvolvido pela Microsoft e lançado para o mercado no ano de 1989. O código fonte deste motor de base de dados é em C e C++. É um motor de base de dados com várias versões, como a *enterprise* e a *basic*. As tabelas desta base de dados são utilizadas para guardar registos ou informação e a sintaxe das *queries* mais simples, como as *queries* das operações *create*, *read*, *update* e *delete* (CRUD), é básica. Uma vez que é uma base de dados escalável pode ser utilizada para projetos pequenos ou de grande dimensão.

2.4.2 MySQL

O MySQL é um sistema de gestão de base de dados *open source* relacional usado na maioria das aplicações gratuitas para gerir bases de dados. O serviço utiliza a linguagem SQL (Structure Query Language), que é a linguagem mais popular para inserir, aceder e fazer a gestão do conteúdo armazenado numa base de dados. A grande vantagem desta linguagem, para além de ser *open source* prende-se com o facto de ser suportada por todos os sistemas operativos e por uma panóplia enorme de linguagens de programação cuja comunicação é efetuada através dos adaptadores ODBC, JDBC e NET mediante a linguagem de programação.

2.5 Servidor de Aplicações Waitress

O *Waitress* é um servidor de produção WSGI (*Web Server Gateway Interface*), desenvolvido puramente em Python e com um bom desempenho para serviços *Web*. Este servidor não tem dependências exceto as que residem na biblioteca padrão do Python. Para os sistemas operativos UNIXm este servidor corre em CPython e para sistemas operativos Windows corre em Python 3.5+. Este servidor suporta HTTP 1.0 e HTTP 1.1, tendo como grande limitação o facto de não suportar o protocolo HTTPS. É ideal para aplicações de pequena e média dimensão e que não estejam expostas na *Web*.

2.6 Ambiente de Desenvolvimento

2.6.1 Pycharm

O Pycharm é um IDE utilizado para o desenvolvimento de aplicações em linguagem Python. Com recurso a este IDE, a produtividade no desenvolvimento aumenta, pois tem ferramentas que ajudam a fazer *debug*, identificar erros e auxilia na construção do código através da utilização de palavras chave.

2.6.2 HeidiSQL

O HeidiSQL [17] é um *software* livre e tem como objetivo ser fácil de aprender. Este IDE permite ver e editar dados e estruturas de bases de dados de forma fácil e intuitiva. Suporta motores de bases de dados como a MariaDB, MySQL, Microsoft SQL, PostgreSQL e SQLite. Desenvolvido em 2002 por Ansgar, HeidiSQL pertence às ferramentas mais populares para fazer a gestão de base de dados MariaDB e MySQL.

2.6.3 Postman

Para fazer pedidos HTTP, como um GET ou POST, este IDE é ideal para este fim. Permite validar e testar as API desenvolvidas de forma rápida, pois fornece uma quantidade grande de ferramentas e de opções para fazer os pedidos HTTP. Facilita na criação do *header* e do *body*, pois permite mostrar os dados no formato que pretendemos, como JSON, XML e HTML. Permite também executar testes de performance, com vários exemplos já disponíveis para quem desenvolve utilizar.

2.7 Sumário

Concluindo, o estilo REST é mais flexível do que o RPC, sendo que nos últimos anos o RPC está a perder a sua posição no mercado. Uma vez que o REST é mais leve, a sua utilização é mais intuitiva, fica do lado de quem desenvolve criar a estrutura dos pedidos HTTP e respetivas respostas, limitando assim a quantidade de dados que não são úteis para certas mensagens que o RPC obriga, transmitindo apenas o que se pretende geralmente com a utilização de JSON. Desta forma optou-se por desenvolver todo o serviço com recurso à arquitetura REST.

Na Figura 2.6 é possível visualizar a diferença entre o RPC e o REST em termos de implementação do protocolo de comunicação.

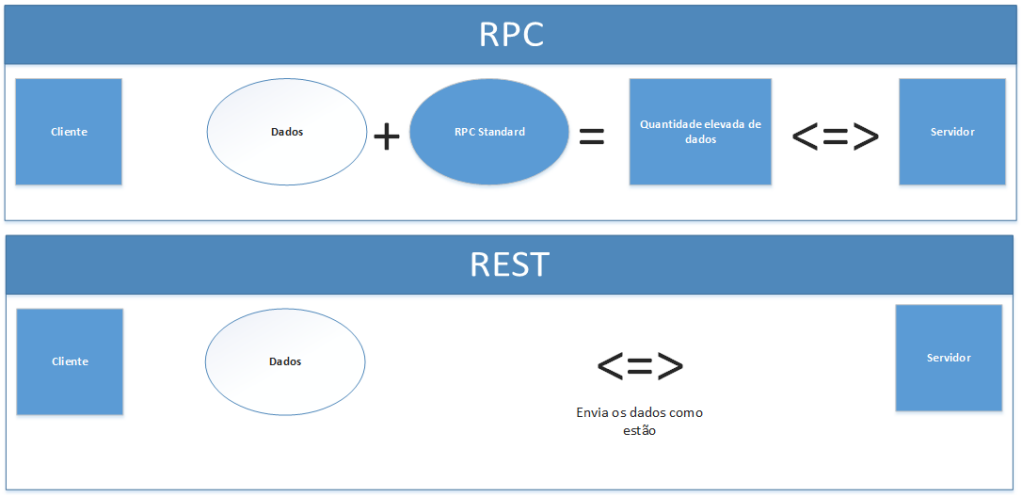


Figura 2.6: RPC XML vs REST [3]

As tecnologias utilizadas foram sugeridas/aconselhadas para ir de encontro à infraestrutura existente na organização.

Capítulo 3

Problema e Proposta de Solução

Este capítulo apresenta a descrição do problema, os casos de uso, requisitos da solução e a arquitetura.

3.1 Problema

A Bosch Security Systems é uma fábrica pertencente à divisão da Bosch Bulding Techonologies que durante os últimos anos tem-se deparado com um problema nas linhas de produção de câmaras de videovigilância, associado ao carregamento tardio da versão de *firmware* sempre que uma atualização é lançada para o mercado. Isto quer dizer que no passado, sempre que era necessário fazer a atualização do *firmware* de uma dada família de câmaras, era necessário introduzir a alteração durante o processo produtivo. Na fábrica de Ovar, existem testes funcionais e finais, sendo que antes da montagem de um produto, todos os PACBA são testados e se necessário programados, por forma a tentar garantir o bom funcionamento de todos os PCBA antes da montagem do produto. Isto é validado pelos testes funcionais, que como referido para além de testar funcionalmente o PCBA, nos que necessitam também tem incorporada a programação de *firmware* do produto. Desta forma, quando era necessário atualizar um produto, era criado um pedido de alteração ao mesmo e o *firmware* era colocado no teste funcional. Isto trazia vários problemas para a linha de produção, pois a compatibilidade entre versão não é garantida, uma vez que o desenvolvimento vai implementando e retirando certas funções do *firmware*, o que levava a que alguns comandos *Rate Control Protocol* (RCP) do protocolo de comunicação do *firmware* deixassem de funcionar e algumas funções alteravam o seu comportamento. Estes problemas levavam a paragens de linha causadas pela depuração do problema por parte da engenharia, problemas estes que podiam levar algumas horas ou até mesmo dias para se

identificar e tentar solucionar, causando perdas avultadas para a organização. Foi então decidido que para ultrapassar este problema a atualização da versão mais recente de *firmware* do produto passaria para o Posto de Embalagem, que é o último posto antes de embalar o produto. Isto não cria problemas de qualidade, pois a função dos testes não é validar o *firmware* mais sim validar as caraterísticas críticas do produto. Desta forma definiu-se que a versão de *firmware* utilizada em produção seria a última utilizada durante a industrialização dos novos produtos antes de entrarem em produção em massa. Em relação aos produtos já em produção em massa, a opção foi manter a versão que estavam a utilizar. Desta forma, passaram a ser carregadas nas câmaras as versões de *firmware* no Posto de Embalagem. Estas versões são descarregadas manualmente do portal Bosch Download Store, que é o portal onde as novas versões são disponibilizadas para os clientes. Este portal mantém as versões de *firmware* lançadas para cada família de produtos.

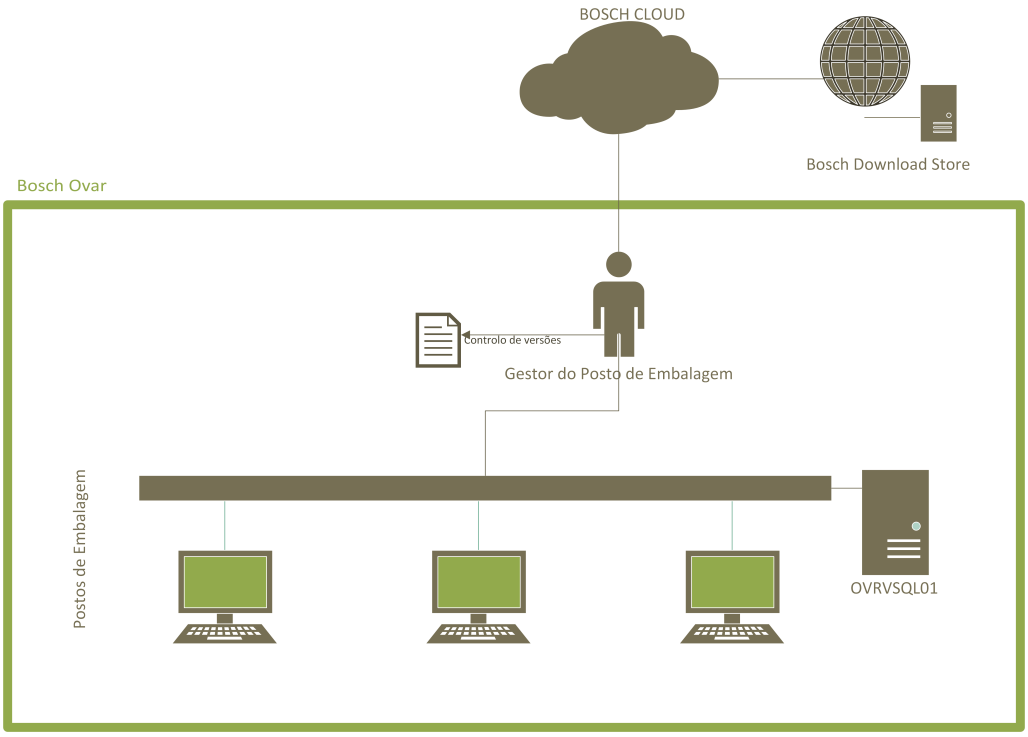


Figura 3.1: Arquitetura Atual

Na Figura 3.1, está representada a arquitetura de gestão manual do sistema. Como se verifica, toda a gestão é manual, sendo que o gestor deste sistema tem que ser notificado por alguém do departamento de desenvolvimento ou ir consultando o portal para verificar se existem novas versões. Sempre que existem, este tem que manualmente verificar quais os produtos afetados, que para isso, acedia ao

documento `version_table.txt`, onde consultava qual versão que cada produto deveria ter instalado. Para gerir, passava toda a informação para uma folha de cálculo e fazia a gestão a partir da mesma. Como se pode imaginar, não é fácil manter um documento onde se faz a gestão de uma grande diversidade de produtos, neste caso corresponde a cerca de 250 códigos SAP. Esta grande diversidade de produtos levava a erros de mercado, uma vez que existem algumas famílias de câmaras que foram desenvolvidas especialmente para certos clientes e cuja a versão de *firmware* só pode ser atualizada a pedido dos mesmos. Estas câmaras são iguais às de produção normal, ou seja, com o mesmo `ID`, apenas com a alteração do código SAP da câmara, logo com tanta diversidade e algumas excepções é expectável que ocorram vários erros ao longo do processo produtivo, que levaram a fazer *recalls* de produtos. Apesar de todo o ganho visível na linha de produção existe a necessidade de automatizar todo o processo, de forma a que não seja necessário qualquer tipo de interação/gestão por parte do gestor do sistema, bem como eliminar erros de produção que levam a custos acrescidos causados por *recalls* a produtos no mercado e também à introdução tardia das novas versões de *firmware* nos postos de embalagem.

Como nem sempre o ficheiro `version_table.txt` estava devidamente atualizado e as exceções dos vários clientes não constavam no mesmo, foi acordado que seria o desenvolvimento e os gestores dos projetos a manter este ficheiro atualizado, com as excepções dos vários clientes incluídas. Assim, a fábrica apenas tem que consultar os dados nele presentes sem ter que se preocupar se estão todos corretos e atualizados. Com este passo garantido, foi possível criar um novo conceito completamente automático para executar todas estas operações. Todas as versões de *firmware* serão disponibilizadas no servidor `LocalDLs`, onde também se consulta o ficheiro `version_table.txt`. O servidor `LocalDLs` não é nada mais do que uma réplica do servidor Bosch Download Store. Foi decidido fazer uma réplica do servidor, pois o `LocadlDLs` não está acessível através da Internet, apenas dispositivos ligados à Bosch Cloud e utilizadores com privilégios podem aceder a este servidor.

3.2 Requisitos

Após a identificação do problema e para implementar o serviço de forma a colmatar as várias falhas apresentadas anteriormente, nomeadamente a introdução tardia de novas versões de *firmware*, a introdução de versões de *firmware* erradas em alguns produtos e a descarga automática das novas versões de *firmware*, foram enumerados vários requisitos que o serviço tem que incluir e fornecer de forma a corrigir as falhas mencionadas. O projeto terá uma API REST e também a criação de um *back-office*, dividido em *back-end* e *front-end*. Este *back-office* será utilizado para fazer a gestão e consulta de toda a informação, para que a Bosch

Ovar possa resolver os seus problemas neste campo. De seguida, estão definidos os requisitos deste serviço REST:

Back-office

- Lista de produtos e respetivas versões de *firmware*;
- Possibilidade de adicionar produtos manualmente para testes quando necessários e *sample runs* durante algumas etapas da industrialização dos produtos;
- Lista de *emails* para notificar sempre que existirem atualizações;
- Gestão de utilizadores divididos em duas categorias, administrador e não administrador;
- Gestão de Computadores do posto de embalagem e estação de carregamento de *firmware* com acesso à API REST;
- Possibilidade de verificar quando foi a última atualização e o estado da mesma;
- Histórico de atualizações do sistema e por produto;
- Enviar *email* aos administradores em caso de erro durante algum processo e armazenar o erro na base de dados;
- Botão para forçar uma atualização/verificação de novos *firmware* caso necessário (administrador);
- Adicionar utilizadores e permitir o registo de novos utilizadores;
- Todas as operações devem ser executadas através de pedidos HTTP à API REST.

API REST

- Verificar todos os dias se existem atualizações;
- *Download* do ficheiro `version_table.txt` com a lista produto e *firmware*, conversão do ficheiro para o formato JSON;
- Comparar versões do JSON com as versões disponíveis na base de dados e caso existam diferenças ou novas entradas, atualizar a base de dados;
- Notificar via *email* sempre que for executada uma atualização ou introdução de produtos novos;
- Descarregar automaticamente as novas versões de *firmware*;

- Fazer toda comunicação com a base de dados;
- Implementar mecanismo de segurança com recurso aos JSON Web Tokens;
- Responder a todos os pedidos do posto de embalagem, da estação de carregamento, de *firmware* e do *back-office*;
- Criar recursos para que todos os pedidos sejam processados pela API por forma a garantir que apenas esta se liga à base de dados, garantindo menos *overflow* e melhorando o desempenho da base de dados, por exemplo, `GET ovrprodsrv01:6001/token`;
- Responde a pedidos GET, POST, PUT e DELETE;

Carregamento de *Firmware*

- Criar novo *software* em LabVIEW para fazer o *firmware upload* de forma a eliminar a dependência do *software* fornecido pela Bosch Nuremberga para o efeito;
- Fazer com que o *software* suporte a nova plataforma de câmaras da Bosch baseadas em Android. O *software* de Nuremberga não suporta esta nova plataforma;
- Este *software* deve fazer todos os pedidos à API REST;
- Eliminar a dependência de ficheiros de configuração;
- Desenvolver a compatibilidade deste *software* com o posto de embalagem atual, para isso apenas são precisos o URI da API e identificador SAP do produto.

VI de Carregamento de *Firmware*

- Fornecer o *software* em LabVIEW para ser integrado no código fonte do posto de embalagem.

Posto de Carregamento de *Firmware*

- Criar novo *software* para fazer o *firmware* upload de forma a eliminar a dependência do *software* fornecido pela Bosch Nuremberga para o efeito;
- Fazer com que o *software* suporte a nova plataforma de câmaras da Bosch baseadas em Android. O *software* de Nuremberga não suporta esta nova plataforma.

- Este *software* deve fazer todos os pedidos à API;
- Eliminar a dependência de ficheiros de configuração;
- Permitir o carregamento de *firmware* em paralelo, por forma a aumentar o capacidade das linhas de produção.

3.3 Casos de Uso

Grande parte dos erros encontrados em sistemas orientados a objetos tem origem no modelo definido no início do projeto. Para evitar erros e sistemas mal definidos, com recurso ao UML e aos casos de uso é possível descrever todos os aspetos, funcionalidades a capacidade de execução do sistema, bem como o que cada ator pode fazer. O ator pode ser um utilizador ou uma entidade ou serviço. Irão ser descritas as principais funcionalidades do sistema e a interação com os atores. Nesta fase do projeto, a ideia não é aprofundar os detalhes mais técnicos, mas sim representar cada tarefa.

De seguida, estão representados os atores de cada caso de uso:

- *Back-office*
  - Administrador
  - Utilizador
  - Utilizador não registado
- Posto de embalagem e Estação de Carregamento de *Firmware*
  - Posto de Embalagem
  - Estação de Carregamento de *Firmware*

3.3.1 Back-office

Os casos de uso de cada ator do *back-office* estão representados na Tabela 3.1.

Tabela 3.1: Casos de Uso *Back-office*

Ator	**Casos de Uso**	
Administrador	Consultar lista de produtos	Transferir produtos para a lista de excepções
	Consultar lista de excepções de produtos	Gerir dados dos produtos
	Consultar lista de Máquinas	Gerir Máquinas
	Consultar lista de utilizadores	Gerir utilizadores
	Consultar lista de Firmware	Gerir *Firmware*
	Consultar lista de transações do sistema	
	Registar novo utilizador	
	Forçar Atualização/Verificação de *Firmware*	
Utilizador	Consultar Lista de produtos	
	Consultar estado da ultima atualização do sistema	
Utilizador não Registado	Fazer registo de novo utilizador	

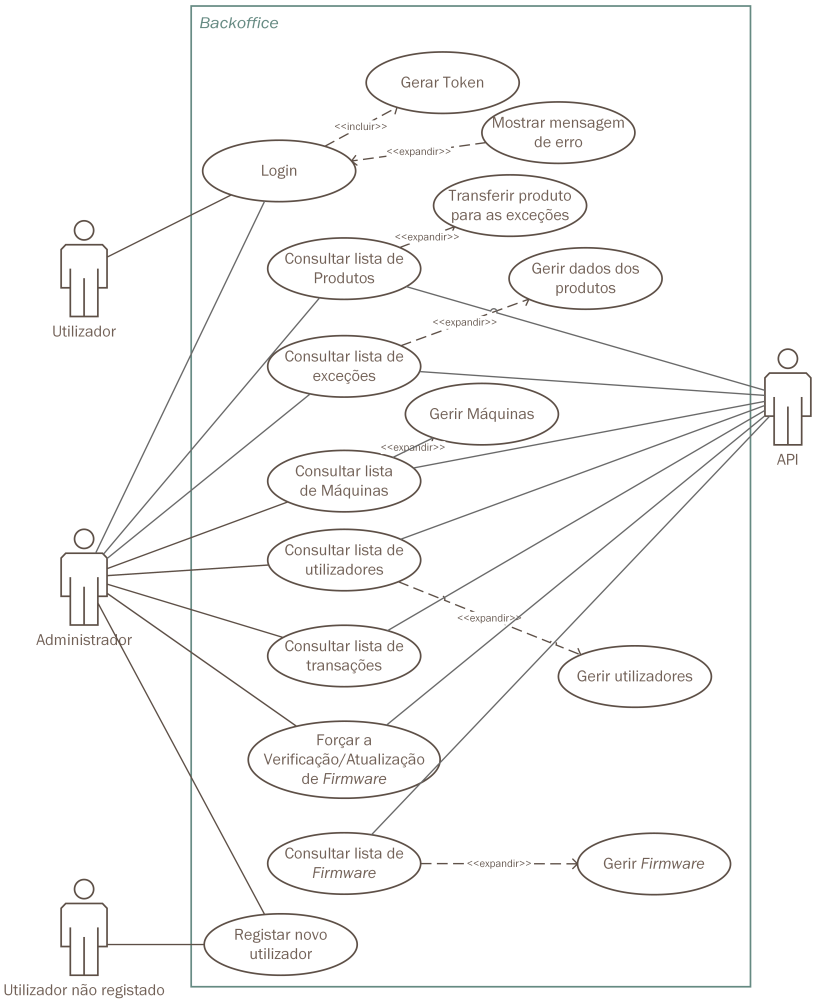


Figura 3.2: Diagrama de casos de uso do Back-office

No diagrama representado na Figura 3.2, é possível verificar todas as funcionalidades definidas na Tabela 3.1. Algumas funcionalidades são partilhadas pelos vários atores. Todas a funcionalidades são fornecidas pela API do serviço, de forma a garantir segurança e evitar ligações diretas dos vários atores à base de dados.

3.3.2 Posto de Embalagem e Estação de Carregamento de Firmware

Na Tabela 3.1 encontram-se os casos de uso do posto de embalagem e da estação de carregamento de *firmware*.

Tabela 3.2: Casos de Uso Posto de Embalagem e Estação de Carregamento de *Firmware*

Ator	**Casos de Uso**
Posto de Embalagem Estação de Carregamento de *Firmware*	Obter *token*
	Obter dados do produto
	Fazer a descarga do *firmware* para o posto de embalagem
	Fazer o carregamento de *firmware* para a câmara
API REST	Verificar se as máquinas têm acesso
	Fornece dados do produto e *firmware*

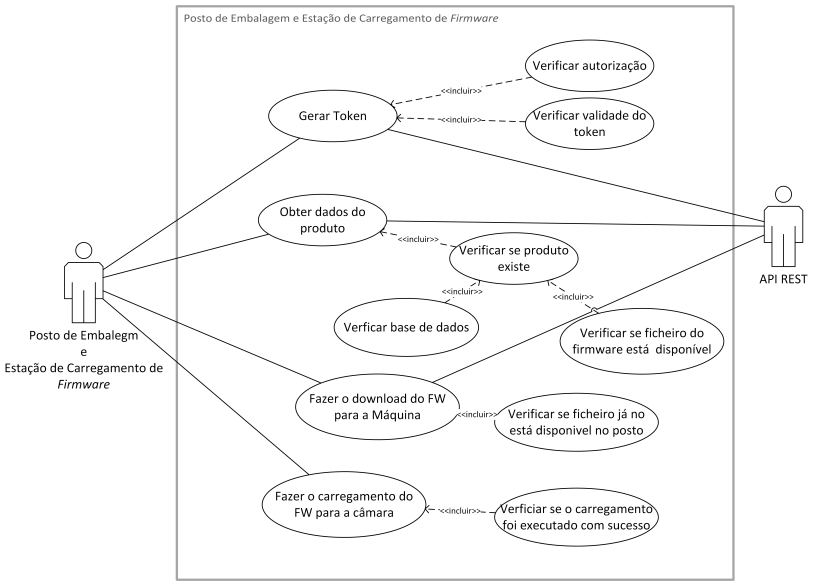


Figura 3.3: Diagrama casos de uso do Posto de Embalagem e Estação de Carregamento de *Firmware*

Na Figura 3.3, é possível visualizar os casos de uso referidos na Tabela 3.2.

3.4 Arquitetura

Para cumprir os requisitos definidos para o projeto, na Figura 3.4 é possível visualizar a nova arquitetura do sistema. Esta arquitetura permite a gestão automática do processo, excetuando casos especiais de produtos e produtos ainda em fase de industrialização.

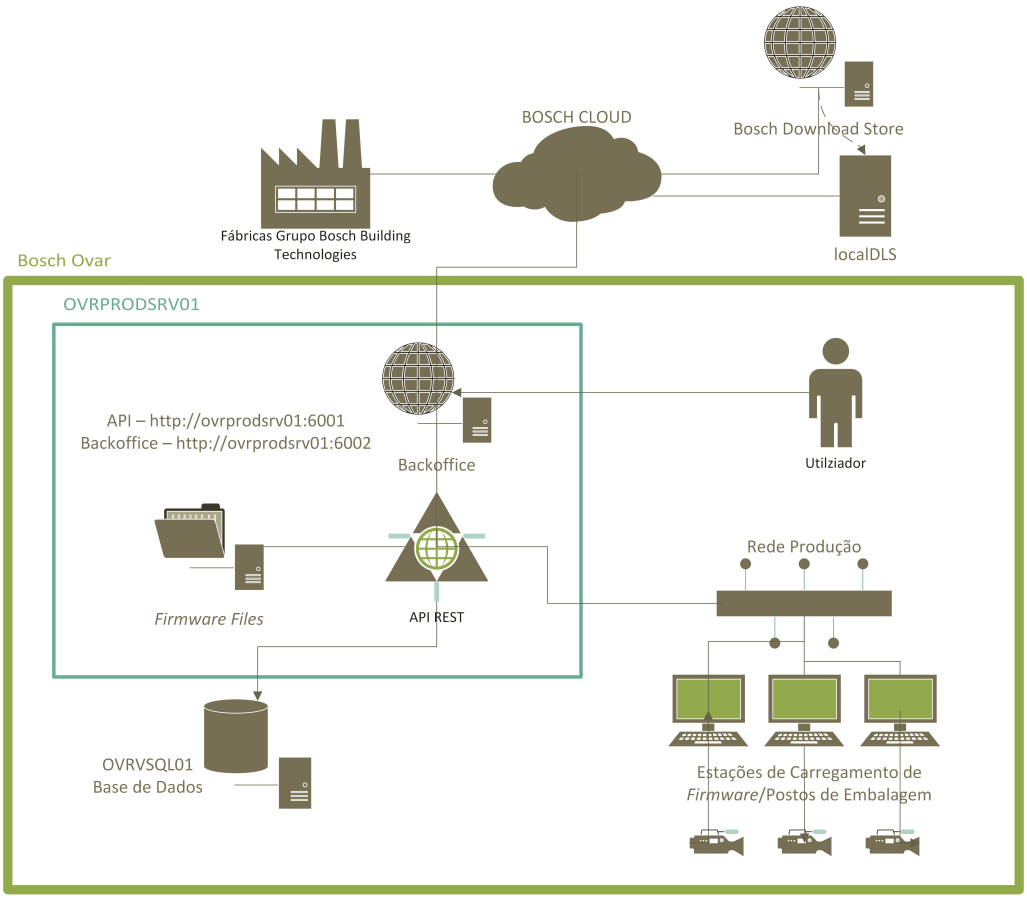


Figura 3.4: Arquitetura Proposta

Como é possível verificar na arquitetura, foi criado pela Bosch Nuremberga, um servidor cujo nome definido é `LocalDLS` (Local Download Store), que é uma réplica a nível de ficheiros do servidor Bosch Download Store. Neste servidor é atualizada a lista de produtos através do ficheiro `version_table.txt` e são disponibilizados todos os *release* de *firmware* que também são disponibilizados no servidor *Web* Bosch Donwload Store. Para aceder a este servidor é necessário estar ligado à Bosch Cloud e ter acesso ao servidor localDLS. A API REST é responsável por manter o serviço `OvrP Central System` sempre com as atualizações mais recentes de *firmware*, o que implica que a API verifique todos os dias

se existem alterações na lista de produtos como já mencionado anteriormente. Para consultar todas as informações de cada produto, será criado um *back-office*, que se divide em dois, o *back-end* e *front-end*. Mediante o nível de acesso, no *front-end* serão exibidos ou ocultados menus. No caso de ser `admin`, poderá aceder a todos os menus e no caso de ser `viewr` apenas terá acesso ao menu com a lista de produtos. O *font-end* disponibiliza toda a informação do serviço através de uma página *Web*, onde os administradores podem fazer a gestão de utilizadores, máquinas, *firmware* e produtos. Os restantes utilizadores podem consultar os dados de versões de *firmware* associados a cada produto. Os casos já foram abordados nos casos de uso do *back-office*. De seguida estão explanadas com mais detalhe as várias funcionalidades implementadas, cujo administrador do sistema tem acesso. O *firmware* das câmaras está divido em 3 ficheiros. O `App1`, `App0` e `bootloader`. Durante o processo produtivo é carregado um ficheiro de *firmware* via USB que contém num só ficheiro as 3 partes. Nas estações de carregamento de *firmware* e nos postos de embalagem, sempre que existe uma atualização, é habitual atualizar apenas a `App1`. Por vezes, devido a problemas de arranque da câmara no mercado, problemas na partição de *recovery* ou quando é necessária alguma correção a nível de segurança, é também necessário atualizar o `App0` e `Bootloader`. O serviço desenvolvido também suporta pedidos de outras fábricas, sendo que para isso apenas é necessário adicionar as máquinas dos outros locais de produção. Neste caso, quando se fornecem outras fábricas estas apenas têm acesso à API sendo que o *back-office* não está disponível.

Por forma a evitar que esta nova arquitetura seja causadora de interferência nas linhas de produção, para as linhas em que o tempo de ciclo de produção de uma câmara é inferior a 5 min utiliza-se o posto de carregamento de *firmware* e para as linhas em que o tempo de ciclo é superior a 5 min é utilizado o posto de embalagem. A vantagem de utilizar o posto de carregamento de *firmware* é que este permite fazer carregamento até quatro câmaras em simultâneo, enquanto o posto de embalagem apenas permite fazer o carregamento a uma câmara. Nos casos em que se utiliza a estação de carregamento de *firmware*, o posto de embalagem apenas verifica que a câmara passou no posto anterior e que tem a versão correta de *firmware*.

3.4.1 Back-office

De forma a garantir estabilidade do serviço criado, foi decidido que todos os pedidos à base de dados seriam processados pela API. Com isto, apenas a API necessita de ter acesso à base de dados de produção, garantindo que todos os pedidos passam por esta. Assim sendo, para além da segurança já fornecida pelo SQL Server, é também garantido que todas as operações necessárias processar na base de dados são sempre executadas da mesma forma, uma vez que os utilizadores e administradoras apenas acedem a rotas e não à base de dados diretamente. Desta

forma, o *back-office* quando é carregado ou tem que fazer uma alteração de dados, este faz pedidos `HTTP GET`, `POST`, `PUT` e `DELETE` à API REST através dos URI expostos pela API, como por exemplo, pedido GET `...:6001/device/<sap>`. Este pedido devolve todos os dados referentes ao código SAP passado (`SAP` é o campo que identifica o produto), evitando assim que o *back-office* aceda diretamente ao motor da base de dados.

3.4.1.1 Administrador

- Lista de Produtos:
  - Visualização de lista de produtos. Nesta tabela é possível visualizar os seguintes campos: `Produtct ID`, `Variant ID`, `SAP`, `App1`, `App0`, `Bootloader`, `App1 Version`, `App0 Version` e `Bootloader Version`.
  - Transferência de produtos para a lista de exceções, caso necessário. Isto permite que sejam detetados e temporariamente corrigidos erros até que a Bosch Nuremberga faça as devidas atualizações.
  - Verificação forçada de atualizações. É possível manualmente forçar o *back-office* a verificar a existência de atualizações do sistema.
  - Possibilidade de efetuar a descarga dos ficheiros de *firmware* para o computador pessoal.
  - Consulta do histórico de atualizações por produto.

- Lista de Exceções de Produtos/ Lista de Produtos em Industrialização:
  - Visualização da lista de exceções de produtos. Nesta tabela é possível visualizar os seguintes campos: `Produtct ID`, `Variant ID`, `SAP`, `App1`, `App0`, `Bootloader`, `App1 Version`, `App0 Version` e `Bootloader Version`;
  - Adição manual de produtos à tabela da `pack_version_table`. O administrador tem a possibilidade de carregar manualmente os ficheiros de *firmware* que serão utilizados num determinado produto;
  - Verificação automatica da versão de *firmware*, com base nos nomes, de cada um ficheiro de *firmware* carregado pelo *back-office*.

- Lista de *Firmware*:
  - Consultar todas as versões de *firmware* separadamente por tipo;
  - Adicionar novas versões de *firmware*;
  - Editar versões de *firmware*;

  - Apagar ficheiros de *firmware*.

- Lista de Utilizadores:

  - Adicionar novos utilizadores. Quando é adicionado um novo utilizador pelo administrador, este insere o *email* do novo utilizador e seleciona se este utilizador pode aceder ao *back-office* e qual os privilégios que tem, nomeadamente se é administrador ou não. Depois de criada a conta, o novo utilizador recebe um *email* a notificar que lhe foi criada uma conta no *back-office* e com uma *password* gerada aleatoriamente com 8 caracteres;
  - Editar os utilizadores;
  - Apagar utilizadores.

- Lista de Máquinas:

  - É neste menu que o administrador adiciona as máquinas do Posto de Embalagem e da Estação de Carregamento de *firmware* que têm acesso à API;
  - Editar as várias máquinas;
  - Apagar máquinas.

- Lista de Transações:

  - Consultar todas as transações e erros que ocorreram na API REST e no *Back-office*.

3.4.1.2 Utilizador

- Lista de Produtos:

  - Visualização de lista de produtos. Nesta tabela é possível visualizar os seguintes campos: `Produtct ID`, `Variant ID`, `SAP`, `App1 Version`, `App0 Version` e `Bootloader Version`
  - Consultar o histórico de atualizações por produto;
  - Consultar o estado da última verificação.

3.4.2 Posto de Embalagem

O posto de embalagem foi alterado por forma a funcionar de forma autónoma. Como principal função, o posto de embalagem imprime a etiqueta da caixa de um produto e regista na tabela de produtos produzidos da base de dados de produção que uma determinada câmara foi produzida e embalada para ser enviada para cliente com sucesso. Com o surgimento das novas necessidades de funcionalidades, como já descrito anteriormente, esta estação passou a incorporar também o carregamento de *firmware* e de certificados Bosch. Como já descrito, todo este sistema era manual, e quer a lista de produtos que um determinado posto pode fazer quer o *firmware* a utilizar eram configurados manualmente pelo administrador do sistema. O seu funcionamento antes das alterações centrava-se no carregamento de *firmware* definido num ficheiro de texto ou CSV, mediante o *firmware* definido para uma determinada diversidade. Para iniciar o processo do posto de embalagem, após se ligar uma câmara ao *Power Over Ethernet* (POE), o posto espera pelo arranque da mesma, lê os `ID` e verificar se correspondem à diversidade em produção, ou seja, se a diversidade está correta. Verifica qual a versão de *firmware* da câmara e compara com a definida manualmente no ficheiro de configuração do posto de embalagem. Em caso de diferenças, atualiza para a nova versão.

Na Figura 3.5 é possível verificar o fluxo do processo antigo do posto de embalagem. Como é possível verificar, o posto de embalagem em todas a interações verifica sempre o ficheiro de configuração onde está a lista de produtos e respetivas versões de *firmware*. Era um processo simples, mas como já referido várias vezes, era também um processo passível de vários erros, uma vez que era um processo demasiado manual.

Uma vez que não existia a necessidade de alterar completamente o posto de embalagem, a alteração centrou-se no módulo que faz a gestão do *firmware* e que verifica se uma câmara tem que ser atualizada. O *software* criado para substituir o *software* fornecido pelo centro de desenvolvimento tem as seguintes caraterísticas:

- Executar pedidos efetuados à API REST;
- Pedir os dados de um produto à API REST quando se terminar uma caixa múltipla de modo a evitar que as câmaras de um mesmo lote de produção cheguem ao cliente com diferentes versões de *firmware*;
- Pedir o *token* de acesso aos URI disponibilizados pela API no início da produção, que tem uma validade de 24 horas. O posto de embalagem verifica periodicamente quando necessita de pedir novo *token*, diminuindo o número de ligações à API;

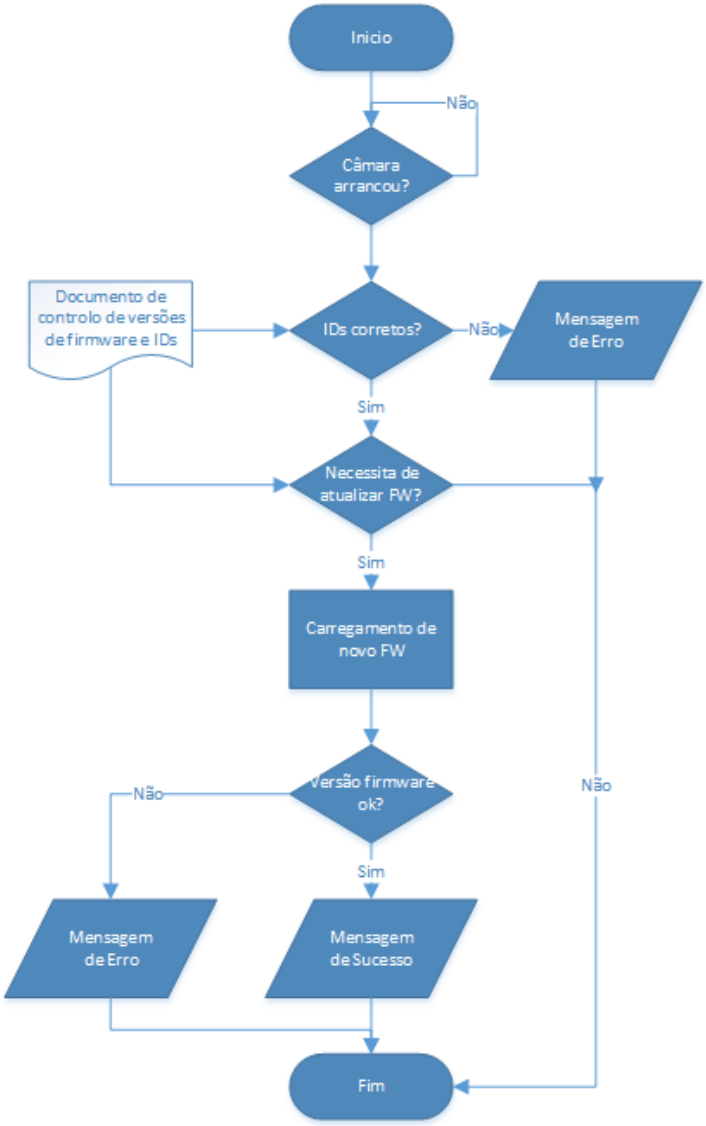


Figura 3.5: Fluxograma Posto Embalagem - Funcionamento Offline

- Verificar se a última versão do *firmware* de um determinado produto se encontra no disco local antes da efetuar qualquer descarga.
- Descarregar um ficheiro de *firmware*, efetuando um pedido à API com o URI correto.

Como é possível visualizar no diagrama da Figura 3.6, o posto de embalagem é completamente autónomo. De forma a garantir que não há câmaras com diferentes versões de *firmware* num lote, o posto de embalagem pede, no início de cada lote de produção, informação sobre o dispositivo à API REST. Descrevendo

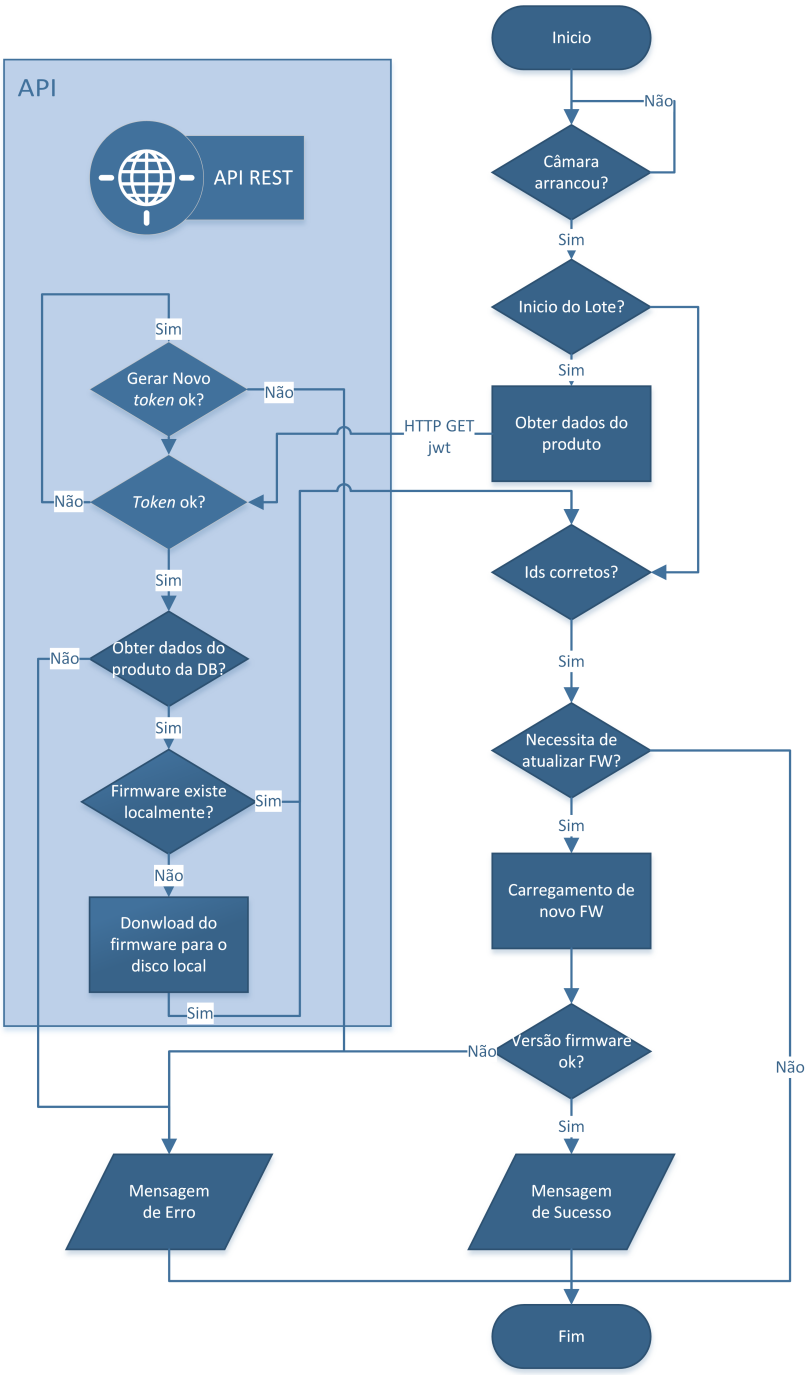


Figura 3.6: Fluxograma Posto Embalagem - Funcionamento com API

o fluxo, após ser conectada uma câmara ao *switch* POE do posto de embalagem, este espera pelo arranque da câmara. Depois de a câmara arrancar, o posto de embalagem verifica se os ID da câmara correspondem aos ID da diversidade na

qual se fez *set-up* no posto de embalagem. Após este processo, se for no início de um novo lote, o posto de embalagem faz um pedido `HTTP GET` à API REST de forma a obter a versão correta de *firmware* a utilizar. Caso o posto de embalagem esteja corretamente adicionado à lista de máquinas com permissão para aceder à API REST, esta retorna a versão correta de *firmware* a utilizar quer para a `App1`, `App0` e `Bootloader`. O acesso é validado pelo *token* enviado no cabeçalho do pedido HTTP. Caso o *token* tenha expirado, a API gera novo *token* e retorna-o ao posto de embalagem, para que nos próximos pedidos não seja necessário estar sempre a fazer pedido de *token*. Após retornar a versão de *firmware* a utilizar, o posto de embalagem verificr se o ficheiro de *firmware* já está disponível localmente. Em caso positivo procede ao carregamento do novo *firmware* e em caso negativo faz o *download* do ficheiro de *firmware* através do pedido `HTTP GET` à API REST. Após terminar o carregamento de *firmware*, é necessário esperar pelo *reboot* da câmara, e só depois do *reboot* é que se verifica se o carregamento correu conforme o esperado, caso contrário o produto é rejeitado.

3.4.3 Estação de Carregamento de Firmware

Esta estação foi concebida para libertar o posto de embalagem em linhas cuja produção hora seja muito elevada. Este posto tem como grande vantagem permitir fazer o carregamento de *firmware* em simultâneo até quatro dispositivos, como já referido anteriormente, fazendo com que o tempo de ciclo seja reduzido significativamente. Uma vez que foi desenvolvido de raiz, foi necessária fazer toda a lógica, bem como prever todos os estados e erros. Este posto faz o registo de todas as câmaras na base de dados de produção, como todos os dados de identificação, os certificados carregados, o *firmware* inicial e o *firmware* depois do carregamento. O *firmware* é registado separadamente, ou seja, é feito o registo das versões que do `App1`, `App0` e do `Bootloader` para além da informação básica que identifica o dispositivo como o número de série e MAC *address*.

3.4.4 API REST

As interfaces REST constituem um padrão para serviços *web*, permitindo a criação de serviços com recurso ao que o protocolo HTTP tem para oferecer, como forma de comunicação entre aplicações.

A API REST é o serviço responsável por garantir o acesso correto aos dados do sistema, com segurança e de forma eficaz e padronizada. Desta forma, todos os pedidos quer sejam provenientes do *back-office*, quer sejam provenientes das máquinas de produção, são sempre realizados através das rotas disponibilizadas pela API REST, que por sua vez responde aos pedidos com recurso ao formato de mensagens JSON. Pedidos HTTP implementados na API REST: `GET`, `DELETE`, `POST` e `PUT`. Como já referido, a API é responsável por processar os pedidos dos

utilizadores ou máquinas, consultar tabelas da base de dados, adicionar dados a tabelas, atualizar dados e apagar registos. Desta forma garante-se que não existe a possibilidade de fazer erros ao aceder à base de dados, pois todos as *queries* são sempre executadas da mesma forma e pela mesma interface.

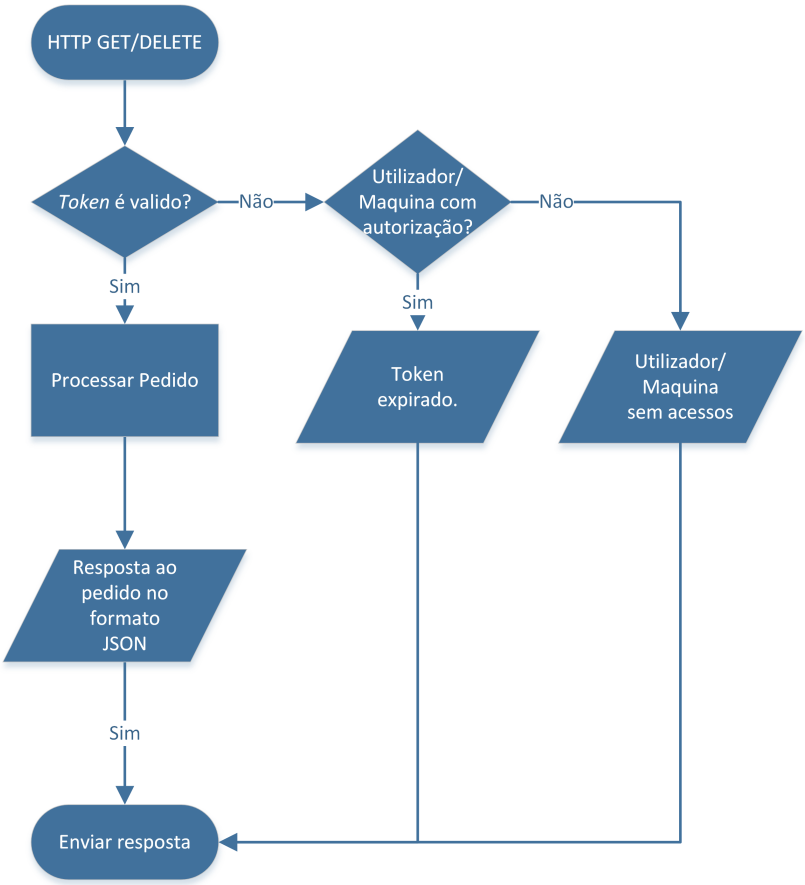


Figura 3.7: Fluxograma Pedido HTTP GET e DELETE

A Figura 3.7 apresenta o fluxograma de um pedido `HTTP GET` ou `DELETE`. Uma vez que neste pedido não existe *body* da mensagem, os dados são passados por parâmetros no URI do pedido, isto é, para um pedido GET obter os dados de um determinado produto, a composição do URI é `http://...:6001/device/F.01U.321.597`. O pedido `HTTP DELETE` funciona exatamente da mesma forma, isto é, o pedido HTTP tem o mesmo formato.

Quanto aos pedidos `HTTP POST` e `PUT`, como já refiro anteriormente, o pedido `HTTP POST` é executado quando uma aplicação ou utilizador querem adicionar uma entrada nova e o pedido `HTTP PUT` é utilizado quando é necessário atualizar dados já existentes, neste caso, na base de dados de produção. Na figura 3.8 é possível visualizar o fluxograma dos pedidos `HTTP POST` e `PUT`.

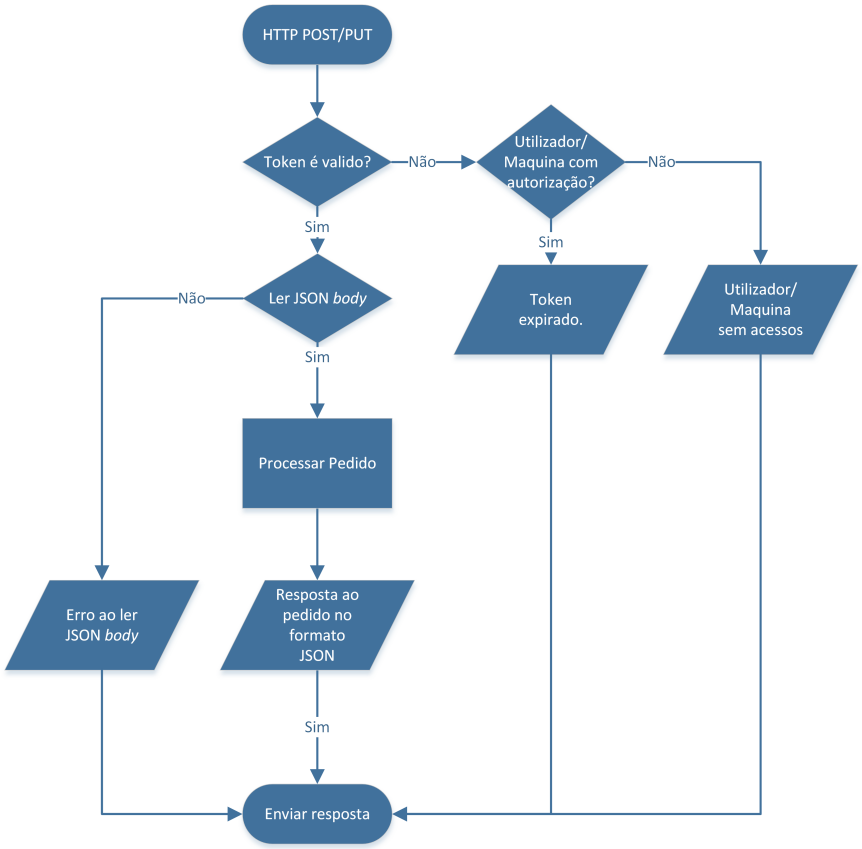


Figura 3.8: Fluxograma Pedido HTTP POST e PUT

Através dos fluxogramas dos diagramas anteriores, pode observar-se que antes de processar o pedido, a API REST verifica sempre se no *header* do pedido existe o campo `x-access-token`, campo este onde é passado o *token* e se o *token* é válido. Caso não exista *token* ou o utilizador/máquina não tenham permissão, a API REST responde com uma mensagem de estado HTTP 401 - Acesso não autorizado. Caso o *token* enviado já tenha expirado a sessão, a API REST responde com a mensagem de estado `HTTP 401 - Token expired`.

Quanto ao módulo que verifica periodicamente a existência de atualizações ou novas entradas, este é chamado através da API REST, quer seja a pedido do administrador através do *back-office* do sistema, ou pela função periódica implementada na API. É a API que executa todos os dias esta tarefa. Foi decidido desta forma, uma vez que as funções desenvolvidas para comunicar com as tabelas da base de dados estão implementadas na API, faz sentido criar um função periódica que chama a função responsável pela execução das várias funções que executam esta tarefa. Desta forma, está tudo a correr na mesma aplicação, evitando criar trocas de mensagens entre diferentes aplicações para fazer esta operação.

3.5 Sumário

Para garantir que os produtos produzidos em Ovar são sempre embalados com a última versão de *firmware* decidiu-se automatizar o processo de carregamento de *firmware* para câmaras de videovigilância através de uma API REST. A API verifica se existem versões novas de *firmware*, disponibiliza rotas de consulta, alteração, criação e eliminação de dados ao *back-office*, postos de embalagem e estações de carregamento de *firmware*. Esta solução aumenta a segurança do sistema, através do recurso a *tokens*, limita o número de utilizadores e dispositivos autorizados e diminui o volume de dados na rede da Bosch Security Systems, pois só a API faz *queries* à base de dados de produção. O serviço implementado também pode ser consumido noutras fábricas da Bosch. Nesse caso, não é necessário implementar HTTPS porque tanto o serviço como as fábricas estão ligadas à rede Bosch e não à Internet. Na Figura 3.9 está representado o Diagrama de Instalação UML da nova arquitetura do sistema, onde se pode ter uma visão mais detalhada e técnica de todo o sistema.

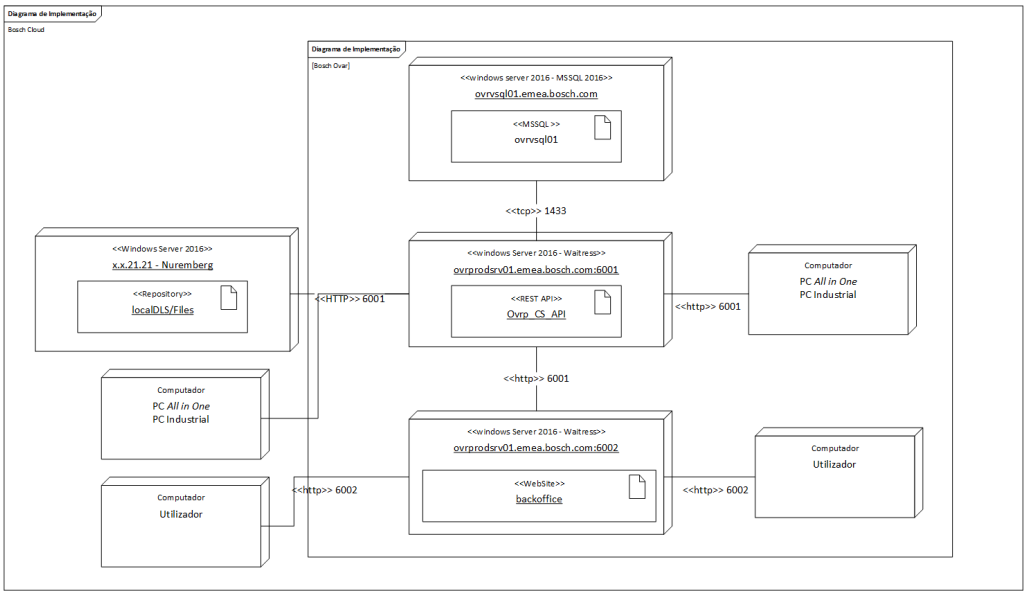


Figura 3.9: Diagrama UML de Instalação

Capítulo 4

Desenvolvimento da Solução

Neste capítulo serão apresentados todos os aspetos mais importantes do desenvolvimento da solução, nomeadamente da API REST, Back-office, Carregamento de Firmware e Estação de Carregamento de Firmware

4.1 Modelo de Dados

Através de *software* de gestão de base de dados, como é o caso do Microsoft SQL Server Management Studio, é possível gerar o diagrama da base de dados automaticamente. Neste caso, e uma vez que o controlo da base de dados é da responsabilidade do gestor de ITM da Bosch Security Systems, a nível de facilitar o desenvolvimento do projeto foi inicialmente desenvolvida toda a estrutura de base de dados em MySQL. Desta forma, foi possível fazer alterações e melhorias antes de fazer o *deploy* da aplicação em produção sem a necessidade da intervenção do responsável pela base de dados de produção da Bosch Security Systems, sempre que existia a necessidade de alterar as tabelas. Após validação de toda a estrutura, foi convertida a estrutura de MySQL para Microsoft SQL Server. Pare ter uma visão geral da estrutura da base de dados foi gerado um diagrama *Enhanced Entity-Relationship* (EER) que pode ser visualizado na Figura 4.1. Este diagrama foi gerado com recurso ao MySQL Workbench, sendo possível verificar qual o tipo de relação entres as várias tabelas. De forma a evitar duplicação de dados, as tabelas estão relacionadas entre si, quando possível. A estrutura será detalhada nas secções seguintes.

4.1.1 Relação entre Tabela users e Permissões

Nesta secção são apresentadas as tabelas `users` e `user_type` e a relação que existe entre elas. A relação entre estas tabelas é de um para muitos (Figura 4.2),

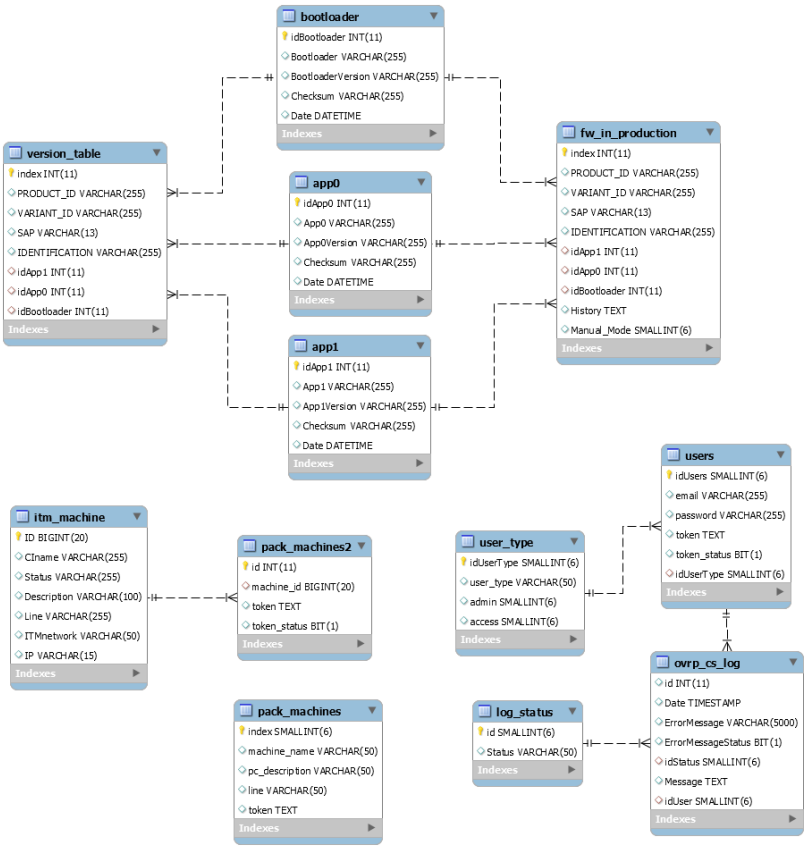


Figura 4.1: Diagrama EER

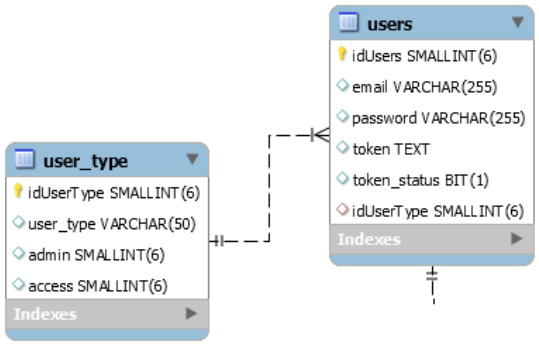


Figura 4.2: Relação tabela `users` e `user` type

ou seja, um utilizador apenas pode ter um tipo de acesso, mas vários utilizadores podem ter o mesmo tipo de acesso. Não existe a necessidade de criar uma tabela extra apenas para armazenar os utilizadores e as suas permissões, uma vez que

apenas podem ter um nível de permissão, sendo estes níveis `admin`, `viewer` e `notify`.

4.1.2 Relação entre as Tabelas Ovrp_CS_Log e Log_Status

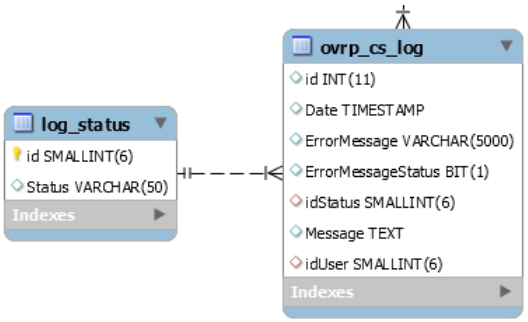


Figura 4.3: Relação tabela `Log_Status` e `OvrP_CS_Log` type

A relação entre estas tabelas é um para-muitos, como se pode confirmar pelo diagrama ER representado na Figura 4.3. Desta forma, um registo na tabela `Ovrp_CS_Log` pode ter apenas um estado da tabela `Log_Status`. Quanto à tabela `Log_Status`, vai armazenar todos os registos.

4.1.3 Relação entre Tabela version_table e Versões de Firmware

Esta secção apresenta a relação que existe entre a Tabela `version_table` e as tabelas `app1`, `app0` e `bootloader`. Na Figura 4.4, a relação entre a tabela `version_table` e as `App1`, `App0` e `Bootloader` é de um para muitos, isto quer dizer que, para cada produto, apenas existe uma versão de `App1`, `App0` e `Bootloader` possível. Os produtos podem partilhar a mesma versão de `App1`, `App0` e `Bootloader`. Os campos definidos para a tabela `version_table` são uma réplica do ficheiro `version_table.txt`.

4.1.4 Relação entre Tabela fw_in_production e Versões de Firmware

Esta secção apresenta a relação entre a tabela `fw_in_production` e as tabelas `App1`, `App0` e `Bootloader`. A tabela `fw_in_production` mantém atualizados os dados dos vários produtos e respetiva versão de *firmware* (`App1`, `App0` e `Bootloader`) que deve ser utilizado em produção. Na Figura 4.5, a relação entre a tabela `fw_in_Production` e as `App1`, `App0` e `Bootloader` é de um para muitos, isto quer dizer que para cada produto, apenas existe uma versão de `App0`, `App1` e `Bootloader` possível, como o que acontece na tabela `version_table` Esta tabela apenas é atualizada depois de a tabela `version_table` ter sido atualizada

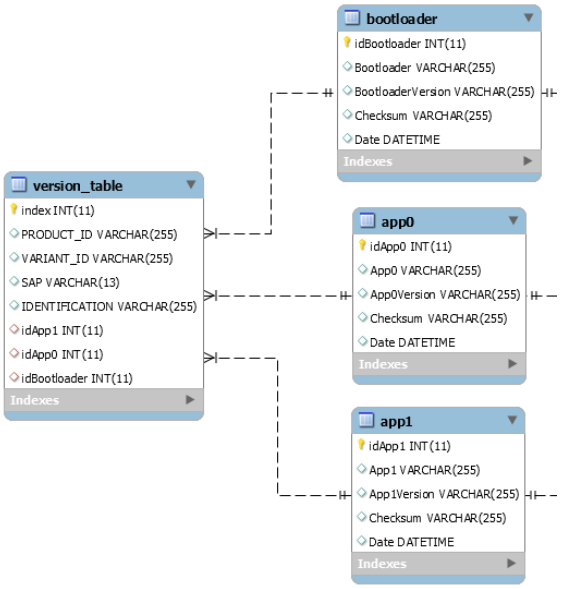


Figura 4.4: Relação tabela `version_table` e as versões do *firmware*

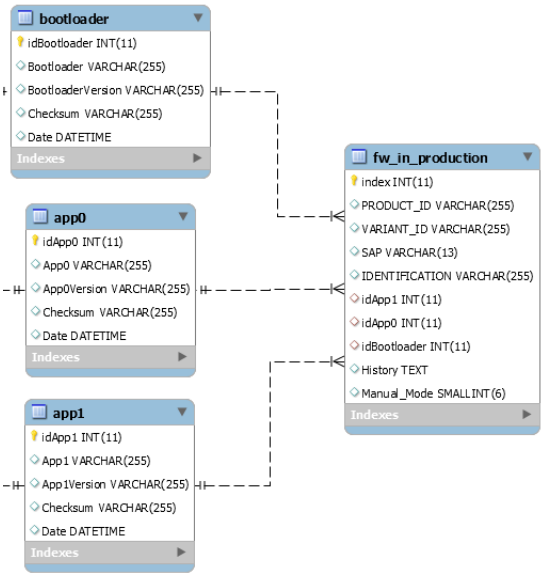


Figura 4.5: Relação tabela `fw_in_production` e versões do *firmware*

com sucesso, desta forma garante-se que nenhum dado está corrompido e que as conversões feitas durante o processo de atualização/verificação do sistema.

4.1.5 Tabela Independente

Para além das tabelas e relações descritas anteriormente, existe uma tabela que não tem relação com outras, mas que não deixa de ser tão importante. Esta tabela está representada na Figura 4.6.

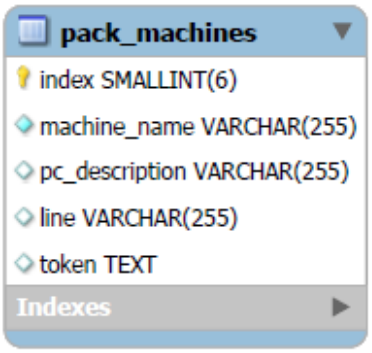


Figura 4.6: Tabelas `pack_machines`

A tabela `pack_machines` é utilizada para guardar todas as máquinas que não pertencem à tabela ITM.Machine. Desta forma, é utilizada para adicionar computadores que não pertençam à produção de Ovar, como os da engenharia de testes e também de outras fábricas da Bosch, caso seja necessário prestar este serviço a externos.

4.2 API

A implementação da API, foi executada com recurso à linguagem de programação Python e aos módulos Flask, Flask_AUTH, JWT, JSON, Requests, Waitress e o Apscheduler.

De forma a garantir a segurança de acesso à base de dados, que todas as *queries* e são executadas da mesma forma e de forma a automatizar todo o processo, foi necessário criar uma API. Esta API é responsável por executar todos os pedidos quer dos utilizadores do *back-office*, dos postos de embalagem e das estações de carregamento de *firmware*. Para além de satisfazer os pedidos, é a API que faz a verificação diária de atualizações de sistema de forma completamente autónoma. De forma aos utilizadores poderem consultar a base de dados através do *back-office* e os administradores de sistema, para além de consultar a base de dados, poderem adicionar, atualizar e apagar informação no *back-office*, a API disponibiliza todas as operações de consulta e gestão do sistema.

4.2.1 Ligação à Base de Dados

Para garantir que a API REST é independente do motor de base de dados utilizado, recorreu-se à biblioteca SQLAlchemy Object Relational Mapper (SQL ORM) para criar a abstração na criação de *queries*. Esta biblioteca faz com que as tabelas sejam representadas através de classes, que por sua vez contêm objetos com atributos. O primeiro passo consiste em criar a estrutura da base de dados. Uma vez que o sistema declarativo é normalmente utilizado pelo SQL ORM para definir as classes mapeadas para as tabelas de bases de dados relacionais, basta criar uma variável e defini-la como sistema base declarativo (`Base = declarative_base()`). Ao criar as várias classes referentes a cada tabela, coloca-se entre parênteses a variável, por exemplo, `Classe exemplo(Base)`. Assim, a classe passa a mapear uma tabela de uma base de dados relacional. De seguida, listam-se todas as classes de integração e comunicação com o motor de base de dados relacional criadas.

1. Classe `Version_Table`
    - Esta classe foi criada para mapear a tabela da base de dados `Version_table`. A classe criada está descrita no Anexo A.1.
2. Classe `FW_in_Production`
    - Esta classe faz o mapeamento da tabela `FW_in_Production`. A classe criada está descrita no Anexo A.2.
3. Classes `App1`, `App0` e `Bootloader`
    - Estas classes, foram implementadas para mapear as tabelas da base de dados correspondentes, nomeadamente, a tabela `app1`, `app0` e `bootloader`. A explicação da implementação destas classes está disponível no Anexo A.3.
4. Classes `Users` e `UsersType`
    - Estas classes fazem o mapeamento das tabelas da base de dados `Users` e `UsersType` e estão descritas no Anexo A.4.
5. Classe `Pack_Machines`
    - Esta classe faz o mapeamento da tabela `Pack_Machines` e está descrito no Anexo A.5.
6. Classes `Pack_Machines2` e `Machine`

- Estas classes foram criadas para mapear as tabelas da base de dados `Pack_Machines2` e `Machine`. A explciação das mesmas está descrita no Anexo A.6.

7. Classes `OvrP_CS_Log` e `Log_Status`

   - Estas classes foram criadas para mapear as tabelas da base de dados `OvrP_CS_Log` e `Log_Status`. Podem ser consultadas no Anexo A.7.

4.2.2 Rotas

Nesta secção descrevem-se as rotas implementadas e as funções desenvolvidas.

4.2.2.1 Rota Token

Para garantir a segurança e restringir o acesso a algumas rotas mediante o tipo de utilizador, recorreu-se ao JSON Web Token (JWT)[18]. A vantagem de utilizar *tokens* para garantir a segurança na comunicação entre a API e os clientes é que por natureza comunicações com recurso a *tokens* são eficientes e leves. Algumas caraterísticas importantes dos tokens são:

- Baseado no standard RFC7519 da W3C [19];
- Utilizado para realizar comunicações seguras de objetos JSON mesmo que utilizados em ligações HTTP;
- Um *token* é composto por um *header*, *payload* e uma assinatura;
- Incorpora toda a informação necessária para ser desencriptado.

Tabela 4.1: Rota Token

URI	/token
HTTP	POST
Headers	Content-Type: application/json x-access-key: <base64 encoded>

Request Body	{
	“name”: “<str>”
	}

Resposta	{
	“token”: “<token generated>”
	}

Na API desenvolvida, o *token* é gerado com recurso à biblioteca `jwt`, que tem classes de encriptação e desencriptação e exceções de tratamento de erros. Para aceder aos recursos do *token*, os utilizadores têm que estar registados na base de dados na tabela `users`. Quanto às máquinas dos postos de embalagem e estações de carregamento de *firmware*, apenas podem fazer pedidos à API se as mesmas estiverem registadas na tabela `pack_machines` ou `pack_machines2`.

Uma vez que o sistema é para ser utilizado apenas na rede Bosch, não surgiu a necessidade de implementar um outro tipo de segurança para aceder a esta API REST. Para fazer o pedido do *token*, é necessário respeitar a estrutura do pedido descrito na Tabela 4.1, bastando fazer um pedido `HTTP POST` ao recurso *token* (`http:.../token`). A API gera o *token* com base no *payload* enviado no pedido. No *payload* apenas é necessário enviar o campo `user_machine`. Uma das condições para gerar o *token*, é enviar a chave encriptada em base64 no campo `x-access-key` do *header*.

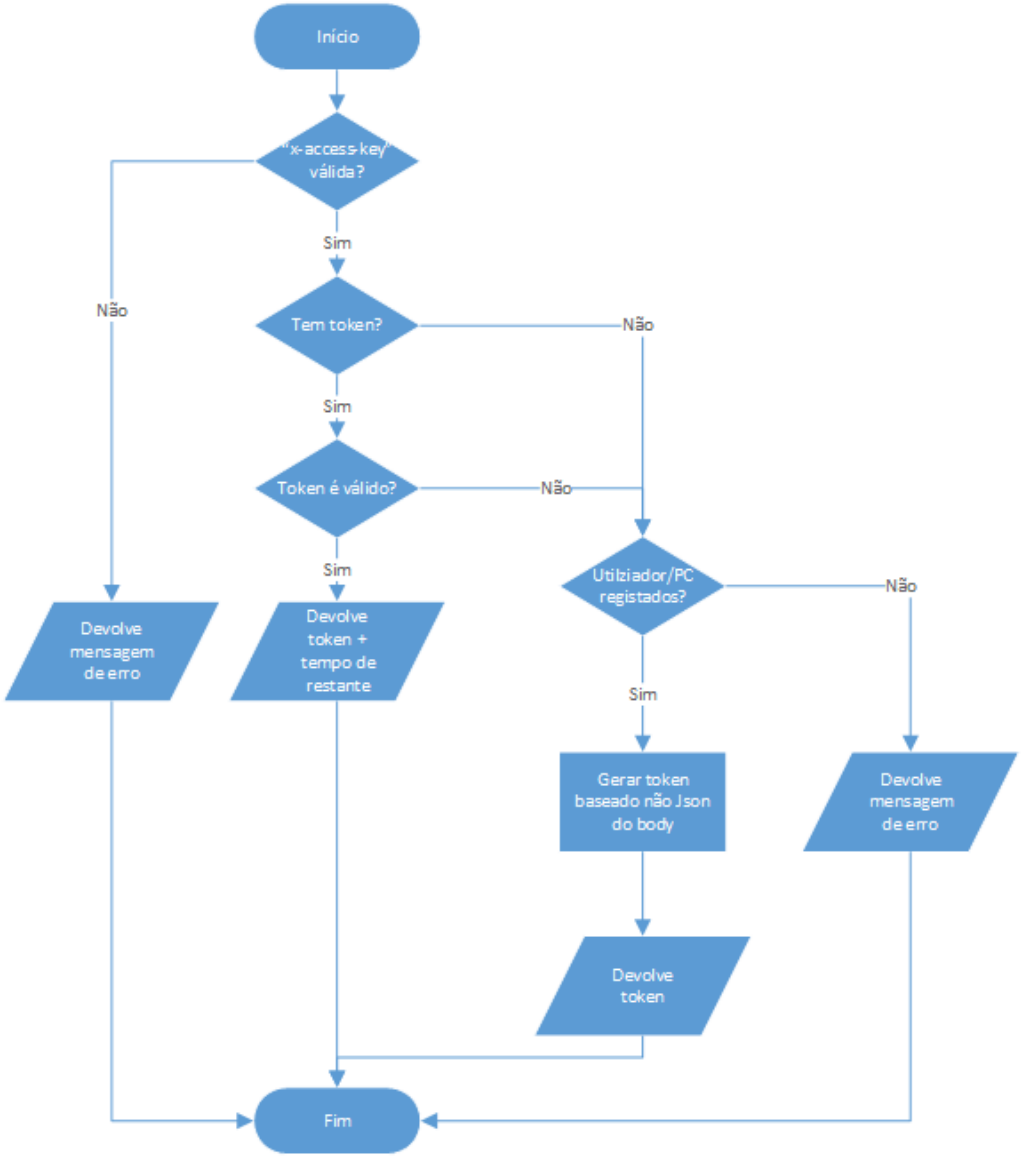


Figura 4.7: Fluxograma do pedido de um *token*

Na Figura 4.7 apresenta-se o fluxograma do pedido de um *token*. A primeira verificação é a validação da `x-access-key`. Em caso de validação, verifica se o utilizador ou máquina têm um *token* associado. Caso exista um *token* atribuído, verifica-se a sua duração. Caso ainda seja válido, a API responde com o mesmo *token* e o respetivo tempo de validade. Caso não exista *token*, a API verifica se o nome do utilizador ou da máquina presente no *payload* está registado nas tabelas de máquinas ou utilizadores. Se estiver, gera um *token* e envia-o ao utilizador. O *token* é guardado no campo `token` da tabela de utilizadores ou de máquinas. Se o utilizador ou máquina não estiver registado na base de dados, a API devolve uma mensagem de erro. O *token* gerado contém parâmetros seguintes:

- `user_machine`, que corresponde à máquina ou ao utilizador;
- `iat`, corresponde ao momento exato a que o token foi gerado;
- `exp`, corresponde à validade do *token*;
- `secrete_key`, é a chave secreta definida na API REST;
- `sgnature`, corresponde à assinatura para encriptar o token, que para esta situação é utilizado HS256.

De forma a evitar que operações sejam executadas de forma indevida, por exemplo, apagar registos da base de dados, as rotas definidas para processar pedidos `HTTP POST`, `PUT` ou `DELETE` obrigam que o utilizador tenha privilégios de administração. Para isso, foram criados dois tipos de validação de *token* antes de executar cada pedido. As funções criadas para este efeito são a `token_required` e `admin_toke_reqeuired`.

Na Figura 4.8 pode ser consultado o fluxograma da validação do *token* antes de executar um pedido HTTP. Para pedido `HTTP GET`, o utilizador ou máquina não necessitam de ser administradores do sistema. Neste caso, é utilizada a função `token_required`, que só permite à API responder ao pedido caso o *token* seja válido. Para aceder a rotas cujo o método `HTTP` seja `POST`, `PUT` ou `DELETE`, é utilizada a função `admin_token_required`, destinada apenas a administradores. A explicação mais detalhada de ambas está disponível no Anexo B.1.

4.2.2.2 Rota Device

A API fornece os métodos `HTTP GET`, `POST`, `PUT` e `DELETE` para esta rota. O pedido `HTTP GET` devolve toda a informação de produção referente ao produto em questão. Na Tabela 4.2 é possível visualizar a estrutura do pedido e respetiva resposta No *header* deste pedido é enviado o *token*. Como se trata de um `HTTP GET`, este não contém *body*, sendo que o dispositivo é identificado na construção do

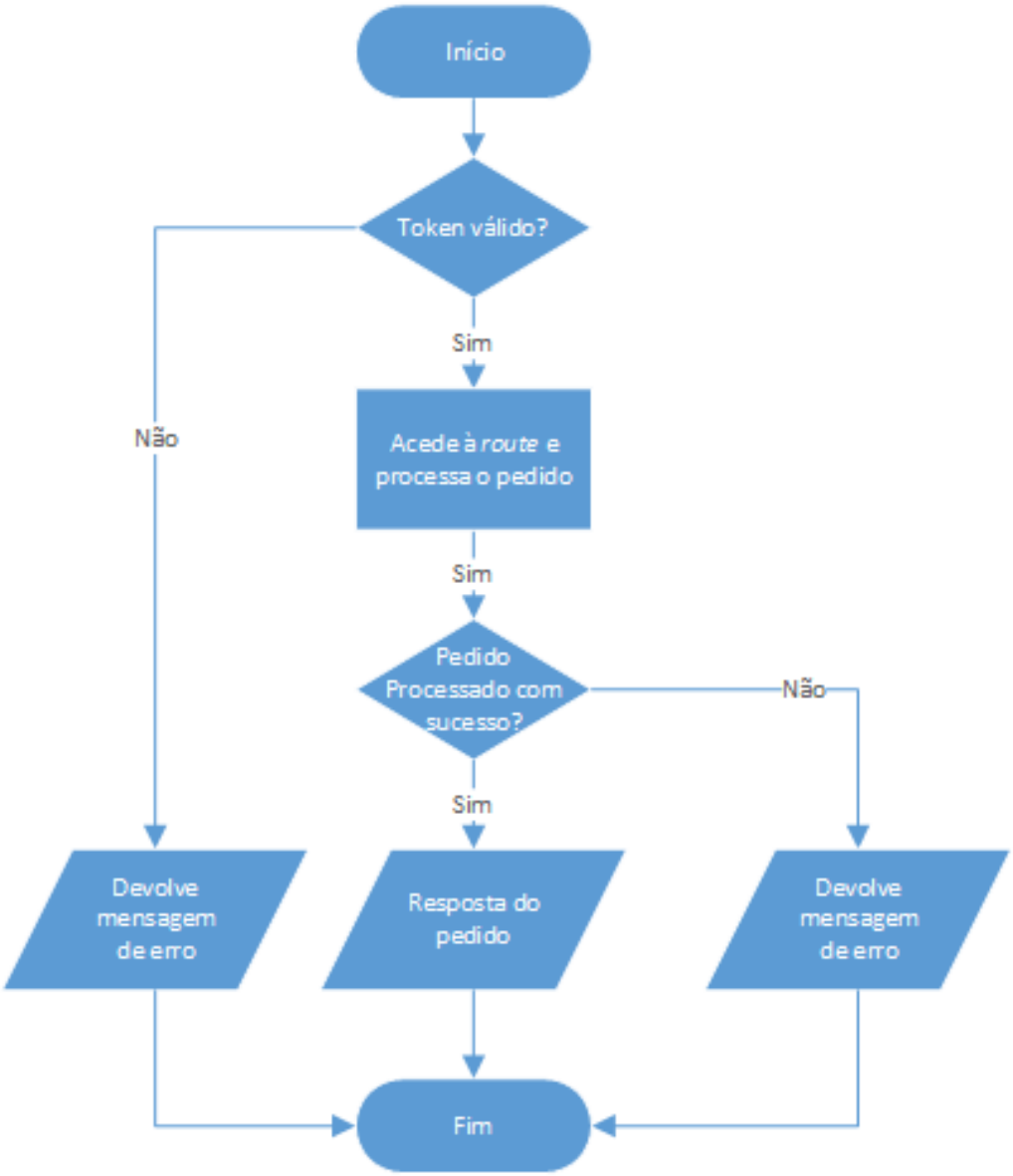


Figura 4.8: Routes - Token Required - Fluxograma

URI, device/<sap>. A variável `sap` identifica o produto (câmara). Na resposta, o *header* informa o cliente que o formato do *body* é em JSON, através do campo Content-Type. No *body* são devolvidos todos os campos referentes ao produto. No Anexo B.2.1.1 é possível verificar o Diagrama de Sequência UML deste método.

Na Tabela 4.3 é possível visualizar a estrutura do pedido `HTTP POST` e a respetiva resposta. Apenas tem permissão para executar com sucesso este pedido os utilizadores cujo nível de acesso é de administrador do sistema.

Tabela 4.2: HTTP GET Device

URI	/device/<sap>	
HTTP	GET	
	Pedido	**Resposta**
Header	x-access-token: <token>	Content-Type='application/json'

Body	NA	{
		"sap": "<str>",
		"variantid": "<str>",
		"productid": "<str>",
		"app1": "<str>",
		"app0": "<str>",
		"bootloader": "<str>",
		"app1version": "<str>",
		"app0version": "<str>",
		"bootloaderversion": "<str>"
		}

Tabela 4.3: HTTP POST Device

URI	/device/<device_id>	
HTTP	POST	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json'	Content-Type = 'application/json'

Body	{
	"sap": "<str>",
	"variantid": "<str>",
	"productid": "<str>",
	"App1": "<str>",
	"App0": "<str>",
	"Bootloader": "<str>",
	}
		{
		"Status": "<str>",
		}

Com este pedido é possível adicionar produtos ao sistema e definir manualmente qual a versão de *firmware* do mesmo. A necessidade de permitir esta gestão manual deve-se a requisitos internos da fábrica de Ovar. De forma a utilizar o sistema e validar os produtos durante a sua fase de industrialização, foi necessário adicionar esta funcionalidade. Caso contrário, não daria para produzir em fase de *sample Runs*, pois para produtos novos, o *firmware release* só fica disponível depois do início da produção em massa. Desta forma a única opção foi criar a possibilidade de adicionar manualmente produtos e também em algumas exceções alterar alguns produtos de produção para gestão manual. O Diagrama de Sequência UML deste pedido, pode ser consultado no Anexo B.2.1.2.

Na Tabela 4.4 é possível visualizar a estrutura do pedido `HTTP PUT`. Apenas têm permissão para executar com sucesso este pedido os utilizadores cujo nível de acesso é de administrador do sistema. Esta opção também surgiu devido aos requisitos mencionados anteriormente.

Este método é importante, pois permite aos administradores do sistema atualizar os dados de um determinado produto. Quer dizer que durante a industrialização de um produto, é possível alterar a versão de *firmware* durante os vários *sample runs*, que normalmente é necessário atualizar, pois são sempre introduzi-

Tabela 4.4: HTTP PUT Device

URI	/device/<device_id>	
HTTP	PUT	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json'	Content-Type = 'application/json'

Body	{
	"sap": "<str>",
	"variantid": "<str>",
	"productid": "<str>",
	"App1": "<str>",
	"App0": "<str>",
	"Bootloader": "<str>",
	"manual_mode":"<str>"
	}
		{
		"Status": "<str>",
		}

das funcionalidades que só ficam disponíveis em certas versões de um determinado *firmware*. Este pedido está descrito no diagrama de sequência do Anexo B.2.1.3.

Quanto ao pedido `HTTP DELETE`, este está descrito na Tabela 4.5. Como se trata de um `DELETE`, não é enviado qualquer *body*, sendo que a identificação do produto que se pretende apagar é enviada no URI, através da *tag* "<device_id>".

Tabela 4.5: HTTP DELETE Device

URI	/device/<device_id>	
HTTP	DELETE	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json'	Content-Type = 'application/json'

Body	"NA"	{
		"Status": "<str>",
		}

Para apagar um registo é feita uma *query* SQL à base de dados com o intuito de apagar da um determinado índice da tabela "fw_in_production", como é possível verificar no Diagrama de Sequência UML exposto no Anexo B.2.1.4.

4.2.2.3 Rota User

Nesta rota foram implementados os métodos `POST`, `PUT` e `DELETE`. Através desta rota é possível adicionar utilizadores ao sistema, alterar os dados dos utilizadores e apagar utilizadores sempre que seja necessário. Na Tabela 4.6 pode visualizar-se a estrutura do pedido `HTTP POST`. Desta forma é possível adicionar utilizadores de forma a garantir o acesso ao *back-office* e à API. A forma mais fácil de gerir o acesso à API e *back-office* foi através da criação de uma tabela apenas para utilizadores e não utilizar o `activedirectory` do domínio, pois a gestão dos vários níveis de acesso seria dificultada, uma vez que os grupos de utilizadores são geridos pela equipa central de IT da Bosch. Apenas os administradores do sistema podem criar utilizadores. Para criar um utilizador novo é necessário o email do utilizador e escolher o nível de acesso do mesmo, sendo que os níveis implementados são

o `Admin`, `Viewer` e `Notify`. Ao adicionar o utilizador, é gerada uma *password* aleatória que é enviada via email aquando da criação da conta. Para adicionar segurança, as *passwords* dos utilizadoras são encriptadas, sendo que o algoritmo escolhido foi o *hash* com recurso ao protocolo `sha256` e o identificador escolhido foi o `PBKDF2`. No Anexo B.5 é possível consultar o Diagrama de Sequência UML

Tabela 4.6: HTTP POST User

URI	/user/<user_id>	
HTTP	POST	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json' U = <email>	Content-Type = 'application/json'

Body	{
	"email": "<str>",
	"password": "<str>",
	"user_type": "<str>"
	}
		{
		"Status": "<str>",
		}

deste método.

Através do método `HTTP PUT`, a API permite atualizar os dados de todos os utilizadores do sistema, como por exemplo, alterar a *password*, o nível de acesso e até mesmo forçar um novo *token* em casos extremos. Na Tabela 4.7 está descrita a estrutura deste pedido. No Anexo B.2.2.2 é possível consultar o respetivo

Tabela 4.7: HTTP PUT User

URI	/user/<user_id>	
HTTP	PUT	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json' U = <email>	Content-Type = 'application/json'

Body	{
	"email": "<str>",
	"password": "<str>",
	"user_type":"<str>",
	"token":"<str>"
	}
		{
		"Status": "<str>",
		}

Diagrama de Sequência UML. Para apagar um utilizador do serviço, é necessário recorrer ao método `HTTP DELETE`. A estrutura do pedido está descrita na Tabela 4.8

Tabela 4.8: HTTP DELETE User

URI	/user/<user_id>	
HTTP	DELETE	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json' U = <email>	Content-Type = 'application/json'

Body	"NA"	{
		"Status": "<str>",
		}

No Anexo B.2.2.3 está disponível para consulta o Diagrama de Sequência UML do método DELETE.

Para redefinir a *password*, o utilizador pode fazê-lo através da rota /user/password/user_id. Este `HTTP PUT` verifica se a *password* atual está correta, e compara a nova *password* com a confirmação da nova *password*. Este pedido descrito está na Tabela 4.9 e foi implementado exclusivamente para ser utilizado através do *back-office*.

Tabela 4.9: HTTP PUT User password reset

URI	/user/password/<user_id>	
HTTP	PUT	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json'	Content-Type = 'application/json'

Body	{
	"currentpassword":<str>,
	"newpassword":<str>,
	"confirm_newpassword":<str>
	}
		{
		"status": "<str>"
		}

4.2.2.4 Rota Machine

Esta rota permite fazer a gestão de todas as máquinas do sistema. Através do método `POST` é possível adicionar máquinas ao serviço, como da mesma forma que através do `HTTP PUT` é possível alterar os dados de determinada máquina. Para remover as máquinas do sistema, foi implementado o método `HTTP DELETE`. Para melhor gestão do número de acessos ao serviço, foi decidido que apenas as máquinas registadas na base de dados têm acesso ao mesmo. Desta forma, adiciona-se um nível de segurança e fica-se com o registo de todas as máquinas ligadas ao sistema.

Uma vez que o sistema está concebido para trabalhar também noutras fábricas do grupo Bosch, foi decidido implementar duas formas de adicionar máquinas ao serviço. A primeira forma, referente às máquinas da fábrica de Ovar, em que para adicionar apenas é feita uma listagem de todas as máquinas existentes na fábrica e o administrador apenas seleciona a que quer dar acesso ao serviço. A segunda forma é referente a máquinas que estão noutras fábricas, na qual o serviço não tem acesso à lista de máquinas. Para esta situação é o administrador que tem que preencher manualmente o nome, descrição e linha de produção.

A estrutura do pedido `HTTP POST` está descrita na Tabela 4.10 e para a facilitar a implementação do *back-office*, a API através do campo `CIName` do *body* consegue fazer a distinção entre as ambas as situações. Se {"CIName":"None"}, quer dizer que a máquina que se pretende adicionar não pertence à fábrica de Ovar, caso contrário, pertence. Para fazer a distinção no pedido `HTTP POST`, caso uma máquina pertença a outra fábrica, o campo `CIName` do *body* é igual a `None`.

A estrutura deste pedido está visível na Tabela 4.10. Caso esteja na Bosch Ovar,

Tabela 4.10: HTTP POST Machine Non ITM

URI	/machine	
HTTP	POST	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json' U = <email>	Content-Type = 'application/json'

Body	{
	"machine_name": "<str>",
	"pc_description": "<str>",
	"line": "<str>"
	"CIName": "None"
	}
		{
		"Status": "<str>",
		}

apenas é enviado no *body* o campo `CIName` com o respetivo nome da máquina. A estrutura deste pedido pode ser consultada na Tabela 4.11. A descrição detalhada

Tabela 4.11: HTTP POST Machine ITM

URI	/machine	
HTTP	POST	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json' U = <email>	Content-Type = 'application/json'

Body	{
	"CIName": "<str>"
	}
		{
		"Status": "<str>",
		}

do pedido `HTTP POST`, está descrita no Diagrama de Sequência UML disponível no Anexo B.2.3.1.

Quanto à atualização de máquinas, apenas é possível para as que não pertencem à fábrica de Ovar. Recorrendo-se ao pedido `HTTP PUT`, pode-se atualizar os dados de uma determinada máquina, nomeadamente alterar o nome, a descrição, a linha e o *token*. Na Tabela 4.12 está especificada a estrutura referente a este pedido.

Tabela 4.12: HTTP PUT Machine

URI	/machine/<machine_id>	
HTTP	PUT	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json' U = <email>	Content-Type = 'application/json'

Body	{
	"machine_name": "<str>",
	"pc_description": "<str>",
	"line": "<str>"
	"CIName": "None"
	}
		{
		"Status": "<str>",
		}

Para se obter detalhe técnico deste pedido, o Diagrama de Sequência UML pode ser consultado no Anexo B.2.3.2. Quanto ao `HTTP DELETE`, a estrutura do

pedido está representada na Tabela 4.13. Para mais detalhes em relação a este

Tabela 4.13: HTTP DELETE Machine

URI	/machine/<machine_id>	
HTTP	DELETE	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json' U = <email>	Content-Type = 'application/json'

Body	"NA"	{
		"Status": "<str>",
		}

método, pode ser consultado o Anexo B.2.3.3.

4.2.2.5 Rota App0

Para a rota `App0`, foram implementados os métodos `GET`, `POST`, `PUT` e `DELETE` do protocolo HTTP. Como já mencionado, com estes métodos é permitido obter, adicionar, atualizar e remover os dados da base de dados de determinada versão de `App0`. Através do método `GET`, a API fornece ao utilizador/administrador a função de efetuar o *download* do ficheiro de *firmware* `..._app0_....fw`. Para este método, não é necessário ser administrador do sistema, pois não há o risco de apagar ou modificar alguma das entradas da base de dados. A estrutura deste método está descrita na Tabela 4.14.

Tabela 4.14: HTTP GET App0

URI	/app0/<app0_version>/<app0>	
HTTP	GET	
	Pedido	**Resposta**
Header	x-access-token: <token>	Content-Type='application/json'
Body	NA	{ "Status": "Firmware file not found" } ou Envia o ficheiro app0 para o destino

Todos os passos para efetuar o *download* de um ficheiro de *firmware* app0 estão descritos no Diagrama de Sequência UML disponível no Anexo B.2.4.1

Para adicionar um ficheiro de *firmware* `App0`, foi necessário implementar o método `HTTP POST`. Uma vez que os ficheiros da gestão automática de *firmware* têm sempre o mesmo formato, ou seja, plataforma, seguido da parte do *firmware* e respetiva versão, como por exemplo, "CPP7.3_app0_7.62.0003.fw", em que decompondo o nome, a plataforma corresponde ao `cpp7.3`, a parte do *firmware* corresponde ao `app0` e versão é `7.62.0003`, para adicionar manualmente um ficheiro é preciso garantir que o nome do mesmo está neste formato. A estrutura do pedido `HTTP POST` está explicada na Tabela 4.15.

No Anexo B.2.4.2 está descrito de uma forma mais detalhada este pedido bem como o Diagrama de Sequência UML.

Tabela 4.15: HTTP POST App0

URI	/app0	
HTTP	POST	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json' U = <email>	Content-Type = 'application/json'

Body	{
	"app0": "<str>"
	}
		{
		"Status": "<str>",
		}

Caso seja necessário atualizar os dados de uma dada entrada da tabela `app0`, a API fornece ao administrador essa opção através do pedido `HTTP PUT`. Se este pedido for executado através do *back-office*, a opção implementada é substituir o ficheiro de *firmware*. O resumo do pedido `HTTP PUT` está representado na Tabela 4.16 O campo `<app0_id>` identifica a entrada da tabela onde se pretende fazer a

Tabela 4.16: HTTP PUT App0

URI	/app0/<app0_id>	
HTTP	PUT	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json' U = <email>	Content-Type = 'application/json'

Body	{
	"App0": "<str>"
	}
		{
		"Status": "<str>",
		}

atualização. O Diagrama de Sequência UML deste pedido está representado no Anexo B.2.4.3.

Por último, para remover uma entrada da tabela, foi implementado o método `HTTP DELETE`. Para apagar uma entrada, basta fazer um pedido `HTTP DELETE` e no **URI** identificar a entrada, como se pode confirmar na Tabela 4.17

Tabela 4.17: HTTP DELETE App0

URI	/app0/<app0_id>	
HTTP	DELETE	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json' U = <email>	Content-Type = 'application/json'

Body	"NA"	{
		"Status": "<str>",
		}

O Diagrama de Sequência deste método está disponível no Anexo B.2.4.4.

4.2.2.6 Rota App1

Esta rota tem exatamente as mesmas funcionalidades da rota `App0`, isto é, os métodos HTTP implementados são `GET`, `POST`, `PUT` e `DELETE`. Deste forma, permite

adicionar, editar, apagar e obter um determinado *firmware*. A diferença comparando com o `App0`, é o nome do ficheiro, que nesta caso segue a mesma lógica do anterior, ou seja, plataforma, seguido da parte do *firmware* e respetiva versão, como por exemplo, `CPP7.3_H.264_7.62.0003.fw`, em que decompondo o nome, a plataforma corresponde ao `cpp7.3`, a parte do *firmware* corresponde ao `H.264` e versão é `7.62.0003`. Para o `App1`, no nome dos ficheiros consta sempre H.264/265 ou FW em vez de `App1`. A estrutura do pedido `HTTP GET` está representada na Tabela 4.18 e o Diagrama de Sequência UML está no Anexo B.2.5.1.

Tabela 4.18: HTTP GET App1

URI	/app1/<app1_version>/<app1>	
HTTP	GET	
	Pedido	**Resposta**
Header	x-access-token: <token>	Content-Type='application/json'
Body	NA	{ "Status": "Firmware file not found" } ou Envia o ficheiro `app1` para o destino

Quando é necessário adicionar manualmente um ficheiro de *firmware* correspondente à parte `App1`, faz-se um pedido `HTTP POST` com a estrutura da Tabela 4.19.

Tabela 4.19: HTTP POST App1

URI	/app1	
HTTP	POST	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json' U = <email>	Content-Type = 'application/json'

Body	{
	"app1": "<str>"
	}
		{
		"Status": "<str>",
		}

No Anexo B.2.5.2 está representado o Diagrama de Sequência UML com os detalhes deste pedido. Para atualizar/alterar um ficheiro de *firmware*, foi implementado o método `HTTP PUT`. A estrutura deste pedido está descrita na Tabela 4.20.

Tabela 4.20: HTTP PUT App1

URI	/app1/<app1_id>	
HTTP	PUT	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json' U = <email>	Content-Type = 'application/json'

Body	{
	"App1": "<str>"
	}
		{
		"Status": "<str>",
		}

Quanto ao Diagrama de Sequência UML, onde se pode verificar de uma forma mais detalhada todas as funções necessárias para responder a este pedido, o mesmo pode ser consultado no Anexo B.2.5.3.

O pedido implementado para remover um ficheiro de *firmware* App1, está definido na Tabela 4.21 e o Diagrama de Sequência onde se pode verificar de forma detalhada como é que o pedido é processado está disponível no Anexo B.2.5.4.

Tabela 4.21: HTTP DELETE App1

URI	/app1/<app1_id>	
HTTP	DELETE	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json' U = <email>	Content-Type = 'application/json'

Body	"NA"	{
		"Status": "<str>",
		}

4.2.2.7 Rota Bootloader

Para a rota `Bootloader`, foram também implementados os métodos `HTTP GET`, `POST`, `PUT` e `DELETE`. Na Tabela 4.22 está descrita a estrutura do pedido `HTTP GET`. No Anexo B.2.6.1 pode ser consultado o Diagrama de Sequência deste pedido. Este pedido permite ao utilizador ou às máquinas a funcionalidade de descarregar o ficheiro pretendido para um determinado produto.

Tabela 4.22: HTTP GET Bootloader

URI	/bootloader/<bootloader_version>/<bootoloader>	
HTTP	GET	
	Pedido	**Resposta**
Header	x-access-token: <token>	Content-Type='application/json'
Body	NA	{ "Status": "Firmware file not found" } ou Envia o ficheiro bootoloader para o destino

Na Tabela 4.23 está descrita a estrutura do pedido `HTTP POST`. Este pedido permite ao administrador adicionar manualmente novas versões de *firmware* referentes ao Bootloader. No Anexo B.2.6.2 pode ser consultado o Diagrama de Sequência deste pedido.

Tabela 4.23: HTTP POST Bootloader

URI	/bootloader	
HTTP	POST	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json' U = <email>	Content-Type = 'application/json'

Body	{
	"bootloader": "<str>"
	}
		{
		"Status": "<str>",
		}

Para atualizar uma entrada, na Tabela 4.24, está descrita a estrutura do pedido `HTTP PUT`. No Anexo B.2.6.1 pode ser consultado o Diagrama de Sequência deste pedido.

Tabela 4.24: HTTP PUT Bootloader

URI	/bootloader/<bootloader_id>	
HTTP	PUT	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json' U = <email>	Content-Type = 'application/json'

Body	{
	"Bootloader": "<str>"
	}
		{
		"Status": "<str>",
		}

Para apagar uma determinado ficheiro, basta aceder à rota através do pedido `HTTP DELETE`, cuja estrutura do mesmo está disponível na Tabela 4.25. O Diagrama de Sequência deste pedido está no Anexo B.2.6.4.

Tabela 4.25: HTTP DELETE Bootloader

URI	/bootloader/<bootloader_id>	
HTTP	DELETE	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json' U = <email>	Content-Type = 'application/json'

Body	"NA"	{
		"Status": "<str>",
		}

4.2.2.8 Rota pack_version_table

Um dos requisitos deste projeto foi definir uma hora a que o sistema vai verificar se existem novas versões ou novos produtos adicionados ao ficheiro `version_table.txt`, que está armazenado na `localDLS`. Durante o desenvolvimento do projeto, conclui-se que seria bom ter a possibilidade de forçar uma atualização a qualquer hora. Para isso foi implementada esta rota, contudo apenas os administradores do sistema podem fazer uma verificação de versões a qualquer momento. No caso de se verificar que existem atualizações, as mesmas são processadas e o sistema é atualizado de imediato.

Tabela 4.26: HTTP PUT Pack_Version_Table

URI	pack_version_table	
HTTP	PUT	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json' U = <email>	Content-Type = 'application/json'

Body	{
	"update": "yes"
	}
		{
		"Status": "<str>",
		}

Para proceder à atualização é necessário fazer um pedido `HTTP PUT`, com a estrutura da Tabela 4.26. É necessário enviar no *body* o campo `update` com o valor `yes`. A rota, ao receber este `HTTP PUT`, verifica se o *token* pertence a um `administrador`. Após esta validação, a API verifica o *body* do pedido e, caso o campo `update` contenha o valor `yes`, é executado o processo de verificação e atualização de *firmware*. Este processo é o mesmo que faz a verificação automática todos os dias. Ao utilizar as mesmas funções evitam-se problemas e duplicação de código. Desta forma, caso seja necessário atualizar alguma função de uma dada classe, fica disponível quer para a verificação automática quer para a manual. O Diagrama de Sequência UML deste pedido está descrito no Anexo B.23. Após a verificação, o resultado da operação e o utilizador que a efetuou são guardados na tabela `CS_Log`. O utilizador da verificação automática é o `system`.

Os fluxogramas seguintes detalham a obtenção e tratamento do ficheiro `version_table.txt`, seguido da comparação com a tabela `pack_version_table`.

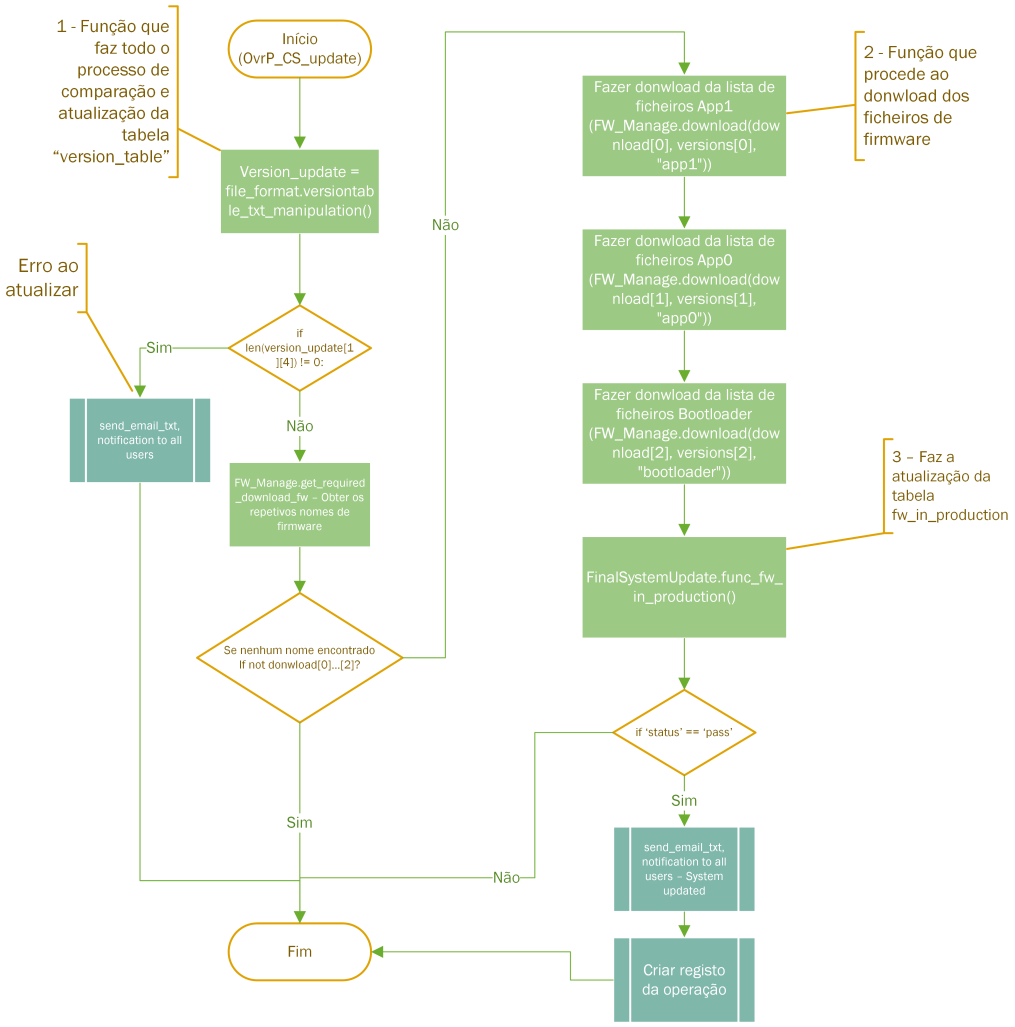


Figura 4.9: Fluxograma geral de verificação e atualização do sistema

Esta tabela é uma réplica da tabela `fw_in_production` de produção e pretende assegurar que a produção não é afetada por algum erro de conversão, ou alguma falha inesperada durante o processo de atualização da tabela. Em caso de erro a tabela de produção não é atualizada e os administradores são notificados por email do erro ocorrido durante a atualização. Esta operação crítica é registada na tabela `Ovrp_CS_Log`. Desta forma, evitam-se paragens de produção e garante-se que a atualização ocorre apenas quando todos os dados estão no formato correto.

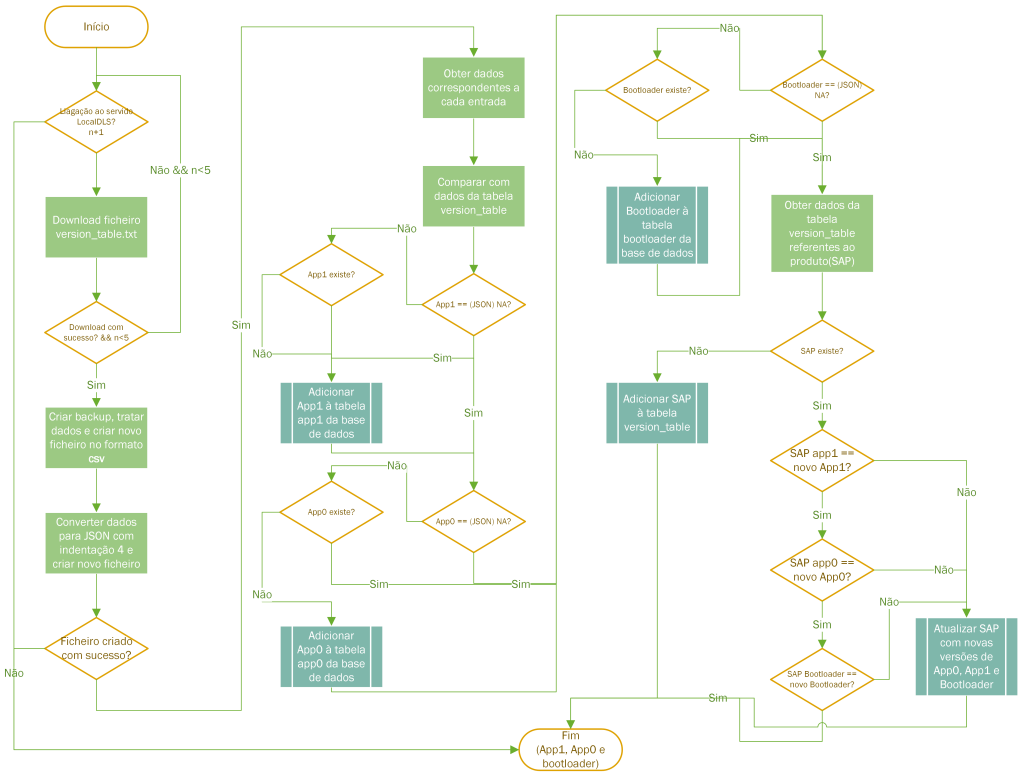


Figura 4.10: Fluxograma detalhado de verificação e atualização do sistema

A Figura 4.9 contém o fluxograma da verificação e atualização das versões de *firmware*. As caixas de texto (1 a 3) explicam a função executada. Primeiro procede-se ao *download* do ficheiro `version_tabel.txt` da localDLS. De seguida, converte-se o ficheiro para `CSV` e, finalmente, para `JSON`, ficando reunidas as condições para aplicar a função `upload_data_version_table` da classe `InitialSystemUpdate`. Esta função compara os dados do ficheiro `version_table.json` com os da tabela `pack_version_table` da base de dados. Caso haja diferenças os respetivos produtos são atualizados com as novas versões. Se o produto não existir na tabela `pack_version_table`, é criado usando os dados do ficheiro. Esta função retorna três listas com as novas versões de cada uma das partes constituintes do *firmware*. Este processo está descrito no fluxograma da Figura 4.10.

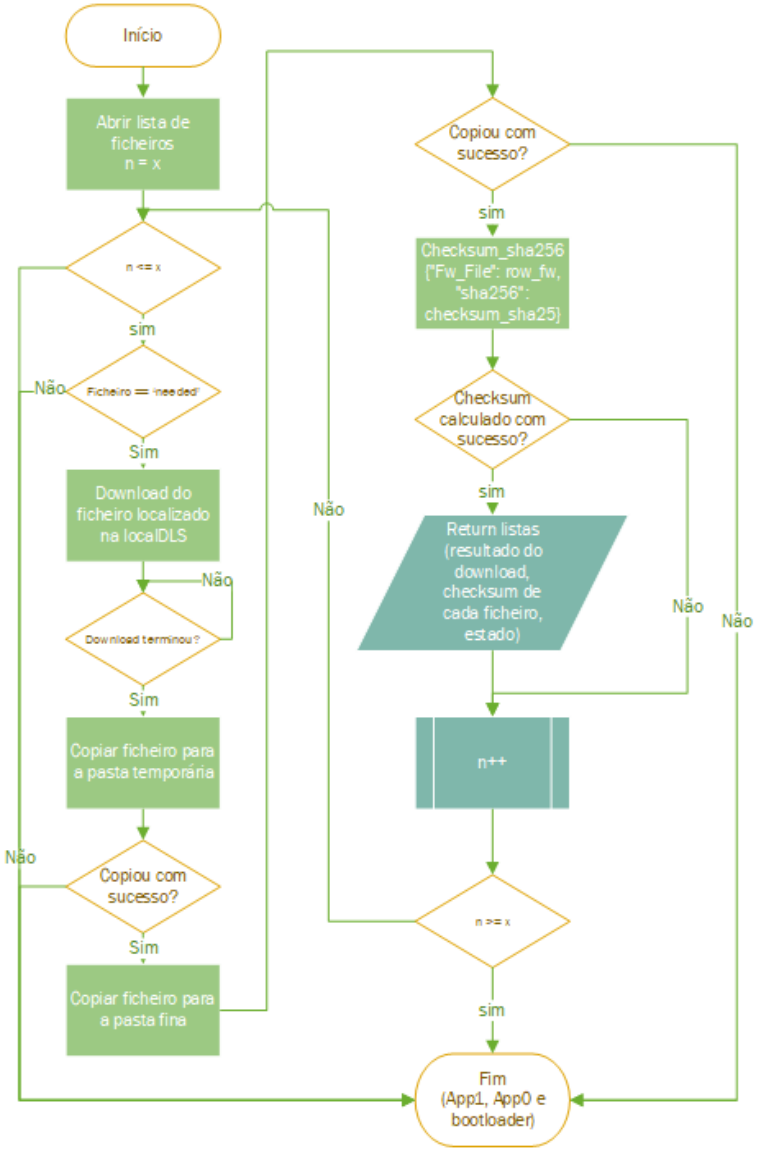


Figura 4.11: Fluxograma detalhado do descarregamento e cálculo do *checksum*

Através destas listas é possível verificar quais os ficheiros de *firmware* que necessitam de ser adicionados ao serviço. A lista, para além das versões novas, contém os nomes dos ficheiros. Uma vez que cada versão é comum a vários produtos, existirão vários nomes de ficheiros repetidos. Para remover as entradas duplicadas, foi criada a função `get_required_download_fw`. A função `download` da classe `FW_Manage` efetua o *download* dos novos ficheiros que estão armazenados na `localDLS`. Esta função recorre à biblioteca `requests`. Esta fornece a função `GET` que, conjuntamente com a criação e escrita de um novo ficheiro, permite copiar da `localDLS` para o novo diretório do sistema

(`\firmware\Plataforma\versão\ficheiro.fw`). Esta função também calcula o `checksum(sha256)` de cada ficheiro. Antes de se proceder ao *donwload* de qualquer ficheiro de *firmware*, verifica-se se já existe no sistema. Desta forma, garante-se que apenas se procede ao *download* de novos ficheiros. No caso de algum evento inesperado ocorrer, é criado um registo na tabela `Ovrp_CS_Log` com o erro, sendo os administradores notificados via email. Tal como nas outras rotas, todas as operações criticas são registadas nesta tabela, incluindo problemas e alterações (Figura 4.11).

Por último, na ausência de erros, o sistema procede à atualização da tabela de produção `fw_in_production`, que é uma réplica da `pack_version_table`. Nesta fase está garantida que a atualização desta tabela irá decorrer sem erros. É feita uma comparação entre cada entrada de ambas as tabelas e caso haja diferenças, a tabela `fw_in_producito` é atualizada com a entrada da tabela `pack_version_tabel`. No caso de um novo produto, este é criado com base nos dados obtidos da tabela `pack_version_tabel`. Após terminar a atualização da tabela, caso haja novas versões ou produtos, é enviado um email para todos os utilizadores com as alterações implementadas no sistema.

Quanto à atualização diária, é executada todos os dias às 00:05. Foi adicionado na `api.py` um `BackgroundScheduler` que verifica todos os dias o ficheiro `version_tabel.txt` da `localDLS`.

4.2.2.9 Rota backoffice/check_access_level

A ligação entre o *back-office* e a base de dados é feita através da API. Foram criadas rotas especificas para a gestão da API como, por exemplo, para obter o nível de acesso de um utilizador, autenticar-se, registar-se, recuperar a *password* e consultar o histórico de atualizações de um produto. A rota `/backoffice/check_access_level` permite verificar o nível de acesso do utilizador. A estrutura deste pedido está representada na Tabela 4.27.

Tabela 4.27: HTTP GET /backoffice_check_user_level

URI	/backoffice/check_access_level/username	
HTTP	GET	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json'	Content-Type = 'application/json'

Body	NA	{
		"user_type": "<str>",
		"admin":<str>
		"access":<str>
		}

4.2.2.10 Rota backoffice/signin

Na rota `/backoffice/signin`, o *back-office* faz o pedido `HTTP POST` à API com os dados do utilizador para autenticação. A API consulta a tabela de utilizadores da base de dados e verifica se o utilizador está registado no sistema e qual o seu nível de acesso. O pedido pode ser verificado na Tabela 4.28.

Tabela 4.28: HTTP POST /backoffice/signin

URI	/backoffice/signin	
HTTP	POST	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json'	Content-Type = 'application/json'

Body	"username":<str>	{
		"user_type": "<str>",
		"admin":<str>
		"access":<str>
		"token":<str>
		"password":<SHA256>
		}

A *password* é validada pelo *back-office*. A *password* é encriptada com recurso ao `SHA256`. Para validar a *password* é necessário utilizar a função `verify` da classe `pbkdf2_sha256`. Este algoritmo ao encriptar uma *password* retorna para o mesmo valor uma *string* encriptada sempre diferente.

4.2.2.11 Rota backoffice/signup

Esta rota permite registar/adicionar um novo utilizador ao serviço. O método implementado é o `HTTP POST` e a informação do novo utilizador é enviada para a API através do *body*. A API verifica se existe algum utilizador com o mesmo email e, em caso negativo, procede ao registo do novo utilizador na base de dados. O pedido `HTTP POST` está representado na Tabela 4.29.

Tabela 4.29: HTTP POST /backoffice/signup

URI	/backoffice/signup	
HTTP	POST	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json'	Content-Type = 'application/json'

Body	"username":<str>	{
		"user_type": "<str>",
		"admin":<str>
		"access":<str>
		"token":<str>
		"password":<SHA256>
		}

4.2.2.12 Rota backoffice/sap_log/

Esta rota foi definida para obter todo o histórico de atualizações de *firmware* de qualquer produto presente na tabela *firmware_in_production*. A estrutura do pedido `HTTP GET` e respetiva resposta podem ser consultadas na Tabela 4.30. Como se pode verificar pelo *body* da resposta, podem ser passadas múltiplas

Tabela 4.30: HTTP GET /backoffice/sap_log

URI	/backoffice/sap_log/<sap>	
HTTP	GET	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json'	Content-Type = 'application/json'
Body	"NA"	[{ "date": "<str>", "App1":<str> "App0":<str> "Bootloader":<str> }, {....}]

respostas JSON, dependendo do número de atualizações que afetam cada produto.

4.2.2.13 Rota backoffice/loadTables

Esta rota foi implementada com a função de obter os dados de todas as tabelas da base de dados, referentes quer aos dados dos produtos quer à gestão do serviço. A especificação do pedido está detalhado na Tabela 4.31. Como se pode verificar, a resposta é uma lista de tamanha 10, em que cada posição da lista contém os dados em formato JSON referentes a cada tabela do serviço. Esta rota é executada sempre que o *back-office* é aberto ou após cada operação executada no mesmo, garantindo desta forma que os dados apresentados aos utilizadores são sempre os mais recentes.

Tabela 4.31: HTTP GET /backoffice/loadTables

URI	/backoffice/loadtables	
HTTP	GET	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json'	Content-Type = 'application/json'

Body	"NA"	{
		"index": <int>,
		"email": <str>,
		"password": <str>,
		"user_type": <str>,
		"access": <str> ,
		"admin": <str>,
		"token": <str>
		},{
		"SAP": <str>,
		"PRODUCT_ID": <str>,
		"VARIANT_ID": <str>,
		"APP1_VERSION": <str>,
		"APP0_VERSION": <str>,
		"BOOTLOADER_VERSION": <str>,
		"APP1": <str>,
		"APP0":<str>,
		"BOOTLOADER": <str>,
		"index": <int>,
		"Manual_Mode": <str>
		},{
		"index": <int>,
		"machine_name": <str>,
		"pc_description": <str>,
		"'line": <str>,
		"token": <str>,
		"machine_id": <str>
		},{
		"idApp1": <int>,
		"App1": <str>,
		"App1Version": <str>,
		"Checksum": <str>,
		"Date": "<date>"
		},{
		"idApp0": <int>,
		"App0": <str>,
		"App0Version": <str>,
		"Checksum": <str>,
		"Date": "<date>"
		},{
		"idBootloader": <int>,
		"Bootloader": <str>,
		"BootloaderVersion": <str>,
		"Checksum": <str>,
		"Date": "<date>"
		},{
		"idUserType": <int>,
		"User_Type": <str>,
		"admin": <str>,
		"access": <str>
		},{
		"CIname": <int>,
		"ID": <int>,
		"Line": <str>,
		"Descritpion": <str>
		},{
		"index": <int>,
		"machine_name": <str>,
		"pc_description": <str>,
		"line": <str>,
		"token": <str>
		]}

4.2.2.14 Rota service/status

A função desta rota é obter os dados referentes a todas as transações feitas quer no *back-office* quer na API, como por exemplo, o processo de verificação automática, em que se faz o registo na tabela `dbOvrpCS_Log` do resultado da verificação. Neste caso a verificação fica registada em nome de `system`. A descrição deste pedido HTTP está especificada na tabela 4.32.

Tabela 4.32: HTTP GET /system/status

URI	/system/status/	
HTTP	GET	
	Pedido	**Resposta**
Header	x-access-token: <token> Content-Type = 'application/json'	Content-Type = 'application/json'

Body	"NA"	{
		"Index": <int>,
		"Date":"<str>"
		"ErrorMessage":"<str>",
		"Status":"<str>",
		"Message":"<str>",
		"User":"<str>"
		}

4.3 Back-office

Nesta secção são abordados todos os aspetos referentes ao desenvolvimento do *back-office*. A imlementação do mesmo foi executada com recurso às linguagens de programação Python, HTML, CSS e JavaScript. Foi decidido utilizar a *micro-framework* Flask, uma vez que a curva de aprendizagem é mais rápida comparando com *framework* Django. A estrutura do projeto pode ser visualizada na Figura 4.12.

Na pasta `templates` são colocadas todas as páginas HTML desenvolvidas, que correspondem ao *front-end* do *back-office*. Para criar um interface mais convidativo e responsivo para o utilizador, utilizou-se o bootstrap v3.4.1, que é um *framework* CSS para ser utilizado no desenvolvimento do *front-end* de interfaces *Web*. O bootstrap utiliza JavaScript e CSS enriquece as páginas com funcionalidades gráficas e funcionais com menus, controlos de paginação, formulários, janelas modais e muito mais. A sua principal caraterística é a responsividade, ou seja, permite que os elementos da página sejam readaptados de modo a correrem em diferentes dispositivos, como portáteis, *tablets*, *smartphones* e computadores de secretária com ecrãs de diferentes tamanhos, adaptando-se sempre ao tamanho e resolução de cada um. Uma vez que o modo de mostrar os dados é através de tabelas, com recurso ao `DataTables`, é possível adicionar interatividade nas tabelas como pesquisa de dados, número de linhas por página e até mesmo exportar as tabelas para ficheiros. O `DataTabels` é um *plugin* da biblioteca `JQuery`, que

permite a integração das funcionalidades apresentadas de uma forma rápida e intuitiva.

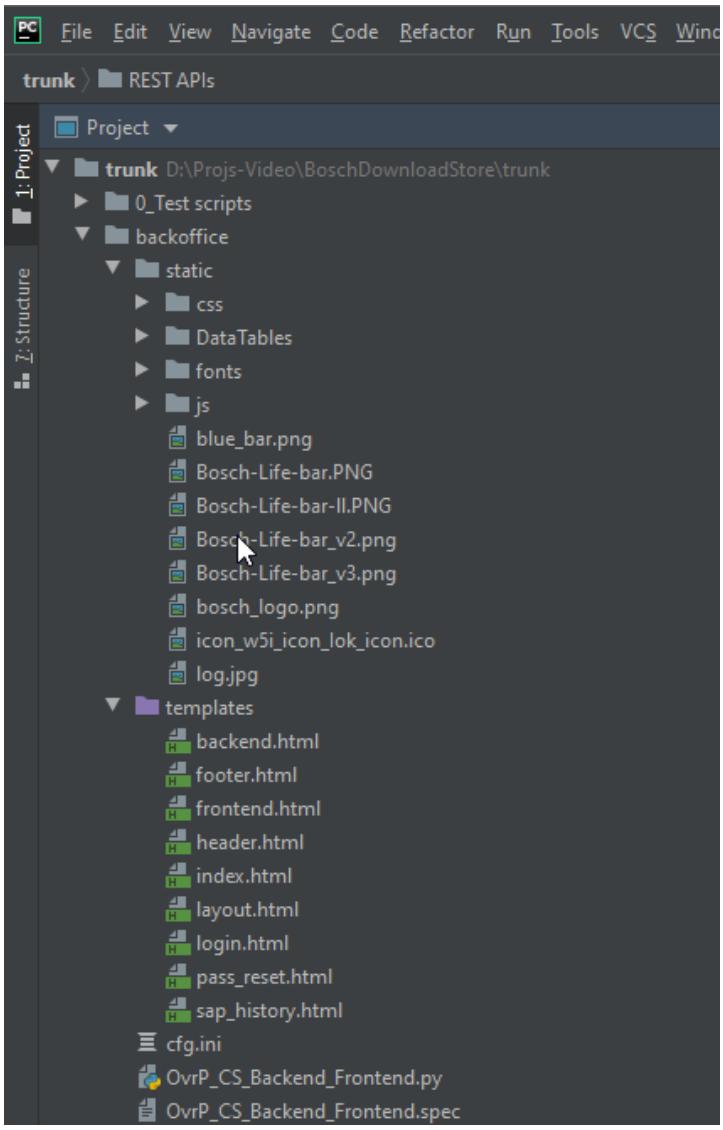


Figura 4.12: Estrutura do Projeto *Back-office*

Para fazer o *render* de todas as páginas `HTML`, uma vez que se utilizou o *flask* como base do *back-office* para desenvolver o *back-end* e o *front-end*, esta biblioteca já tem integrada funções nativas para o fazer nomeadamente a função `render_template`. Para interagir, carregar e apresentar dados de forma dinâmica nas páginas `HTML`, o *Flask* permite passar dados através de variáveis para as páginas `HTML`, sendo que para o fazer, quando se invoca a função `render_template`, esta permite, para além de carregar a página, passar todo o tipo de dados atra-

vés de variáveis definidas por quem está a desenvolver. O excerto de código do Anexo C.1 exemplifica como se procede à passagem de variáveis do *back-end* para o *front-end*. Desta forma, a aplicação *Web* criada para a gestão do sistema, o *back-office*, corre o *back-end* no servidor, que faz a gestão de todos os pedidos do `front-end`, através da invocação das várias funções implementadas. Do lado do *front-end*, para utilizar as variáveis provenientes do *back-end* é necessário chamar da forma {% variável %}. Isto é uma das particularidades do flask. No Anexo C.4 é possível verificar o excerto de código que exemplifica esta abordagem.

Quanto à gestão de utilizadores durante a utilização do *back-office*, a biblioteca `flask` disponibiliza a extensão `session` e o objeto `g`. A extensão `session` é utilizada do lado do servidor e, ao contrário dos *cookies*, os dados que devem ser guardados na sessão são armazenados numa pasta temporária do servidor. A cada utilizador é atribuído um `ID` de sessão e os dados da sessão são armazenados no topo do `cookie`, sendo que os mesmos são encriptados pela aplicação *Web*. A encriptação é feita com base na chave secreta definida na aplicação, através da variável `app.secrete_key`. O objeto `g` é utilizado para ser executado em cada pedido, neste caso, antes de processar cada pedido da aplicação *Web* é utilizado o objeto `g`, no qual é guardado o utilizador, através do `g.user` e é utilizado para garantir que antes de cada pedido, o utilizador está com a sessão aberta, caso contrário o pedido não é processado. O `g.user` é preenchido antes de cada pedido pela função representada no excerto de código do Anexo C.2.

4.3.1 Página login.html

Para entrar no *back-office* o utilizador tem que fazer *sign in* logo ao abrir o URL do *back-office*, em caso de não ter ainda efetuado o *sign in* o navegador *Web* não tem em *cache* as credenciais de acesso, logo a página que abre é a inicial, no caso da *chache* já conter os dados do utilizador este é imediatamente redirecionado para o *back-office*.

Na Figura 4.13 é possível visualizar a página `login.htm`, onde se pode verificar que existe um formulário para preencher *sign in*, um botão para fazer o registo e um botão para recuperar a *password* de acesso. Esta página é carregada através da rota ('/').

4.3.1.1 Sign In

Para fazer `sign in` é necessário preencher o formulário com os dados do utilizador. Ao clicar no botão `Login`, é um pedido `HTTP POST` que é processado pela função `index`. É esta função que é executada quando se abre o URL do *back-office*. Como o *back-office* não tem acesso à base de dados, para validar os utilizadores é feito um pedido à rota `/backoffice/signin` da `API`, que verifica as credencias e o nível de acesso dos utilizadores. O Diagrama de Sequência UML

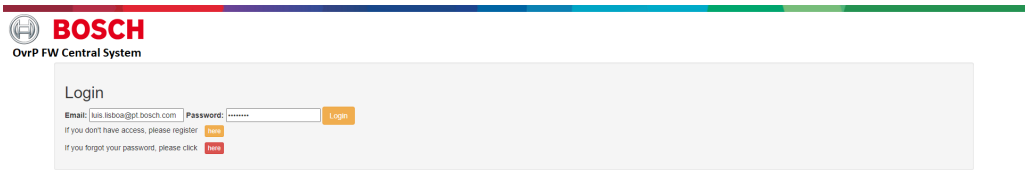


Figura 4.13: Página para fazer *Sign In*

da função index/ está no Anexo C.1. Após o `Sign In`, para garantir que não se perde o utilizador e a respetiva *password*, é utilizado um objeto do `Flask`, `G`, que tem como principal função guardar estes dados durante o tempo que o utilizador estiver com a sessão do *back-office* aberta ou por outras palavras, durante o contexto da aplicação [20].

4.3.1.2 Sign Up

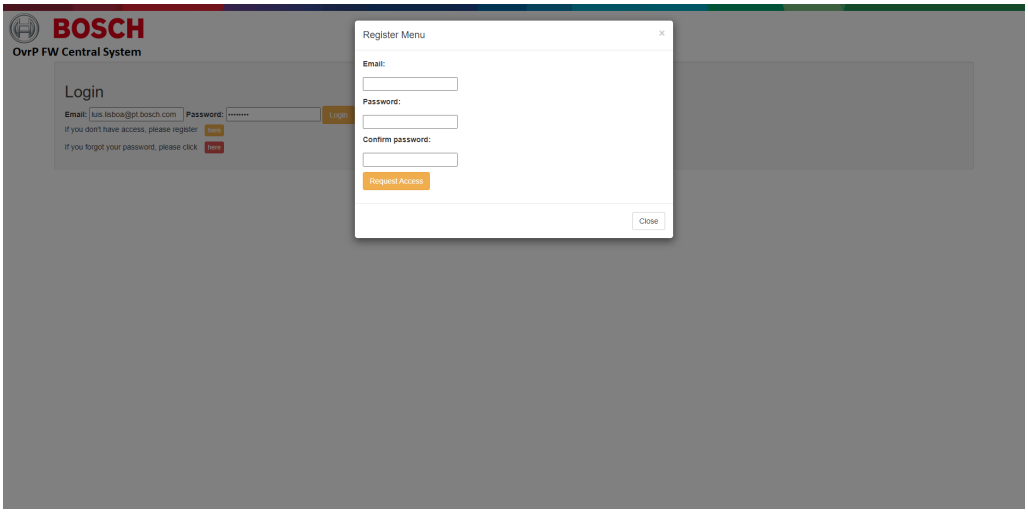


Figura 4.14: Janela Modal para Registar um Novo Utilizador

Para registar um novo utilizador é necessário preencher o formulário que aparece após clicar no botão de registo. O formulário é constituído pelo email, *password* e confirmação da *password*. O formulário pode ser visualizado na Figura 4.14.

Para processar o registo, a aplicação faz um pedido à API através da rota `HTTP POST /.../backoffice/signup` e envia no *body* os dados preenchidos no formulário. No caso de sucesso, é mostrado um `popup` de sucesso na página HTML, no caso de falha, surge um `popup` com a mensagem de erro. A função que faz o pedido à `API` para registar o novo utilizador é a função `register_user` do *back-end*.

4.3.2 Página backoffice.html

O *back-office* está protegido com dois níveis de acesso, sendo que para os `admin` não existe nenhuma limitação a nível de visualização de conteúdos e respetiva gestão. No caso do utilizador ser `viewer`, apenas consegue visualizar a tabela de produtos e respetivas versões de *firmware*. Existe um terceiro nível de utilizador, o `notify`, que não tem acesso à `API` nem ao *back-office*, sendo que apenas recebe notificações via email quando são introduzidos novos produtos ou versões de *firmware* através do processo de atualização automático. A página backoffice.html, com o nível de acesso de administrador está representada na Figura 4.15.

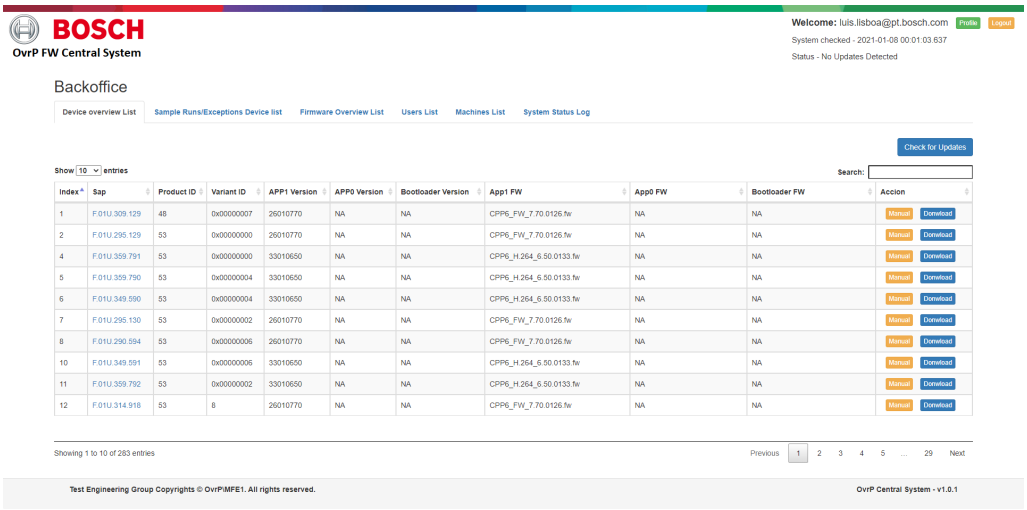


Figura 4.15: Página principal do *Backoffice* em modo `Admin`

Quando se faz *login* antes de carregar a página são processados todos os dados necessários incluir na mesma, para isso é executada a função `backoffice()`. Nesta função antes de carregar a página do *back-office*, verifica o nível de acesso do

utilizador através da rota `HTTP GET../backoffice/check_access_level/user`. O passo seguinte é obter os dados necessários para preencher todas as tabelas dos vários menus do *back-office* através da rota `HTTP GET backoffice/loadTables`. Esta rota devolve uma lista com todos os dados exceto os dados do registo (*log*) das várias operações, que é processado pela rota `HTTP GET /service/status`. Caso o utilizador seja administrador, quando entra no *back-office* consegue visualizar todos os menus, caso contrário apenas pode consultar os dados de produção referentes às versões de *firmware* de cada produto. Este processo está descrito em detalhe no Diagrama de Sequência UML no Anexo C.2.

4.3.2.1 Menu Device Overview List

Neste menu é possível consultar todos os produtos que estão a funcionar em modo automático, ou seja, sempre que existe alguma atualização automática, estes são os produtos afetados pela mesma. Na coluna `Action`, estão disponíveis dois botões, `Manual` e `Downlaod`. O botão manual permite transferir um produto para a lista de exceções, passando assim a ser gerido manualmente pela engenharia da fábrica de Ovar. Apenas em situações especiais se adiciona um produto à lista de exceções. Também nesta tabulação se pode forçar uma atualização do sistema, para isso é necessário clicar no botão `Check for updates`. Este botão vai fazer um pedido `HTTP PUT` à API através da rota `.../pack_version_table`. O resultado de qualquer operação é sempre visualizado através de um *flash popup* no topo centro da página `html`. O Diagrama de Sequência UML do pedido de atualização forçado está no Anexo C.3.

4.3.2.2 Menu Sample Runs/Exceptions Device List

Este menu foi um requisito definido para o projeto devido à alta diversidade de produtos das várias famílias de câmaras de videovigilância, em que podem existir certas situações que requerem que determinado produto não seja incluído nas atualizações de *firmware*, sendo que uma das razões pode ser as concessões de produção, onde normalmente os requisitos são que durante uma produção controlada a versão de *hardware* e *firmware* não se alterem. Esta opção também é utilizada durante a industrialização de produtos, em que, como já foi referido, durante a industrialização os produtos não constam na lista do ficheiro `version_tabel.txt`. Neste menu, a tabela apresentada com os dados dos produtos é igual à do menu `Device overview List`, sendo que na coluna das ações existem três botões em vez de dois. O botão `Edit`, `Download` e `Delete`. Existe também um botão para adicionar manualmente um novo produto. Através do botão `Edit` é possível alterar todos os dados de um produto. Na Figura 4.16 é possível consultar este menu.

A Figura 4.17 apresenta o menu de alteração de dados de um produto. Ao

editar um produto, quando se clica no botão `Update`o formulário preenchido é enviado através do método `HTTP POST backoffice/edit_SAP_exception` para o *back-end* e é processado pela função `edit_SAP_exception`. Esta função pega nos dados do formulário e faz um pedido `HTTP PUT` à rota da API `.../device/<device_id>`, em que no *body* são enviados todos os dados do produto. Este processo está descrito no Diagrama de Sequência UML do Anexo C.4.

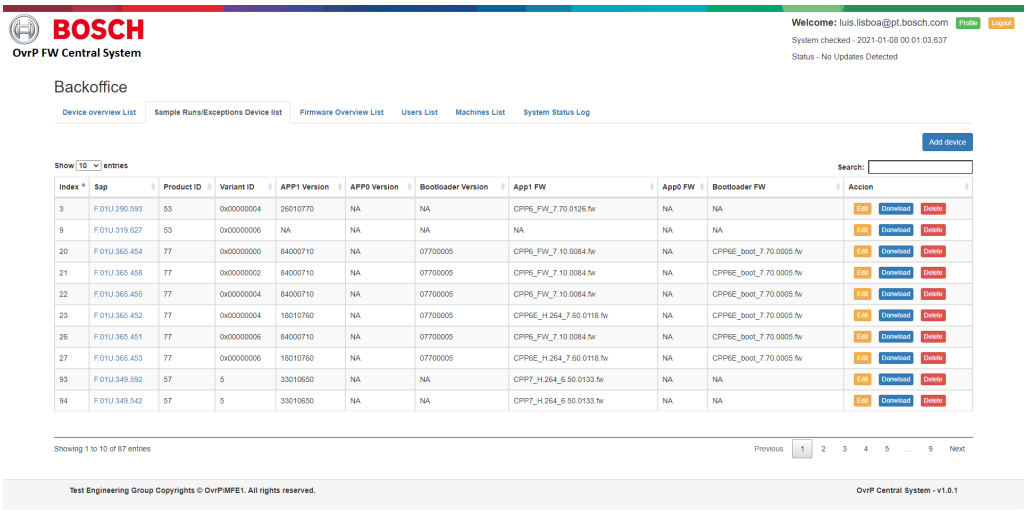


Figura 4.16: Menu `Sample Runs/Exceptions Device List`

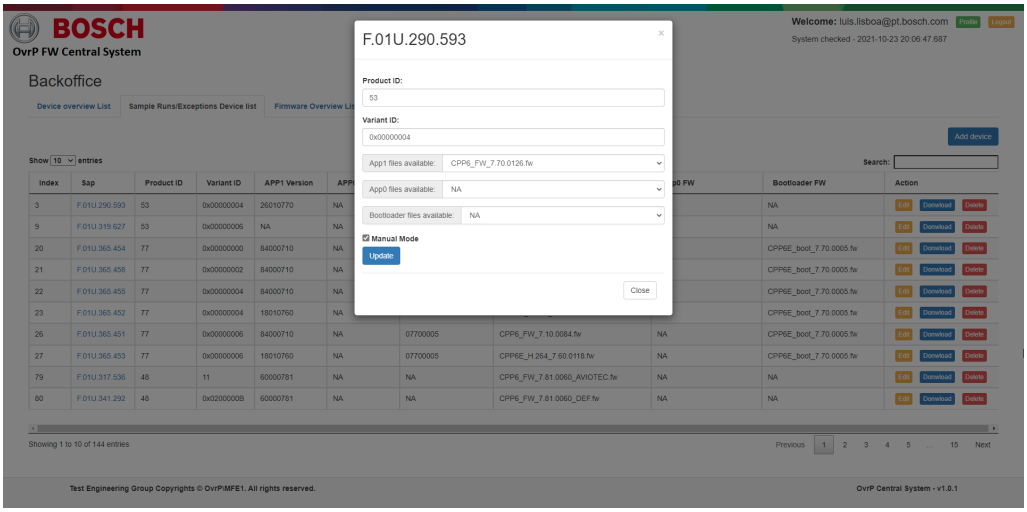


Figura 4.17: Página Modal para editar `Sample Runs/Exceptions Device List`

Para adicionar um produto a esta lista de exceções a lógica é a mesma, sendo

que é preenchido um formulário com todos os dados e enviado para o `back-end` através de o `HTTP POST .../add_device_manually`, que por sua vez, faz com que seja executado no *back-end* a função `add_device_manually`. Esta função faz um pedido `HTTP POST` à rota da API `.../device`, em que os dados são enviados no *body* com o formato `JSON`. No Anexo C.5 está ilustrado o Diagrama de Sequência para adicionar um `device`. O menu de adicionar um dispositivo à lista de exceções está representado na Figura 4.18.

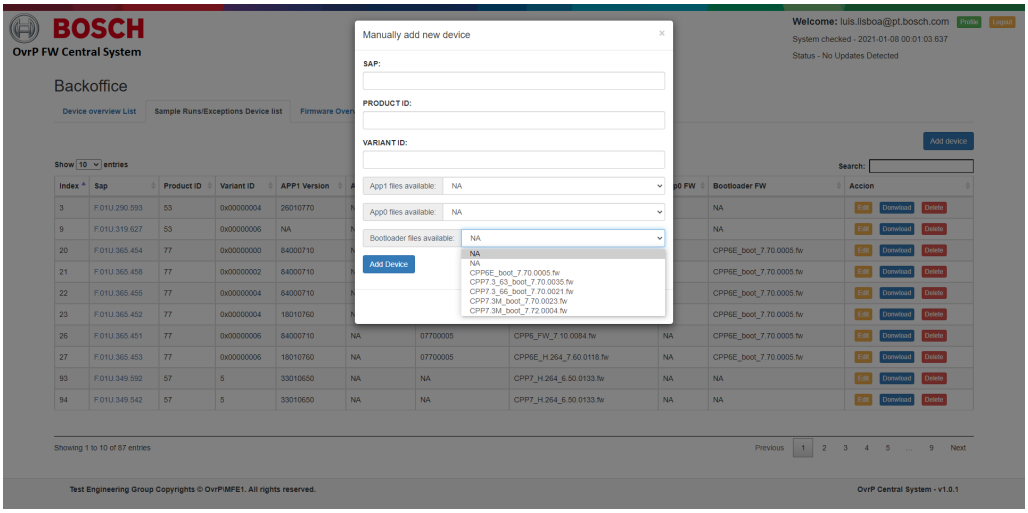


Figura 4.18: Menu `Sample Runs/Exceptions Device List` Add Device

4.3.2.3 Menu Firmware Overview List

Este menu está dividido em três sub menus referentes a cada parte constituinte do *firmware*, ou seja, `App1`, `App0` e `Bootloader`. Em cada um dos sub menus é possível consultar todos os dados referentes a cada versão e a data em que foi adicionada no sistema. Como existe a necessidade de adicionar produtos manualmente, também existe a necessidade de adicionar as partes do *firmware* manualmente, permitindo assim aos administradores adicionar versões de *firmware* que só são utilizadas durante os *sample runs*. Para adicionar novas versões, em cada sub menu existe um botão, como por exemplo, `Add App1`. Para adicionar manualmente um ficheiro é necessário que o nome do mesmo cumpra com o formato que é utilizado no ficheiro `version_tabletct`, ou seja, o nome do *firmware* tem respeitar o formato "plataforma_FW_versão.fw". Ao clicar no botão `Add App1`, vai aparecer um menu modal como se pode verificar na Figura 4.19.

Neste menu apenas aparece a opção de selecionar um ficheiro. Ao clicar `Add App1`, este vai carregar o ficheiro para o sistema e com base no nome vai guardar o mesmo no diretório correto, ou seja, o caminho do diretório será `.../Firmwa-`

`re/App1/plataforma/versão/`. O mesmo se aplica para o `App0` e `Bootloader`. Após adicionar um ficheiro de *firmware*, será apresentado o resultado através da função `flash`. Este processo está detalhado no Diagrama de Sequência UML no Anexo C.6, onde como exemplo é adicionado um ficheiro `App1`.

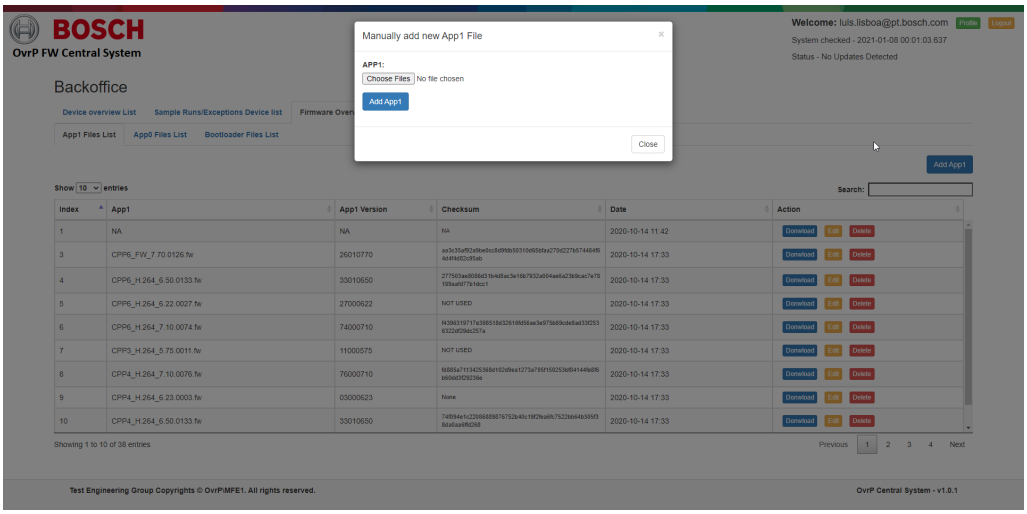


Figura 4.19: Janela Modal para Adicionar `Firmware/App1`

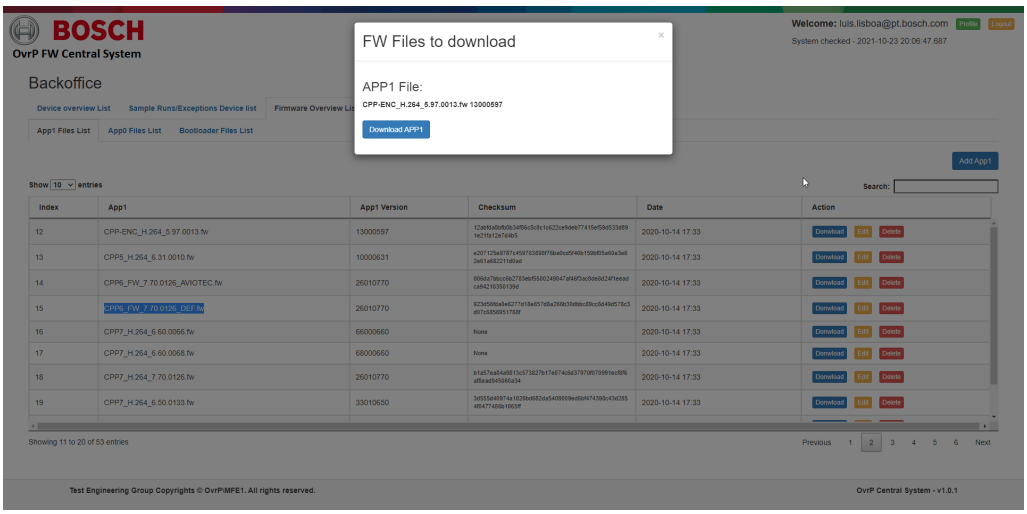


Figura 4.20: Janela Modal para Remover `Firmware/App1`

No caso do `App0` e `Bootloader`, a diferença na sequência seriam as rotas dos pedidos HTTP, sendo que o restante se mantém, não existindo desta forma a necessidade de duplicar Diagramas de Sequência UML. Foi também adicionada a

opção de editar, apagar e fazer o *download*. Cada versão de *firmware* tem estes botões na coluna `Action`. Como o nome indica, o botão `Download` permite ao administrador descarregar o ficheiro para o computador. O botão `Delete` permite ao administrador apagar uma entrada da tabela e o botão `Edit` permite alterar o ficheiro de *firmware*.

Para editar uma entrada, o processo é praticamente o mesmo que é executado na Figura 4.19, sendo que as diferenças residem na rota do pedido do *back-office*, que neste caso é `HTTP POST .../backoffice/update_app1` e o pedido à API, que neste caso seria `HTTP PUT app1/<idApp1>` em que no `body` é enviado o nome do novo ficheiro. Este Diagrama de Sequência UML pode ser consultado no Anexo C.7.

Para descarregar um ficheiro de *firmware* basta clicar no botão `Donwload`. Este vai abrir um menu/janela modal onde se confirma o ficheiro e se clica no botão `Download App1` para iniciar o descarregamento do *firmware*. Esta janela modal pode ser analisada na Figura 4.20. O Diagrama de Sequência UML da eliminação de um ficheiro encontra-se no Anexo C.8.

O processo é igual para cada um dos ficheiros `App1`, `App0` e `Bootloader` do *firmware*, havendo apenas diferença nas rotas da API e do *back-office*.

4.3.2.4 Menu Users List

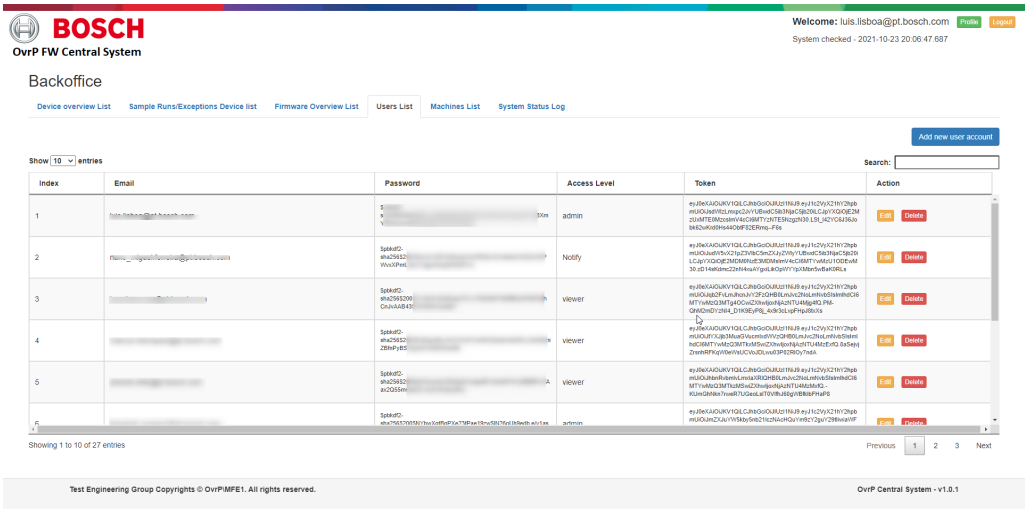


Figura 4.21: Menu `Users List`

Através do menu `Users List` o administrador consegue gerir todos os utilizadores do serviço. No canto superior direito da tabela, existe o botão `Add new user account`, que permite ao administrador adicionar novos utilizadores. Na tabela

onde se podem consultar os dados da tabela da base de dados `users`, estão disponíveis todos os utilizadores adicionados, em que para cada entrada, na coluna `Accion`, estão disponíveis dois botões, o `Edit` que permite editar e o `Delete` que permite apagar uma entrada. Esta informação pode ser constatada na Figura 4.21.

Quando o administrador adiciona um novo utilizador, ao clicar no botão `Add new user account`, uma janela modal abre com os dados para serem preenchidos, sendo estes, o email e o nível de acesso que se pretende atribuir ao novo utilizador. A janela modal pode ser visualizada na Figura 4.22.

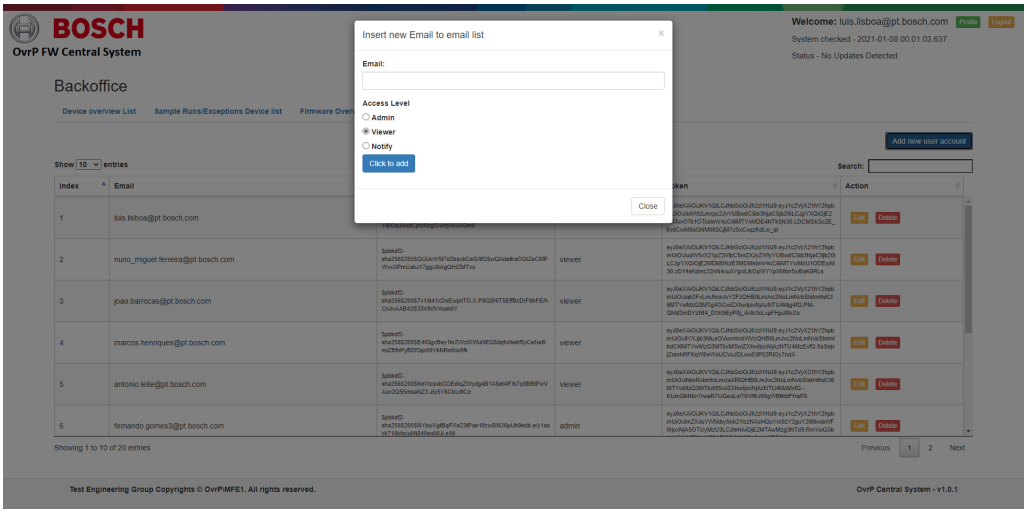


Figura 4.22: Janela Modal para Adicionar novo Utilizador

O diagrama de sequência UML do pedido `HTTP POST .../user` está representado no Anexo C.9, onde se pode verificar de uma forma mais detalhada o pedido para adicionar um novo utilizador. O *back-end* faz o pedido `HTTP POST` à API e no *body* envia os dados em formato JSON. Dependendo do resultado, em caso de sucesso, ou seja, `['status']` igual a `pass`, envia um email com o *link* e `password` provisório ao novo utilizador. O email está em formatado em HTML e está representado no excerto de código no Anexo C.3. Como já referido, o administrador tem a permissão de editar os dados, nomeadamente, alterar o email, atribuir uma nova *password*, alterar o *token* e alterar o nível de acesso ao *back-office*. Quando o administrador clica no botão `Edit`, abre uma janela modal com todos estes parâmetros. Após alteração, ao clicar no botão `Update`, os dados do formulário são passados para o *back-end*, para a função `update_user`. Esta função lê os dados do formulário e faz o pedido `HTTP PUT` à API. Após resultado, se positivo é exibida a mensagem de sucesso, caso contrário falha em caso de

algum erro ou situação inesperada. O Diagrama de Sequência UML deste pedido está referenciado no Anexo C.10.

Para apagar um utilizador do serviço, é necessário clicar no botão `Delete`, que por sua vez lança uma notificação através de uma janela modal, para confirmar. Ao confirmar, a aplicação *Web* do *back-office* faz um pedido `HTTP DELETE` à API. O Diagrama de Sequência UML para apagar um utilizador está descrito no Anexo C.11.

4.3.2.5 Menu Machines List

Neste menu foram criados dois sub menus com o intuito de dividir as máquinas em dois grupos, as geridas pelo departamento de *Information and Thecnology for Manufacture* (ITM) de Ovar e e as que não são geridas por este departamento. Como o próprio nome indica, no menu `OvrP ITM Machines` estão as máquinas da produção e geridas pelo ITM, e as restantes no menu `Non OvrP ITM Machines`. Quanto ao primeiro menu, foram adicionadas duas funções, a de adicionar e de remover. Ao clicar no botão `Add ITM Machine`, abre uma janela modal, que pode ser visualizada na Figura 4.23, com uma listagem completa de todas as máquinas existentes na fábrica de Ovar.

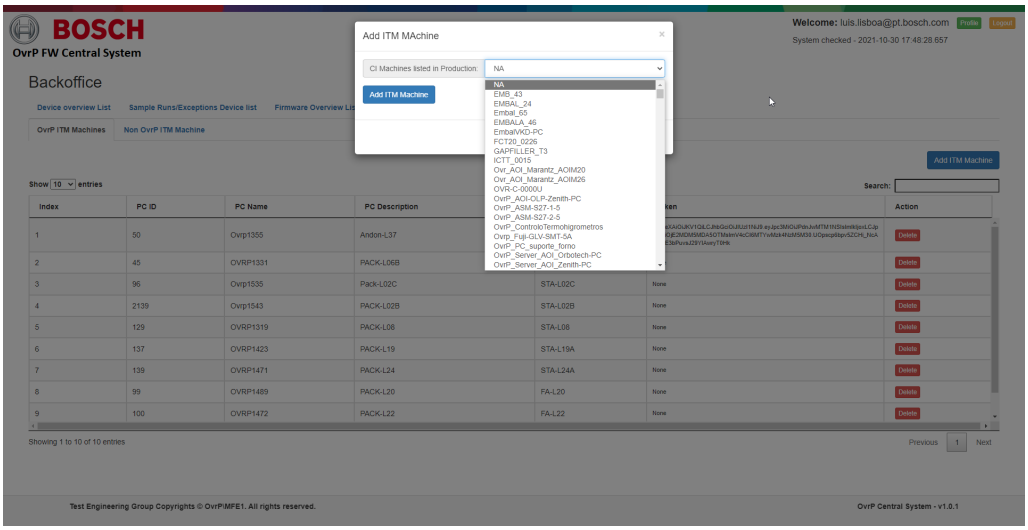


Figura 4.23: Janela Modal para Adicionar Máquina ITM

O administrador seleciona uma das máquinas e clica no botão `Add ITM Machine`. Este botão vai enviar para a função `addnewMachineITM` do *back-end* a máquina selecionada que por sua vez será enviada para a API através do pedido `HTTP POST /machine`. Este processo está detalhado no Diagrama de Sequência UML no Anexo C.12.

Para apagar uma máquina do serviço, ao clicar no botão `Delete`, é aberta uma janela modal a confirmar a operação. Nesta janela é mostrado o nome do PC para validar que é realmente o selecionado que se pretende apagar. Ao clicar no botão `Delete`, é feito um pedido à API através da função da função `delete_machine_from_machine_list()`. Esta função faz o pedido `HTTP DELETE` ao recurso `machine/machione_table/machine_id` da API. O Diagrama de Sequência UML deste pedido está no Anexo C.13. A função que processa o pedido de apagar é a mesma para os dois tipos de máquinas existentes, sendo que é o campo `machine_table` que faz a distinção entre as duas quando é feito o pedido à API.

Quando se pretende adicionar para as máquinas que não são geridas pelo `ITM`, é necessário preencher as informações manualmente. No segundo menu do *backoffice*, é possível adicionar uma máquina nova, para isso é necessário clicar no botão `Add Pack/FW Upload Machine`. Este botão vai abrir uma janela modal que contém o formulário da Figura 4.24

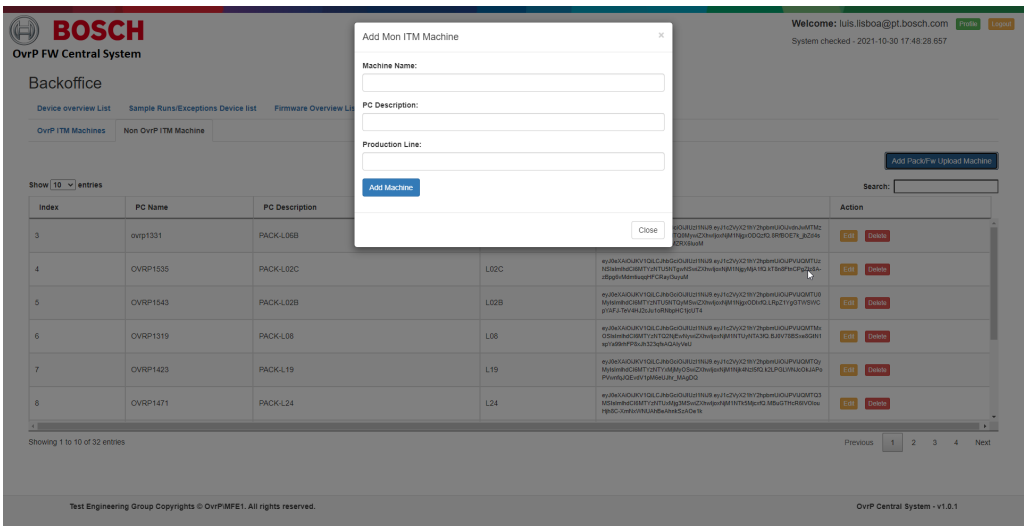


Figura 4.24: Janela Modal para Adicionar Máquina não ITM

Este formulário contém três campos, o nome, a descrição e a linha de produção. Com estes parâmetros é possível identificar de forma clara uma máquina. Após preenchimento do formulário é enviado o mesmo para a função `addnewMachine` do *back-end*. Esta função apropria-se dos dados submetidos no formulário e envia-os para a API através da rota `HTTP POST .../machine`. No corpo da mensagem são enviados os dados em formato JSON. Como a rota é a mesma para ambas as situações, a API faz a distinção através do campo `CIname`, que para este pedido vai com o valor `'None'`. O Diagrama de Sequência UML está no Anexo C.14

Para as máquinas que não são geridas pelo `ITM` é possível atualizar os dados caso seja necessário, como a descrição ou a linha em que a máquina esta inserida. Ao clicar no botão `Edit` de uma dada linha da tabela do *backoffice*, é aberta uma janela modal com o formulário preenchido com os dados da máquina em questão. Após alteração do formulário, ao submeter o mesmo através do botão `Update`, como se pode verificar pela Figura 4.25. Ao clicar no botão, através do `HTTP POST /backoffice/update_machine` é executada a função `update_machine` do *back-end*. Esta função faz um pedido `HTTP PUT .../machine` à API. O Diagrama de Sequência UML deste pedido está no Anexo C.15.

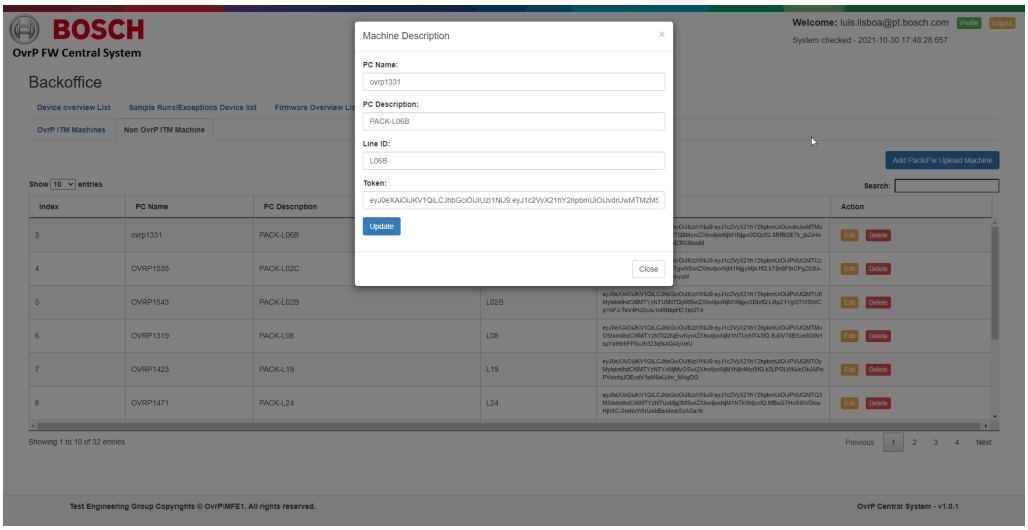


Figura 4.25: Janela Modal para Editar Máquina não ITM

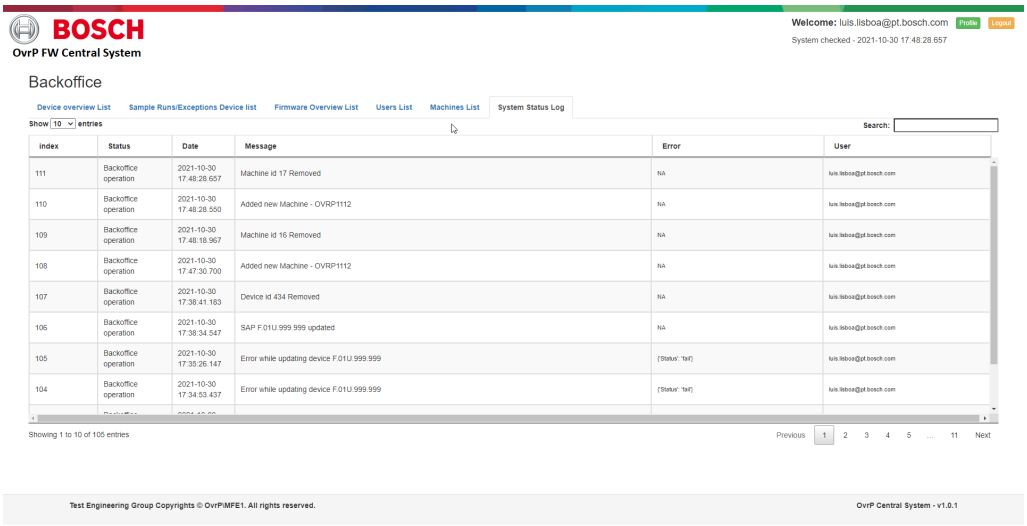


Figura 4.26: Menu System Status Log

4.3.2.6 Menu System Status Log

Neste menu, pode consultar-se as ações/pedidos processados pela aplicação *Web* do *back-office*. Este menu está disponível na Figura 4.26.

Todos as transações efetuadas no *back-office*, no processo de atualização automático e qualquer tipo de erro que aconteça é gravado na tabela `Ovrp_Cs_Log` e pode ser consultado neste menu.

4.3.2.7 Histórico de Atualizações do Produto

Para cada produto, é possível consultar o seu histórico de atualizações e para isso basta clicar no `SAP`, pois está definido no *front-end* como um *link*. Este link redireciona o utilizador para outra página, sendo o link definido para obter o histórico de cada um, `http://backoffice:port/<sap>`. Na Figura 4.27 está representada uma página com o histórico de atualizações de um produto.

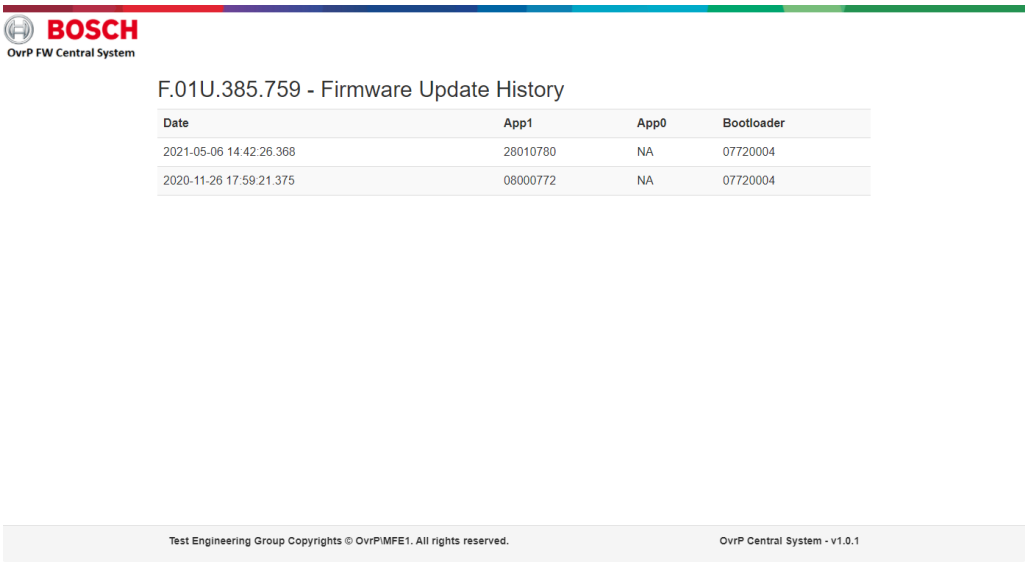


Figura 4.27: Histórico de Atualizações de um Produto

Os dados apresentados contêm a data e a respetiva versão de cada parte do *firmware*. Estes dados são obtidos através da função `history` do *back-end*. Esta função faz o pedido à `API HTTP GET .../backoffice/sap_log`, que por sua vez devolve apenas os dados da coluna `history` da tabela da base de dados `fw_in_production`. Os dados estão guardados no formato JSON, e facilmente são convertidos para uma lista. Esta lista é percorrida e preenchida no *front-end*.

4.4 Carregamento de Firmware

Nesta secção apresenta-se a solução desenvolvida em `Labview`, esta executa todo o procedimento de carregamento de *firmware* para as câmaras após serem validadas no processo de manufatura. O carregamento de *firmware* pode ser executado em dois postos distintos, no posto de embalagem e na estação de carregamento de *firmware*. Como já foi referido, a estação de carregamento de *firmware* permite carregar em simultâneo até quatro câmaras. A necessidade da criação desta estação prende-se ao facto de que apenas é possível ligar uma câmara ao posto de embalagem, o que fez com que este posto se tornasse o *bottleneck* em algumas linhas de produção. De seguida será explicado o desenvolvimento de ambos.

4.4.1 Posto de Embalagem

O posto de embalagem realizava o carregamento de *firmware* através de um *software* fornecido pelo R&D. Esta solução era baseada em *software* de terceiros que executava via linha de comandos. O principal objetivo foi criar uma solução de

carregamento que permitisse a comunicação com a API desenvolvida, automatizando o processo de atualização e carregamento. Como o posto de embalagem está desenvolvido em LabVIEW 2016, o *software* de carregamento automático também foi desenvolvido na mesma linguagem, permitindo a integração direta no código fonte. Uma vez que o posto de embalagem já se encontrava desenvolvido, a integração do VI de carregamento de *firmware* foi feita pelo responsável do posto de embalagem. Nesta secção é detalhado o desenvolvimento do VI de carregamento de *firmware*.

4.4.2 VI de Comunicação com a API

Para que o VI de carregamento de *firmware* comunique com a API, foi desenvolvido um subVI responsável por toda a comunicação. O GUI deste VI está representado na Figura 4.28.

Para obter um *token* válido, é necessário que o VI cumpra os requisitos da API, nomeadamente da rota `HTTP POST .../token`. Para obter o *token* é necessário que, neste caso, a máquina tenha sido adicionada ao serviço. Quanto aos parâmetros do *header*, é necessário enviar os seguintes:

- content-type: application/json;
- x-access-key : "secret key- Encpritado na base 64.

No *body* do pedido é enviado em formato JSON, a chave `machine_name` e respetivo valor. Neste VI, o *header* está definido no código fonte. Após obter um *token*, todos os pedidos que o VI faz à API contêm o *token* no campo `x-access-token` do `header`, uma vez que este é mandatário como já descrito no desenvolvimento da API. Como o *token* tem validade de 24 h, antes de ser feito o pedido de um novo, é verificado se o que está em memória ainda está válido. Desta forma reduz-se o número de pedidos à API, reduzindo diretamente o tráfego.

Quando o VI recebe uma resposta de "`Invalid token`", faz um novo pedido. Para a gestão do *token*, foi criada um controlo com os seguintes campos: `Token`, `Time Left`, `Requested at` e `Renew at`.

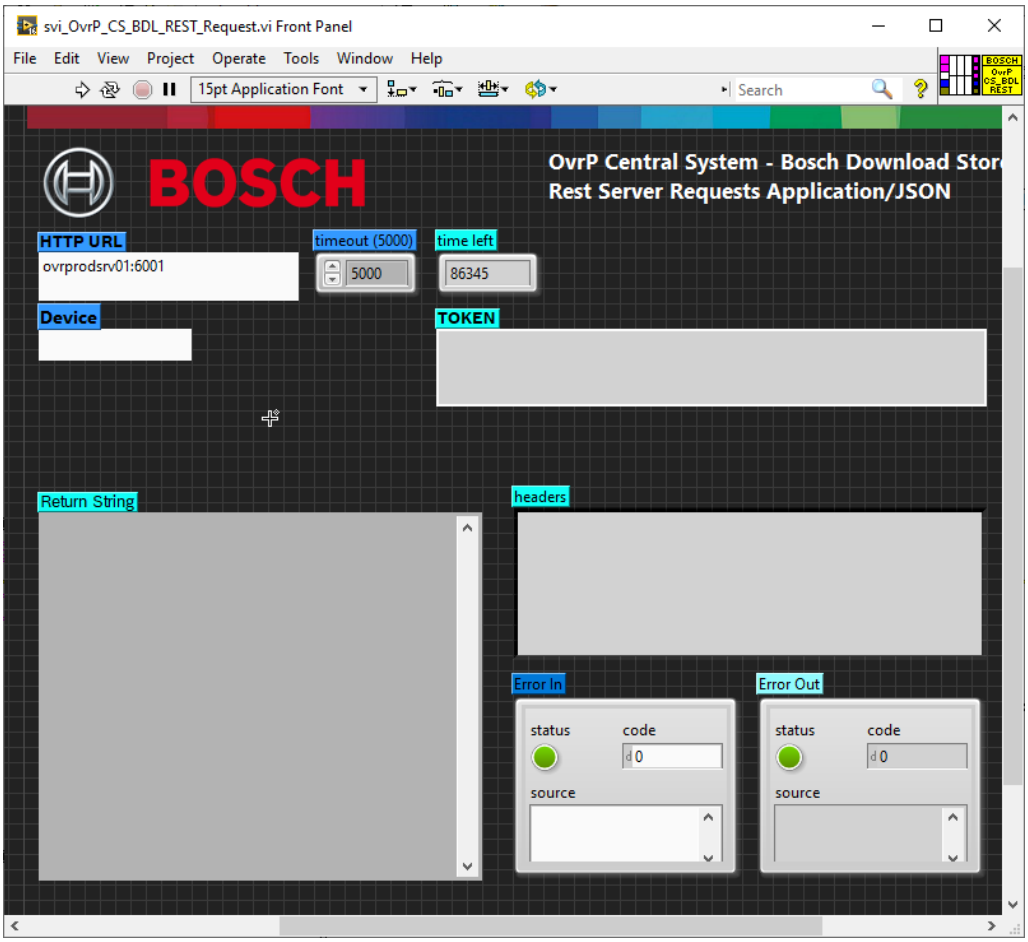


Figura 4.28: subVI que executa pedidos HTTP

Na Figura 4.29 está representado o fluxograma deste VI. Quando o *token* é gerado todos os campos do controlo são atualizados. Nos pedidos seguintes, se o *token* ainda for válido, o VI envia o pedido com o *token* em memória, evitando fazer um pedido de um *token* em cada pedido HTTP. No caso de pedir um sem ser necessário, a API iria retornar sempre o mesmo *token* com um campo extra onde consta por quanto tempo ainda é válido.

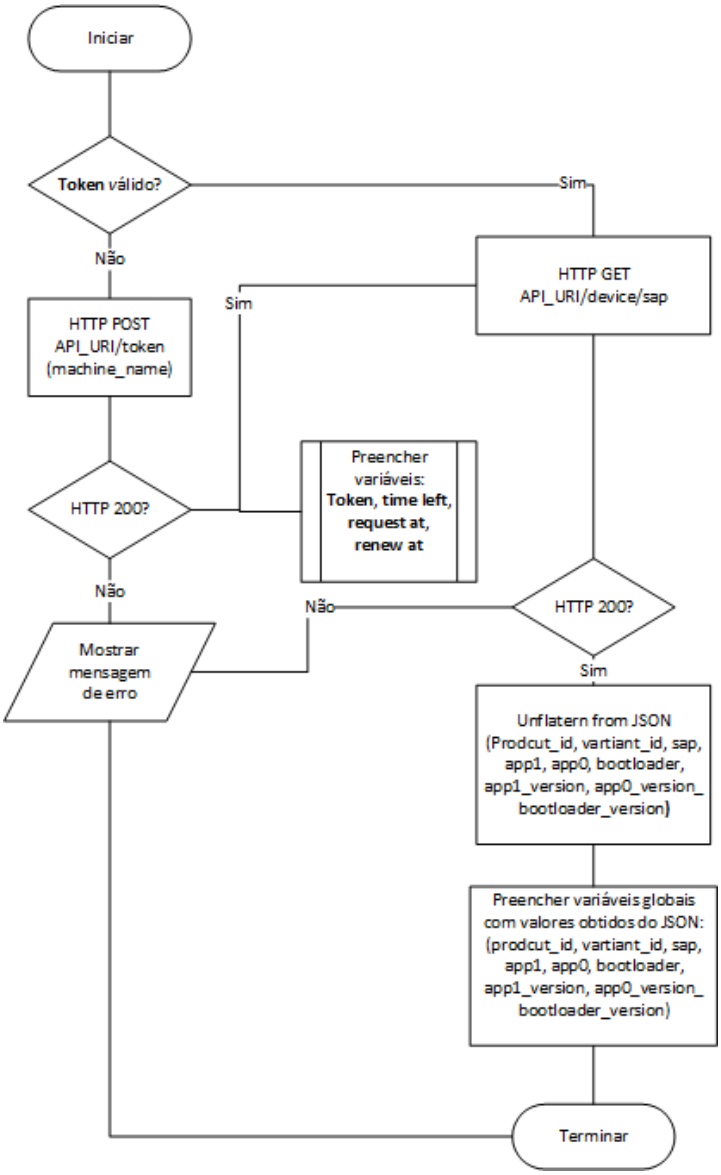


Figura 4.29: Fluxograma de um Pedido HTTP

Para validar se o pedido HTTP do *token* foi executado com sucesso, o VI desenvolvido verifica qual o HTTP status presente no *header* da resposta. Caso o status seja 20X, significa que o pedido foi processado com sucesso. No caso de um status 40X, significa que ao pedido em questão não pode ser processado, sendo que o status segue os padrões do protocolo HTTP. Após ter obtido um *token* válido, o estado seguinte é fazer um pedido `HTTP GET` à rota `.../device/<sap>` para obter os dados de produção de um determinado produto. Se o pedido retornar uma resposta sem erro, é preenchida a variável global `GetDevice_API` com os

dados obtidos. Em caso de erro, o VI retorna uma mensagem para alertar que o pedido não foi executado com sucesso.

4.4.3 VI de Obtenção dos Dados da Câmara

Para obter todos os dados necessários de uma câmara, nomeadamente todas as versões de cada componente instalado, foi desenvolvido um VI (`svi_BoschST-CPPx-GetDUTVersion.vi`).

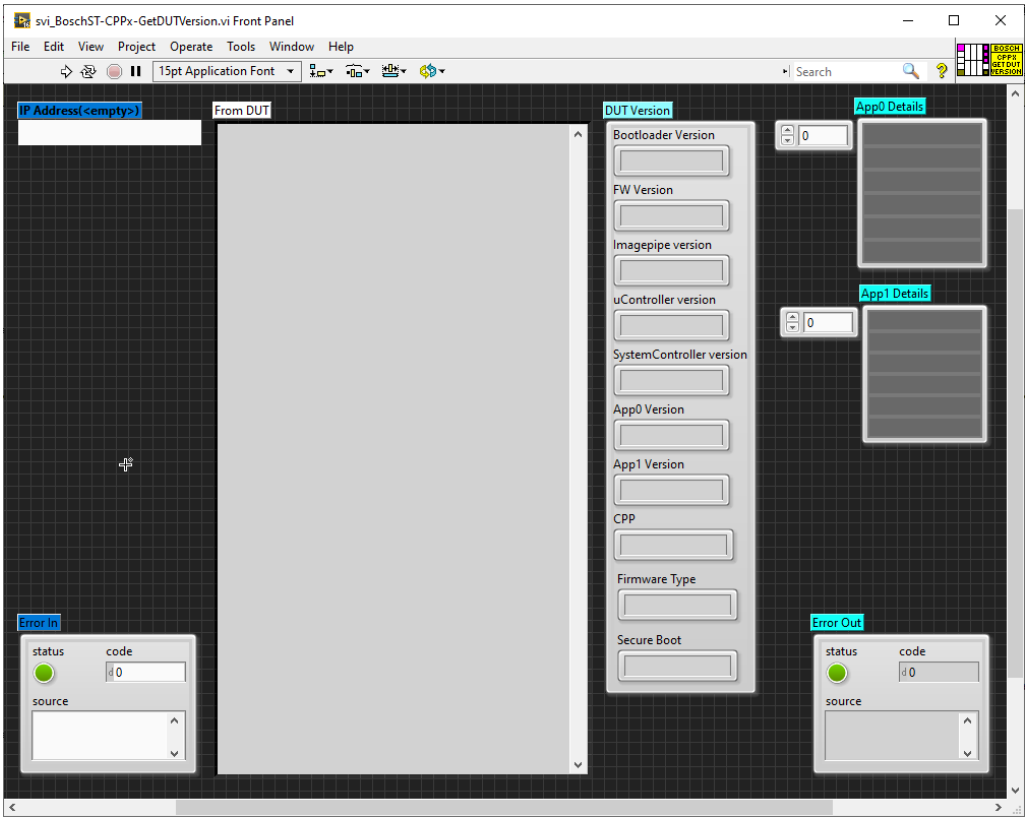


Figura 4.30: subVI - Guardar dados variável global

A interface do VI está visível na Figura 4.30. Como controlos, apenas é necessário o IP da câmara e como indicadores, é possível visualizar todas as mensagens no indicador `From DUT`. O cluster `DUT Version` e os *arrays* `App0 Details` e `App1 Details` contêm a informação de cada indicador depois de ser executado o *parse* aos dados obtidos e guardados no indicador `From DUT`.

Como já referido anteriormente, no caso da plataforma Ambarella, pode ser necessário atualizar o `Bootloader`, `App0` e `App1`. A versão das partes constituintes do *firmware* é obtida através do endereço "`http:\\<ip>\version`", e esta mensagem é guardada no indicador `From DUT`. As variáveis utilizadas para guardar os

dados necessários para se proceder à atualização desta plataforma são `Bootloader Version`, `FW Version`, `Imagepipe Version`, `Controller Version`, `System Controller Version`, `CPP`, `App1 Version` e `App0 Version`.

No caso da plataforma Android, o *firmware* não está dividido em três partes, isto é, não é possível atualizar separadamente partes constituintes do *firmware*, sendo que apenas existe um ficheiro de *firmware* para se proceder ao carregamento/atualização. O indicador que é utilizado para guardar a versão de *firmware* do produto é o `FW Version`. Os indicadores com relevância para esta plataforma são `FW Version`, `Firmware Type` e `Secure Boot`.

Como já referido, o `FW Version` é a versão de *firmware*, o `Firmware Type` corresponde ao tipo de *firmware* e o `Secure Boot` é o indicador onde é guardado o estado de uma *flag* do *firmware*. Esta plataforma tem duas particularidades, a primeira é referente ao tipo de *firmware*, pois pode ser uma versão *debug* ou *release*. Durante a produção das câmaras é necessário utilizar a versão de *debug*, pois só com esta versão é possível carregar os certificados necessários para que as câmaras ativem algumas funcionalidades, após instalar todos os certificados é carregado duas vezes a versão *release* de *firmware*, pois no primeiro carregamento o *firmware* é instalado na primeira partição da câmara, e no segundo carregamento é instalado na segunda partição. Este comportamento foi previsto e implementado por forma a distinguir-se se aquando do carregamento, o programa de teste está na presença de uma câmara da plataforma *Ambarella* ou *Android*.

4.4.4 VI de Carregamento de Firmware

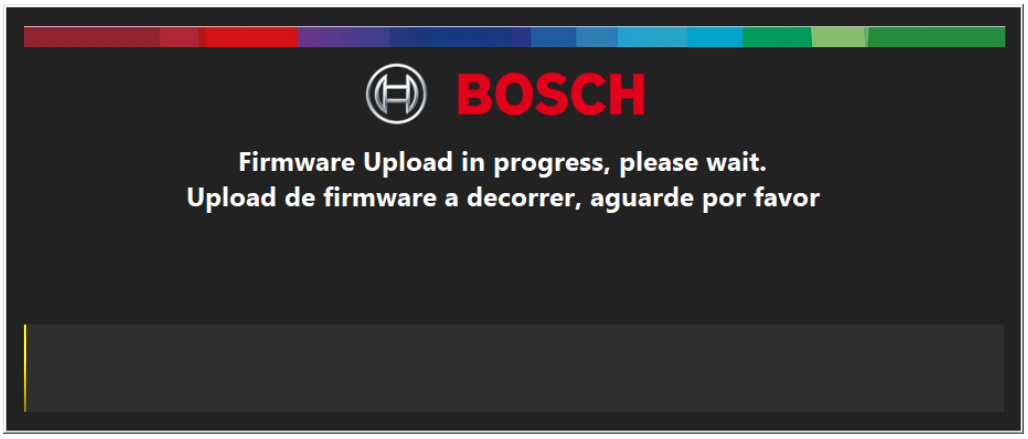


Figura 4.31: VI de carregamento de *firmware*

Na Figura 4.31 está representada a interface de utilizador do VI de carregamento de *firmware*. É uma interface simples que apenas mostra a informação necessária

ao operador. Na mensagem principal informa que está a ser executado o carregamento de *firmware* e na mensagem secundária é mostrado qual a parte do *firmware* está a ser atualizado, nomeadamente a `App0`, `App1` ou `Bootloader`. A barra de estado é atualizada aquando de cada um dos carregamentos. No caso de a plataforma ser Android, a mensagem exibida é `partition 1` e `partition 2`. Na Figura 4.32 estão visíveis todos os controlos e indicadores deste VI.

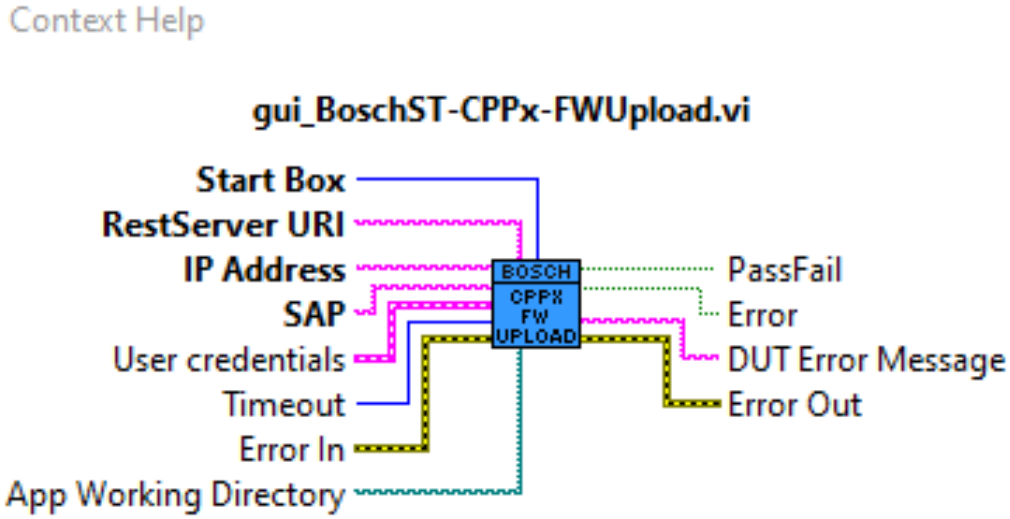


Figura 4.32: Controlos e indicadores do VI de carregamento de *firmware*

Este VI faz o carregamento da versão mais recente de *firmware* de forma automática. Como já foi referido, para cumprir este requisito é necessário que o VI execute pedidos HTTP à API, por forma a obter todos os dados necessários de um determinado produto, proceder ao *download* dos ficheiros de *firmware* necessários e efetuar o carregamento dos mesmos nas câmaras de vídeo vigilância. Com estes requisitos implementados a intervenção por parte da engenharia de testes é praticamente nula, estanto a estabilidade dos vários testes finais e funcionais assegurada, para além de automatizar todo o processo que até à data era manual.

O primeiro passo executado pelo VI é obter todos os dados do produto através da execução do VI `svi_BoschSTCPPxGetDUTVersion.vi`. Em caso de sucesso, o VI de carregamento de *firmware* faz um pedido à API para obter os dados referentes a um produto, executando o VI `svi_OvrP_BDL_REST_Request.vi`. No fluxograma da Figura 4.33 é possível visualizar os passos descritos até agora. Em caso de erro, o processo é terminado e é retornada uma mensagem com o respetivo erro. Em caso de sucesso, o passo seguinte é verificar se o *firmware* já existe localmente no diretório da aplicação.

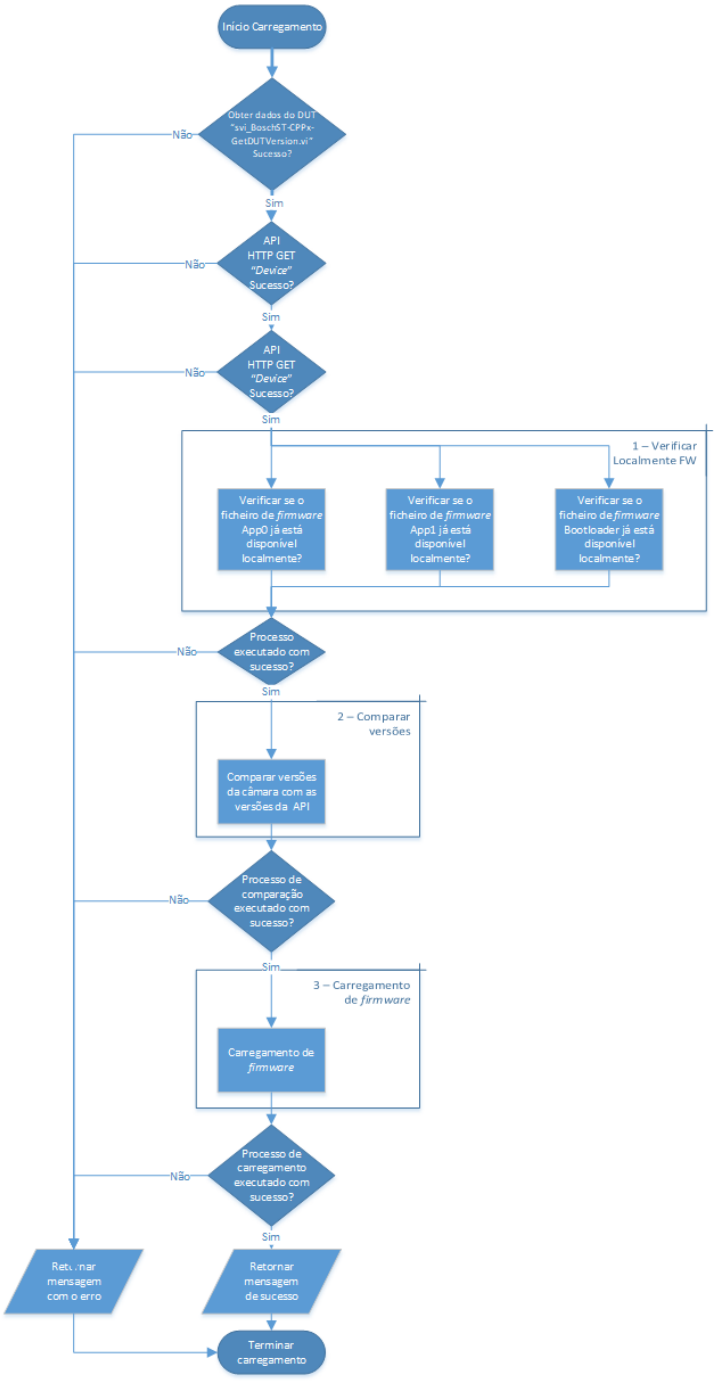


Figura 4.33: Carregamento de *firmware* - fluxograma

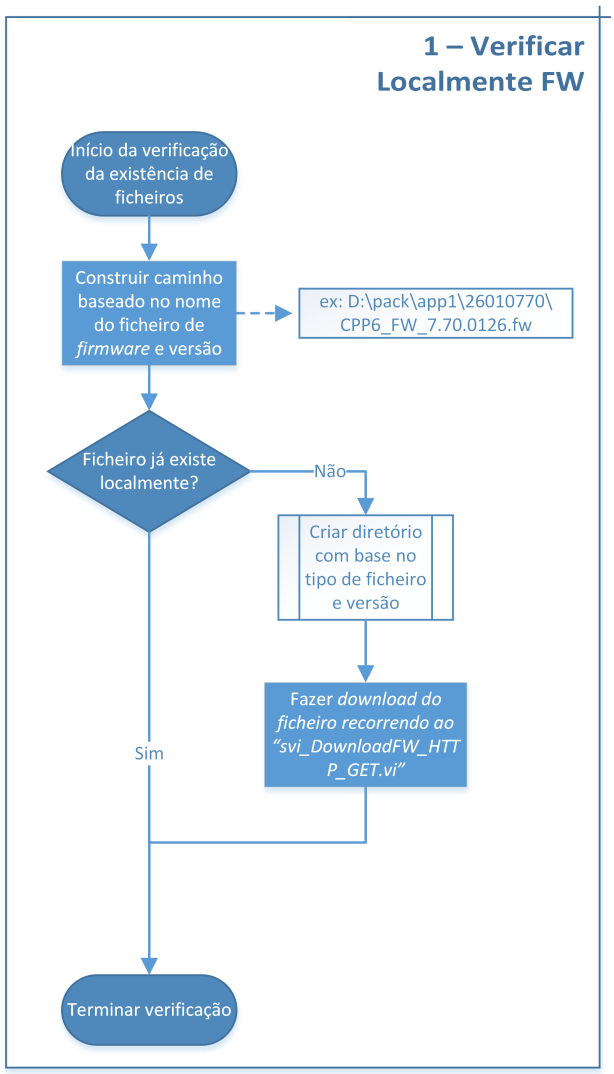


Figura 4.34: Verificação e *download* de *firmware* - fluxograma

No fluxograma representado na Figura 4.34, é possível verificar que o primeiro passo executado pelo VI é construir o caminho para verificar se o(s) ficheiros(s) de *firmware* já estão disponíveis na pasta de *firmware*. Caso não se encontrem os ficheiros, trata-se de um produto que foi atualizado ou que irá ser produzido pela primeira vez. Neste caso, o programa cria o diretório baseado na versão de *firmware* e no tipo de ficheiro, como o `App0`, `App1` ou `Bootloader`. Após criação do diretório, o VI faz o *download* do novo ficheiro para a respetiva pasta.

Para efetuar o *download* de um novo ficheiro foi desenvolvido um VI, cujo nome é `svi_DownloadFW_HTTP_GET.vi` e o interface está disponível na Figura 4.35.

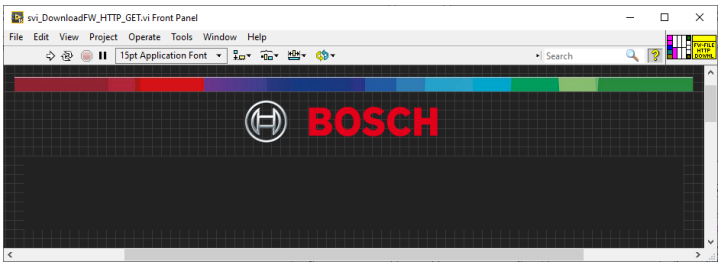


Figura 4.35: Interface Gráfica do VI `svi_DownloadFW_HTTP_GET`

O que este VI faz é um pedido `HTTP GET` às rotas `/api/app1/app1_version/app1_file`, `/api/app0/app0_version/app0_file` e `/api/bootloader/bootloaer_version/bootloader_file`.

Foi definido o *timeout* de 200 s segundos, por forma conseguir fazer o *download* de ficheiros maiores, nomeadamente os das plataforma mais recentes cujo tamanho é aproximadamente 1 GB. Como parâmetros de entrada, este VI tem:

- API URI;
- FW File;
- Timeout;
- Folder path to save the download;
- Token;
- Error IN.

A interface deste VI tem um indicador `Message display` para mostrar o ficheiro de *firmware* que está a ser descarregado. Após terminar o *download*, o VI compara a versão de *firmware* obtida da câmara com a obtida da API.

No fluxograma da Figura 4.36 é visível que o primeiro passo é verificar qual a plataforma da câmara. Como já referido, existem duas plataformas gerais, sendo elas a `Ambarella` e `Android`. A variável `CPP` é a variável utilizada para guardar a plataforma obtida da câmara. Se o CPP for igual a *Ambarella*, quer dizer que o *firmware* está divido em três partes e cada parte pode ter uma versão distinta. Em caso de diferenças verificadas entre as versões instaladas na câmara em comparação com a API, é adicionada ao *array* `To Upload` o booleano verdadeiro. Este tem a dimensão de três, sendo que o índice 0 corresponde à variável `App0`, o índice 0 à `App1` e o índice 2 à variável `Bootloader`. Nos controlos `app0_version`, `app1_version` e `bootloader_version` são preenchidos com a respetiva versão de

cada. No caso de não ser necessário atualizar alguma parte, o array é preenchido com um falso e o controlo respetivo com uma *empty string*.

No caso de a variável CPP ser igual a `Android`, apenas existe uma versão de *firmware*, pois este não está dividido em vários ficheiros como no outro caso. Apenas existe um ficheiro. Este ficheiro pode ser de uma versão *release* ou de uma versão *debug*. Como é possível verificar no fluxograma, se a versão da câmara for diferente da versão da API é adicionado ao array dois booleano verdadeiro, no índice 0 e 1 respetivamente, e aos controlo `app0_version` e `app1_version` a versão de *firmware* necessária para a atualização. Em caso de não ser necessário atualizar a câmara é feita a verificação se a versão instalada é uma versão *release* ou *debug*. Caso seja *release* não é adicionado ao array qualquer entrada, pois a câmara já possuí a versão correta. Em caso de ser *debug*, o procedimento é o mesmo utilizado no caso de ser necessário atualizar. Uma vez que 95 % da produção em Ovar é de câmaras cuja plataforma é a *Ambarella*, o foco foi desenvolver uma aplicação para esta plataforma e posteriormente adaptar à plataforma *Android*. Uma vez que a plataforma *Android*, como já referido, não contém `App0`, `App1` e `Bootloader` separados, de forma a não criar mais variáveis no *software*, foi decidido utilizar as existentes. É por esta razão que a plataforma *Androi*d utiliza as variáveis `App1` e `App0` para escrever os dados da câmara.

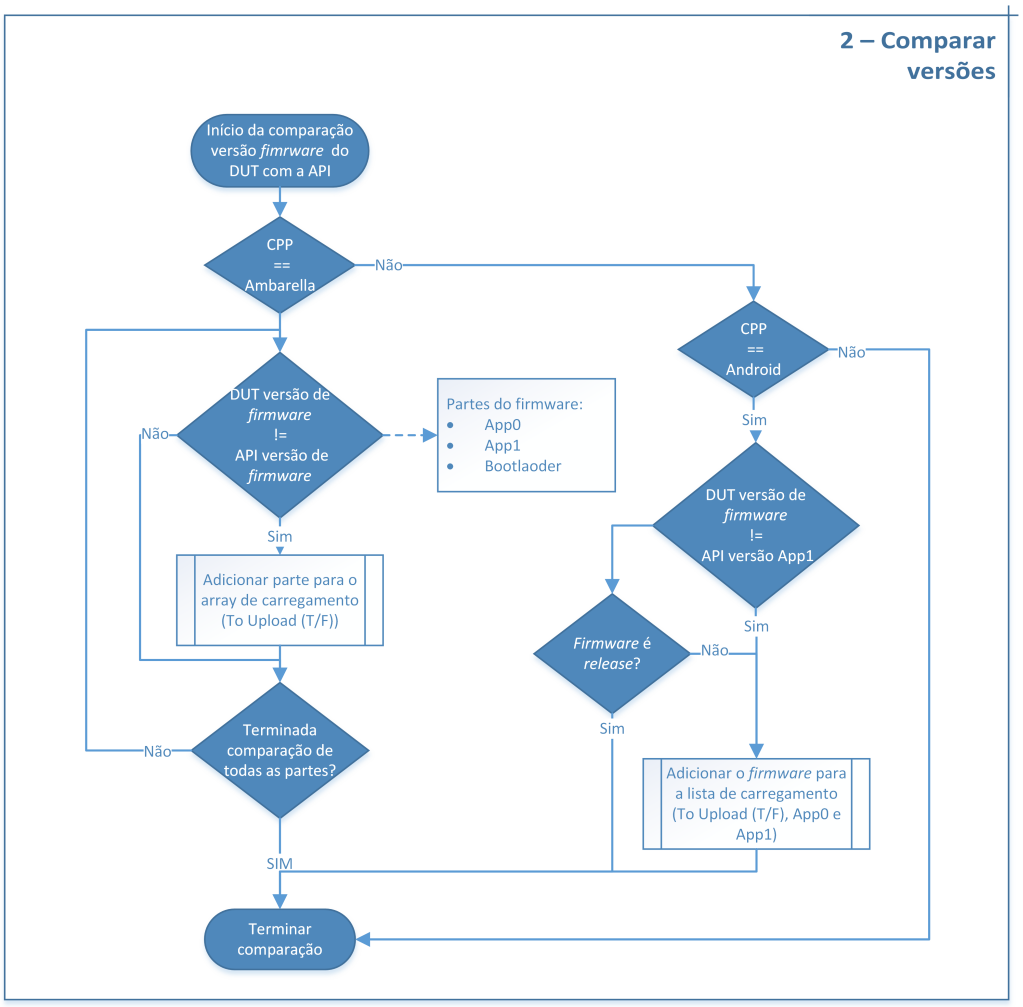


Figura 4.36: Fluxograma de Comparação entre dados do Produto e dados da API

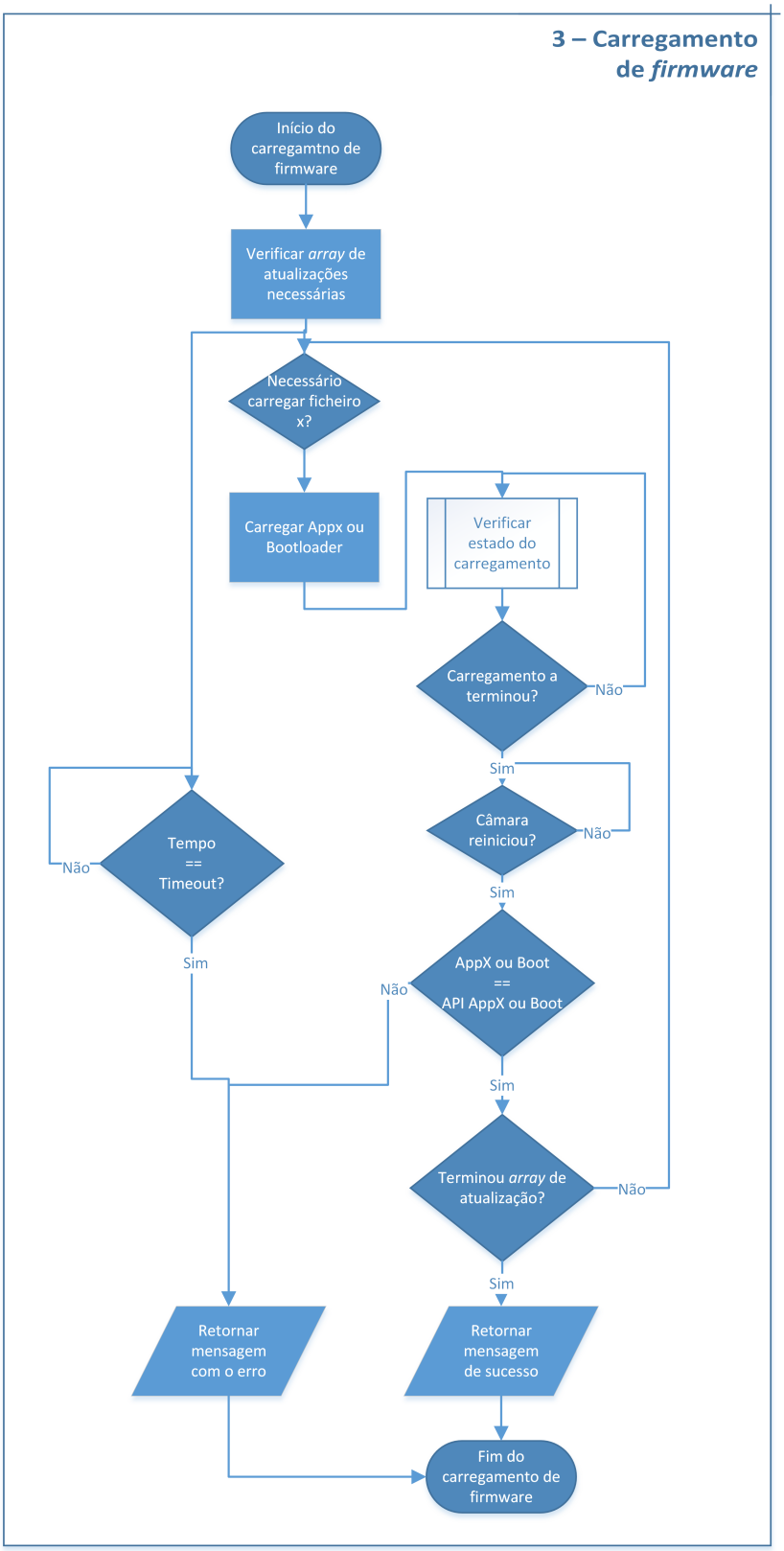


Figura 4.37: Fluxograma do Carregamento de *Firmware*

No fluxograma da Figura 4.37, pode verificar-se todos os passos de uma forma geral, necessários, para o carregamento de *firmware*.

Quando este VI é executado no posto de embalagem, são passados para as entradas deste VI o URI, o SAP, o IP da câmara, o *timeout*, o caminho para a pasta que contém os ficheiros de *firmware* locais e a linha azul pertence à variável de início de caixa coletiva. As saídas são os indicadores `PassFail`, `Error` e `Error Out`. Em caso de erro, é mostrada a mensagem que está no indicador `Error Out`. Em caso de sucesso, ou seja, `PassFail` a *True* e `Error` a *false*, o *software* do posto de embalagem continua o processo.

4.4.5 Estação de Carregamento de Firmware

A necessidade da criação desta estação de carregamento em paralelo de até 4 câmaras, surgiu devido à falta de capacidade de algumas das linhas para cumprir os objetivos de produção. Em alguns casos, como têm que fazer o carregamento não só ao `App1`, mas também às restantes partes, faziam com que o posto de embalagem se tornasse o *bottleneck* da linha. O maior impacto desta estação é para as câmaras cuja plataforma é *Android*, pois passam de demorar 18 min no posto de embalagem, para 4.5 min na estação de carregamento. Com o carregamento nesta estação, o posto de embalagem deixa de ser um problema, pois apenas faz a validação de que as câmara passaram na estação de carregamento e têm a versão mais recente instalada. De seguida é descrito o código criado para o desenvolvimento desta aplicação.

4.4.5.1 Estrutura do Projeto

Ao projeto são adicionadas todas as dependências necessárias para o desenvolvimento da solução. As pastas adicionadas são definidas com *auto populate*, de forma a apagar ou adicionar de forma automática ficheiros ou VI presentes nas mesmas. Expondo a estrutura:

- `_fwUpload.vi` - Utilizado para iniciar a aplicação;
- pasta `cfg` - Contém o ficheiro com as configurações;
- pasta `Database-FWUpload` - Contém os VI responsáveis pela comunicação com a base de dados;
- pasta `gbl` - Contém as variáveis globais;
- pasta `gui` - Contém o VI principal da aplicação;
- pasta `svi` - Contém os vários VI necessários para a construção da aplicação.

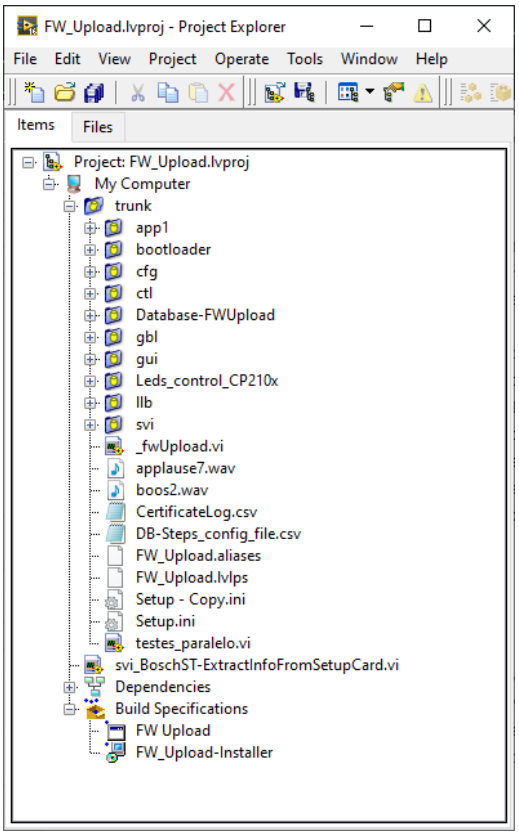


Figura 4.38: Estrutura do projeto - Estação de carregamento de *Firmware*

Apenas serão explicados os VI desenvolvidos de raiz para o projeto, sendo que os restantes utilizados foram desenvolvidos pela equipa de testes, e estão disponíveis no SVN para serem utilizados por todos os membros do grupo.

4.4.5.2 Lançamento da Aplicação

Este VI é responsável por arrancar a aplicação. Na Figura 4.39 está representada a interface deste VI. Tem, como indicadores, uma caixa de texto com informações e instruções, nomeadamente para indicar o estado da comunicação com a API e base de dados. Como controlos, possui uma caixa de texto de entrada e um botão de fecho da aplicação. A caixa de texto é preenchida exclusivamente através de um leitor de código de barras. Após serem validados os periféricos e a comunicação com a base de dados e a API REST, pode-se realizar o *setup* do posto através do leitor de código de barras, do cartão *kanban* e da ordem de produção. Os dados do produto são obtidos do `SAP` e a ordem de produção da `PO`. Não ocorrendo erros, a GUI da estação de carregamento fica disponível.

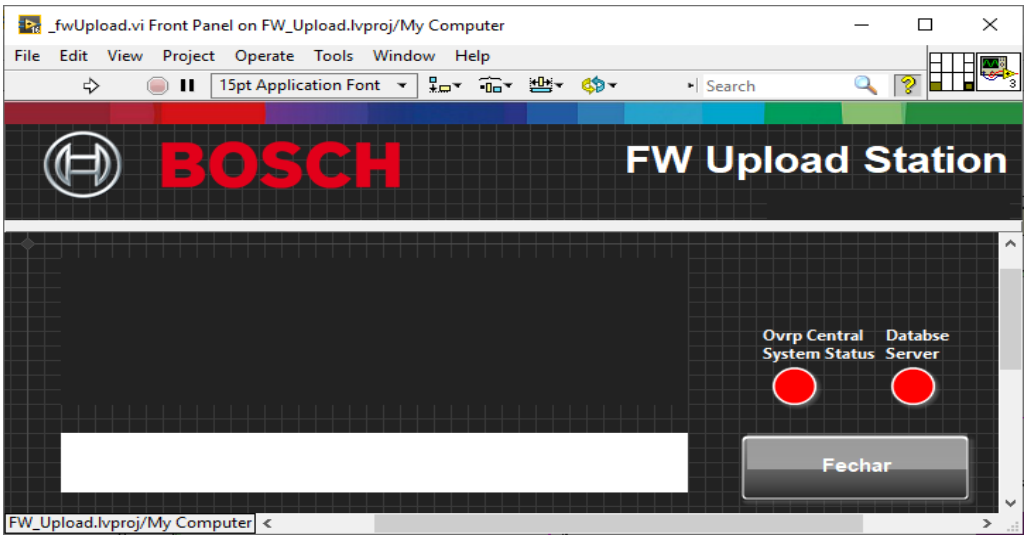


Figura 4.39: GUI - _fwUpload

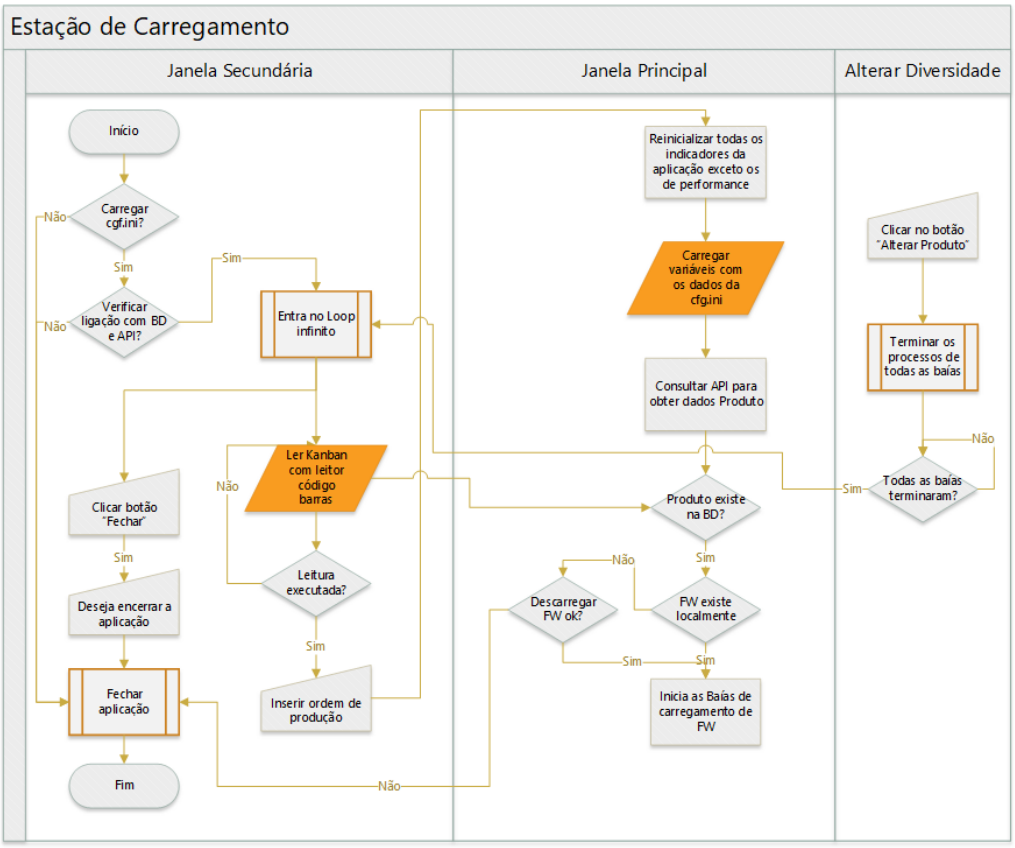


Figura 4.40: Fluxograma de Iniciação do Posto

Na Figura 4.40 está disponível o fluxograma de iniciação do Posto de Carregamento de *Firmware*. Nesta etapa, depois de arrancar a janela principal, verifica a versão de cada parte do *firmware* e verifica também se o(s) ficheiro(s) existem localmente. Caso existam, o posto fica pronto a ser utilizado, caso contrário, efetua o *download* dos ficheiros, ficando o posto pronto a ser utilizado.

4.4.5.3 Carregamento de Firmware nas Baías

Uma vez que para executar o carregamento em paralelo, é necessário invocar o mesmo VI desenvolvido em todas as baías, a forma correta de o fazer no LabVIEW está exemplificada na Figura 4.41, onde é possível verificar que a opção escolhida para o modo de execução é `Prellocated clone reentrant execution`. Este modo permite executar os VI em paralelo, pois em vez de os executar normalmente, cria um *clone* sempre que são executados.

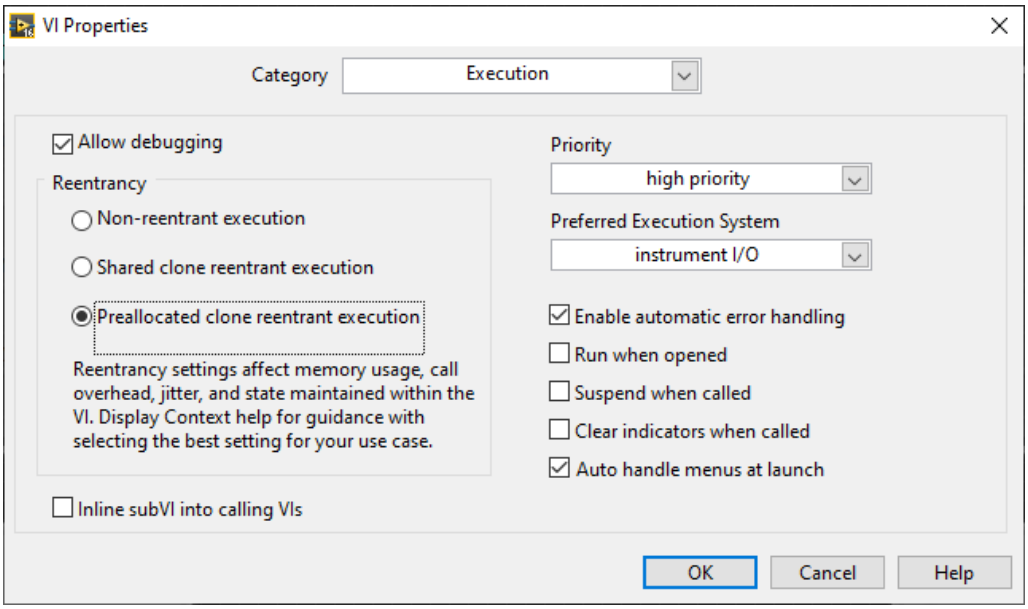


Figura 4.41: modo de execução dos VI

Para algumas situações, executar desta forma não é solução, pois pode existir perda de dados ou mesmo a sua mistura. Pelo que os VI que têm como função guardar dados entre passos, como é o caso do VI que guarda os valores que serão escritos na base de dados referente a cada câmara, são executados como `Non-reentrant execution`. Para este caso foram criados quatro VI, que são associados a cada uma das baías. Quanto às variáveis globais, não há possibilidade de criar clones das mesmas, sendo que é necessário criar variáveis globais específicas para cada uma das baías.

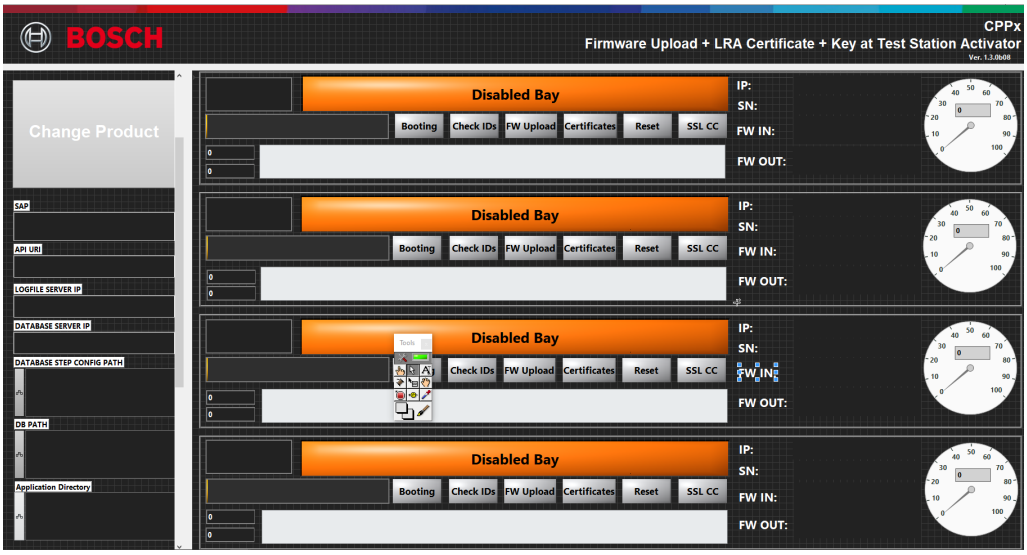


Figura 4.42: GUI estação de carregamento de *firmware*

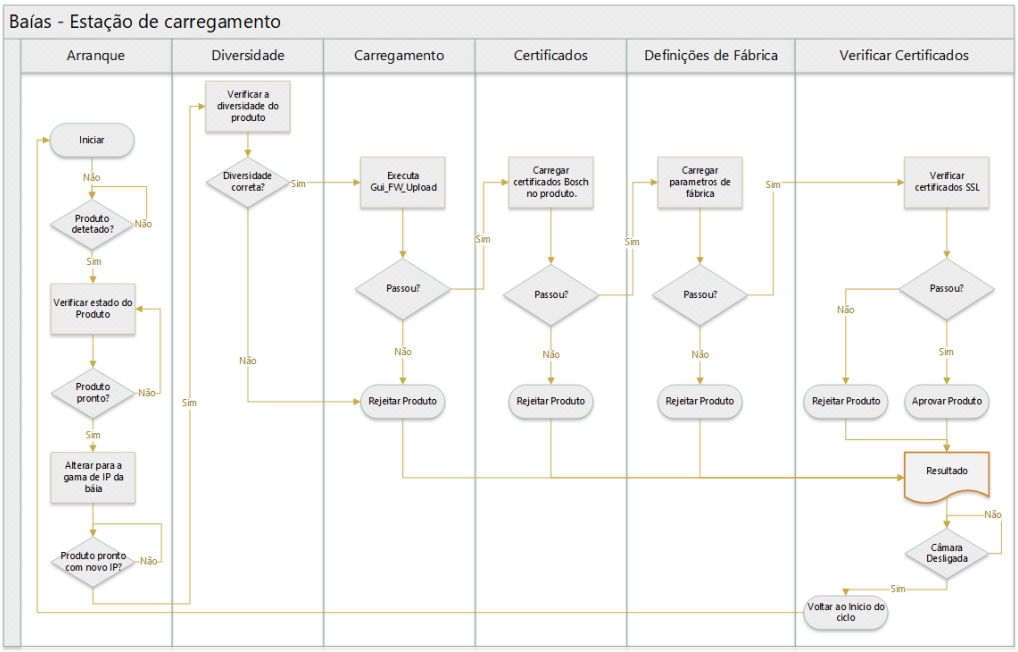


Figura 4.43: Fluxograma das Baías da Estação de Carregamento de *Firmware*

A Figura 4.42 mostra a interface criada para a estação de carregamento de *firmware*. A interface é constituída à esquerda por um botão e alguns indicadores com dados de configurações. No centro estão os indicadores referentes a cada baía.

Os indicadores a laranja são utilizados para dar *feedback* do estado de cada baía, como por exemplo, alertar quando uma câmara termina o processo ou apenas informar que a baía está livre. À direita, estão indicadores referentes ao número de câmaras que fazem o carregamento em cada baía. Quando o VI é iniciado, a primeira função executada em cada baía é a inicialização de todos os controlos e indicadores. Por forma a agilizar o processo produtivo, foi decidido que o processo de carregamento deveria ser iniciado automaticamente, após o operador ligar o cabo de rede à câmara. Enquanto o VI não encontra nenhum IP na placa de rede de uma determinada baía, fica à espera. Quando o VI deteta um IP na gama 169.254.*.*, ligado à placa de rede, avança para o próximo estado. Na Figura 4.43, está disponível para consulta o fluxograma referente ao funcionamento de cada baía.

Após se verificar que uma câmara foi ligada à placa de rede, o posto verifica se a mesma já fez o arranque completo, após estar pronta procede à alteração do IP para a gama da respetiva baía. Após esta alteração o produto volta a fazer um *reboot*. Como se vê no fluxograma, o *software* só avança quando verifica que a câmara já arrancou por completo. De seguida é verificado os `ID` para confirmar que a diversidade é a correta. Caso seja a correta, é chamado o VI de carregamento de *firmware* descrito na secção anterior e procede-se à atualização caso seja necessário. Após validação deste passo, faz-se o carregamento e pedido dos certificados necessários. Caso estes sejam carregados corretamente, procede-se ao passo seguinte que é carregar os parâmetros de fábrica, ou seja, fazer um *reset* via *software*. Após a câmara reiniciar, valida-se que o *reset* foi bem executado, com recurso a comandos BICOM que estão disponíveis no *firmware* da Bosch. Após este *resete*, o IP da câmara volta ao IP por defeito, sendo que para validar o certificados `SSL` é utilizado o IP inicial da câmara. Apenas se alterar o IP da câmara para a gama da respetiva baía, uma vez que para pedir certificados é necessário que o IP de quem pede esteja na mesma gama de IP do *reverse proxy* associado à placa de rede da respetiva baía, caso contrário não seria possível pedir certificados. No final do processo é exibido o resultado, sendo que a baía apenas volta a ficar disponível quando a câmara atual for desligada. Após desligar, o processo volta ao início, como se comprova pelo fluxograma.

4.5 Resumo

Neste capítulo foi descrito o desenvolvimento do projeto. Para a criação da API REST e do *back-office* utilizou-se a linguagem Python, com recurso ao micro-framework *Flask* de desenvolvimento de API REST e de aplicações *Web*. A biblioteca `sqlalchemy` permitiu o mapeamento e manipulação das tabelas da base de dados através de classes, ficando a criação e execução de *queries* independente do motor de base de dados utilizado. No que diz respeito à atualização

de *firmware* na linha de produção, o código foi criado em LabVIEW, uma vez que é a linguagem de programação utilizada em todos os testes desenvolvidos pela engenharia. O VI desenvolvido em LabVIEW foi instalado nos postos de embalagem, eliminando a necessidade de utilizar *software* de terceiros. Durante o desenvolvimento do projeto, foi necessário aumentar a capacidade de produção de várias linhas, transformando o posto de embalagem em *bottleneck*. Para ultrapassar esta questão foi desenvolvido uma estação de carregamento de *firmware* com capacidade de atualizar até quatro câmaras em simultâneo. Por último, esta estação passou a carregar e verificar os certificados das câmaras em vez do posto de embalagem, reduzindo o tempo de teste.

Capítulo 5

Testes e Resultados

Neste capítulo serão apresentados todos os testes executados com o objetivo de demonstrar o progresso e o funcionamento do sistema durante o desenvolvimento das aplicações e os testes após a introdução dos mesmos em produção.

5.1 Testes Funcionais

5.1.1 Funcionalidades do Back-office e API REST

A API REST e a aplicação *Web* do *back-office* foram desenvolvidas de mãos dadas, pois através das funcionalidades implementadas no *back-office* foi possível testar as rotas implementadas na API REST para satisfazer as mesmas. Para fazer o *sign in* no *back-office* é necessário colocar o email e a respetiva *password*. Em caso de sucesso o utilizador é redirecionado para o *back-office*. A rota que executa a validação do utilizador é a `API/backoffice/signin`, que devolve o email, a *password* encriptada e o nível de acesso. Quando o *back-office* está a carregar, o mesmo vai verificar se o utilizador é um `administrador` ou um `viewer`. No caso de ser administrador mostra todos menus implementados, no caso de não o ser, mostra apenas os dados de produção referentes a cada produto.

5.1.1.1 Menus Device Overview List e Sample Runs/Exception Device List

Quando o administrador carrega no botão `Check for updates`, este faz um pedido à API REST através da rota `.../pack_version_table` e quando o pedido é processado, é mostrada uma mensagem ao administrador com o resultado da operação. Para mostrar o resultado do pedido, o *back-office* ao receber a resposta da API, vai carregar os dados novos para a página HTML e a mesma é recarregada para mostrar o resultado da operação e atualizar os dados que são

exibidos ao administrador. Desta forma garante-se que sempre que há uma alteração efetuada por um administrador, o mesmo visualiza de imediato a alteração feita. Este foi o método encontrado para criar a interação entre o *back-office* e o administrador, uma vez que o *flask* é limitado nesta área, sendo uma das formas mais simples de criar a interação. Apesar de ser uma forma simples, não é muito eficiente, pois sempre que há uma ação por parte do administrador, o *back-office* é sempre recarregado, com todos os dados novos e os atuais, aumentando desta forma o tempo de processamento de cada pedido feito através do mesmo.

Para verificar que a introdução manual de produtos no *back-office* está a funcionar, foi adicionado um produto cujo `SAP` corresponde ao `F.01U.999.999`. Após clicar no botão e `Add Device` do menu `Sample Runs/Exceptions Device List`, é exibido uma janela modal com o formulário para preencher os dados do novo dispositivo. Este formulário está representado e preenchido com os dados do novo produto na figura 5.1

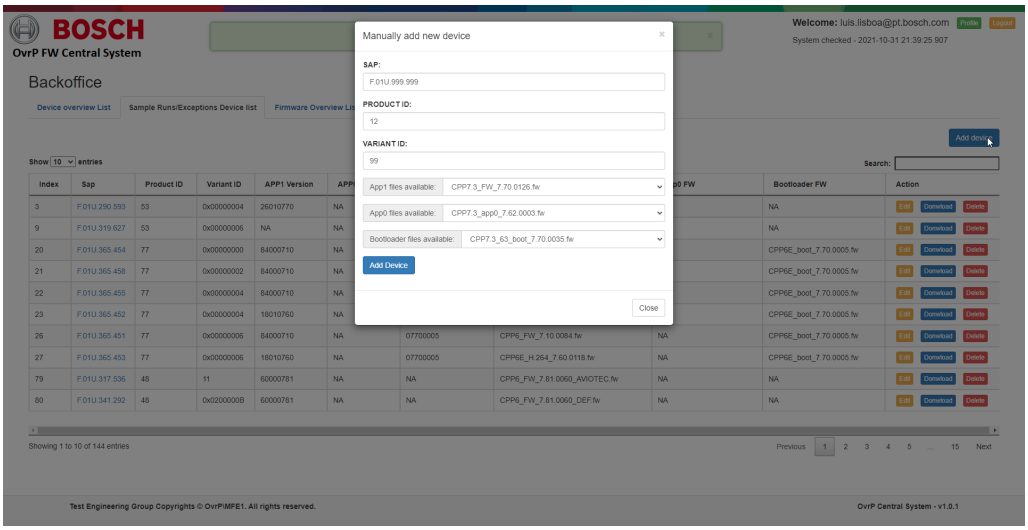


Figura 5.1: Teste Funcional Adicionar Produto

Após clicar no botão `Add Device` da janela modal, o mesmo submete os dados do formulário para a aplicação *Web* do *back-office*, mais especificamente para o recurso `backoffice/add_device_manually`. O resultado desta operação está representado na Figura 5.2, onde se pode visualizar através do modo *debug* do navegador, o pedido efetuado.

Para testar o botão da funcionalidade de editar, clicou-se no botão `Edit` da tabela correspondente ao SAP adicionado, ou seja, ao F.01U.999.999. É mostrada uma janela modal com um formulário pré preenchido com todos os dados do produto. O teste efetuado foi alterar o App1 de `CPP7.3_FW_7.70.0126.fw` para

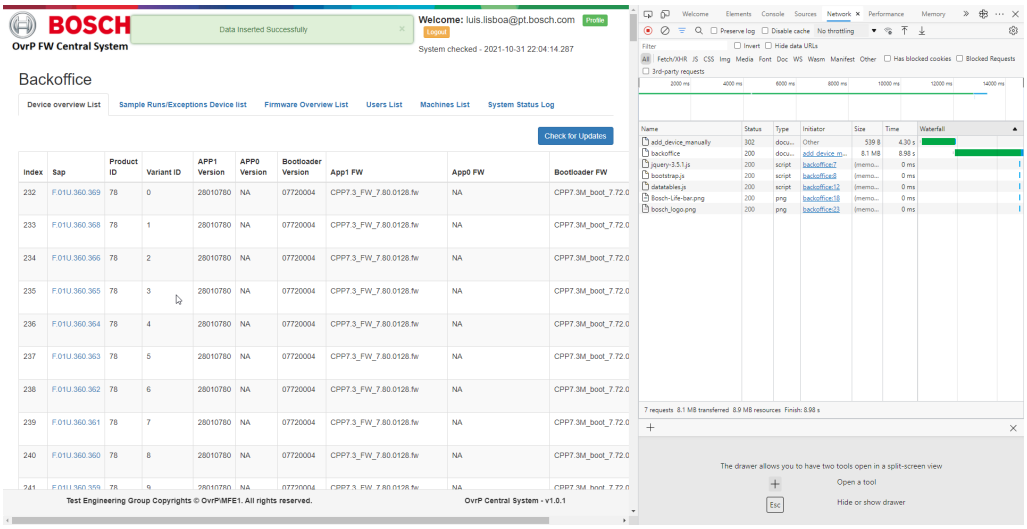


Figura 5.2: Resultado do Teste Funcional Adicionar Produto

`CPP7.3_FW_7.80.0128.fw` e passar o produto para modo de gestão automática, como se comprova pela Figura 5.3.

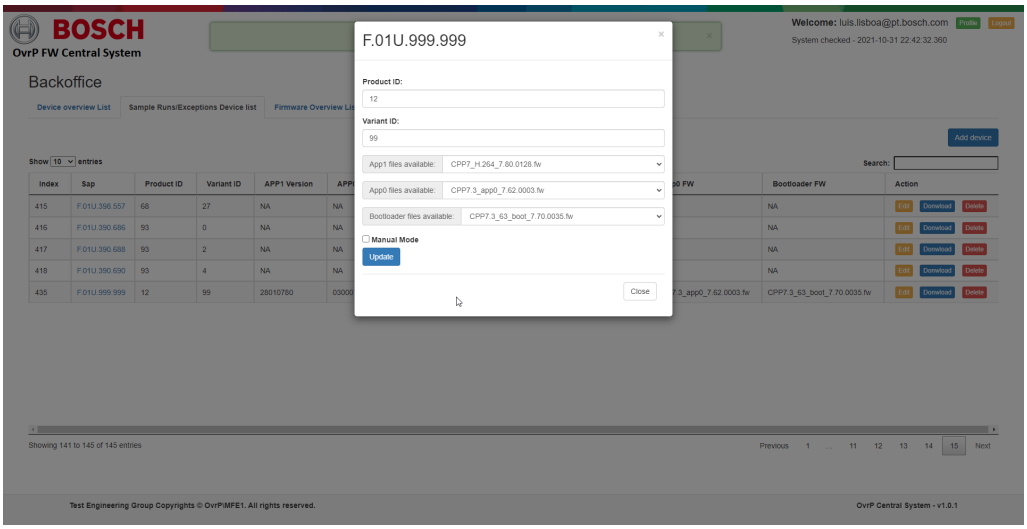


Figura 5.3: Teste Funcional Editar Produto

Após clicar no botão `Update`, o formulário é enviado para o `backoffice/edit_SAP_exceoption`, que por sua vez faz a atualização dos dados com o recurso à rota API/device/435. Após a atualização da API REST, esta envia o resultado da operação para o *back-office* e este mostra o mesmo ao administrador, como se pode confirmar na Figura 5.4.

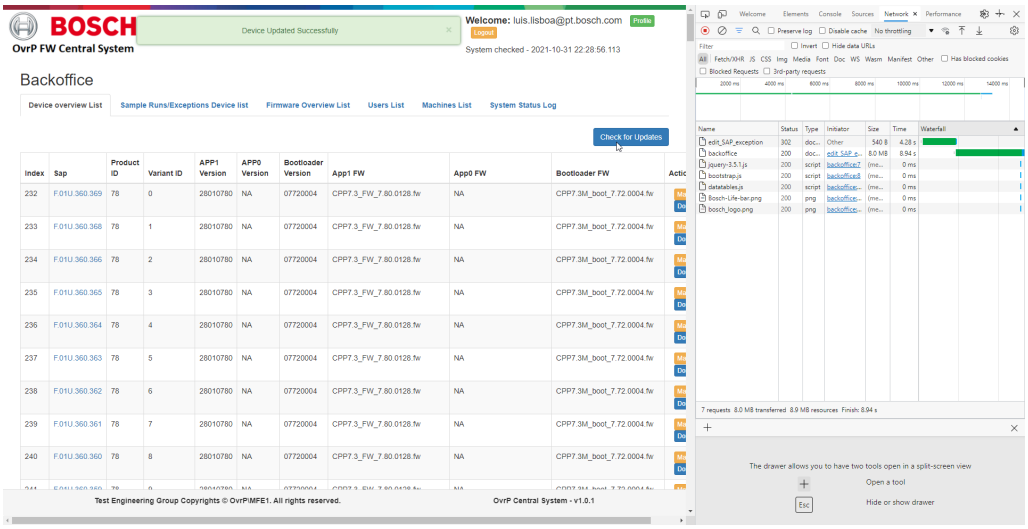


Figura 5.4: Resultado do Teste Funcional Editar Produto

Para voltar a colocar este produto em modo manual, é necessário ir à lista e clicar no botão de editar da coluna `Action`. Após clicar, é aberta uma janela modal com a informação do produto e com a opção de passar o mesmo para modo manual, como se pode verificar na Figura 5.5.

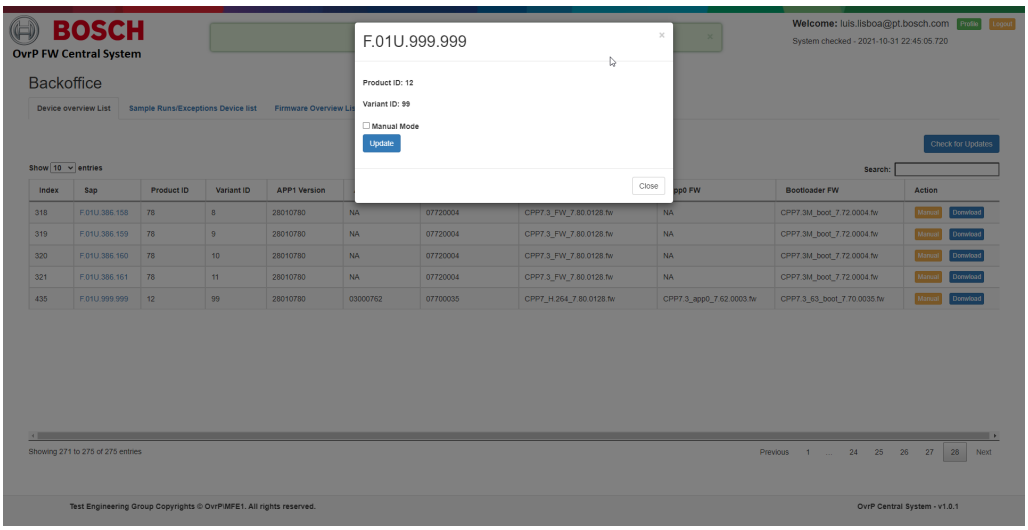


Figura 5.5: Resultasdo do Teste Funcional Editar Produto

Para atualizar, é então alterado o valor da `checkbox` e é enviado o pedido para ser processado pelo *back-office* através da função `edit_SAP_Fw_in_Production`. Esta função faz o pedido à API REST e esta faz toda a parte de comunicação e

alteração dos dados na tabela da base de dados. Na figura 5.6 é possível verificar a alteração de automático para manual.

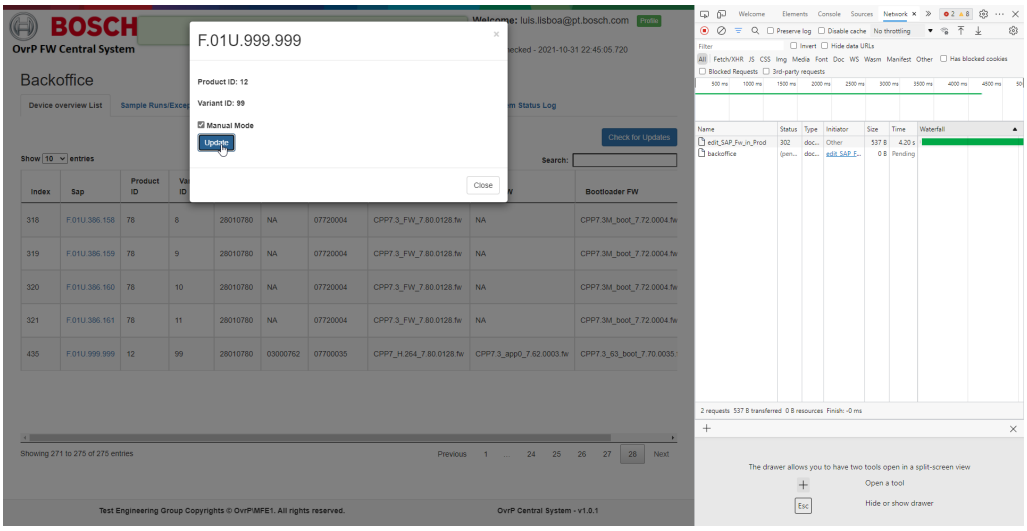


Figura 5.6: Teste Funcional Editar Produto de Modo Automático para Manual

O resultado desta operação, pode ser comprovado na Figura 5.7, onde se pode verificar que o produto está outra vez associado à tabela de exceções de produtos.

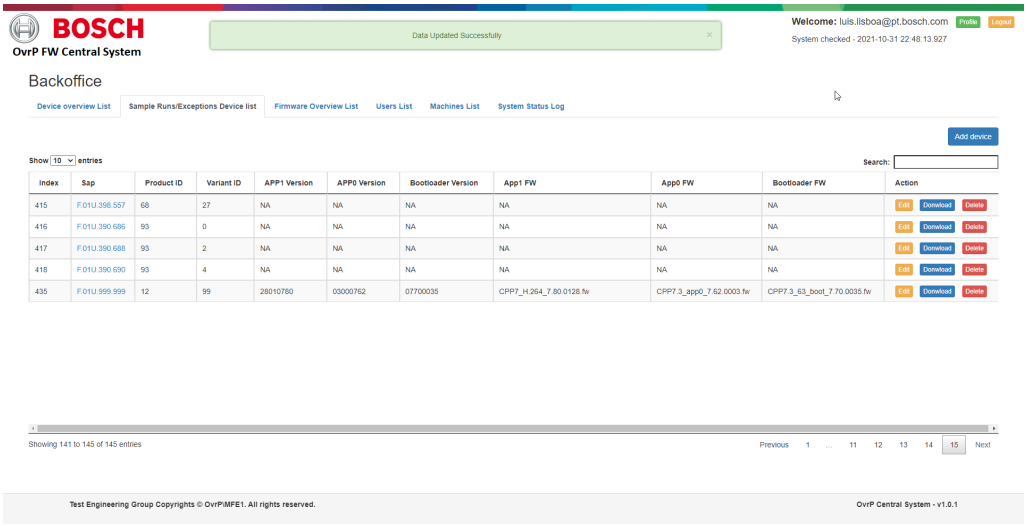


Figura 5.7: Resultado do Teste Funcional Editar Produto de Modo Automático para Manual

No *back-office*, existe a possibilidade de fazer o *download* das várias versões de *firmware*, sendo que para isso, é possível através do botão `download` associado a

cada produto e disponível na coluna `Action` ou através do menu `Firmware Overview List`. Em primeiro lugar, será testado o botão referente ao primeiro caso. Para isso, no menu `Sample Run/Exceptions Device List`, na linha referente ao SAP F.01U.999.999, ao ser clicado o botão `download`, é aberta uma janela modal com as várias versões de *firmware* correspondentes a este produto. Esta janela pode ser consultada na Figura 5.8, e é possível verificar que para além de cada versão, também existem botões para fazer o *download* de cada ficheiro.

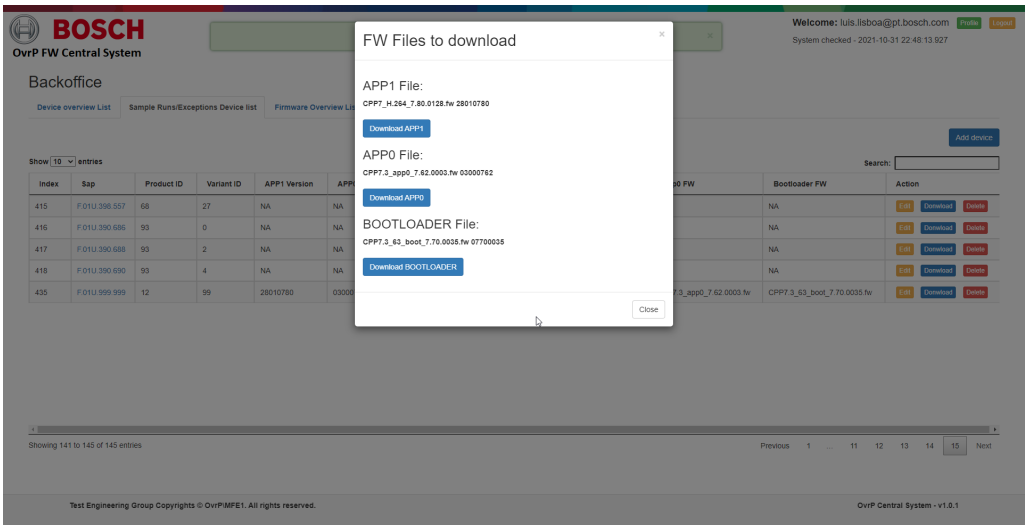


Figura 5.8: Teste Funcional *Download Firmware* do Produto

Ao clicar no botão `download APP1`, a função `download_APP1` do *back-office* faz um pedido `HTTP GET /app1/28010780/CPP7.3_FW_7.80.0128.fw` à API através desta rota. A API com recurso à função `send_from_directory`, envia o ficheiro para o cliente que fez o pedido. O resultado desta operação com a verificação do respetivo *download* do ficheiro de *firmware* `APP1` está representado na Figura 5.9, onde é possível verificar o pedido e o respetivo *download* do ficheiro.

Por último, referente ao produto F.01U.999.999, falta testar o botão `Delete`, que permite apagar o produto da lista. Após clicar neste botão, é aberta a janela modal com os dados do produto que se pretende remover, por forma a que o administrador confirme que está a remover o correto. Esta janela está disponível na Figura 5.10. Após a confirmação, a função `delete_SAP_packversiontable` do *back-office*, faz o pedido `HTTP DELETE API/device/435`, e a API faz todo o trabalho de apagar o registo da base de dados. Devolve o resultado ao *back-office* e este por sua vez recarrega a página e mostra o resultado ao administrador. Na Figura 5.11 é possível verificar o resultado da operação.

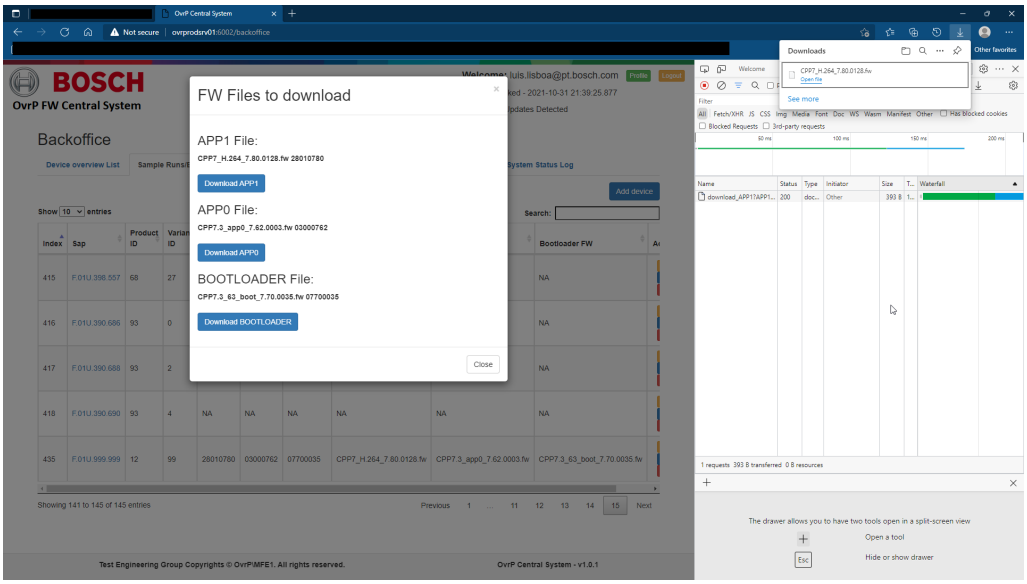


Figura 5.9: Resultado do Teste Funcional *Download Firmware* do Produto

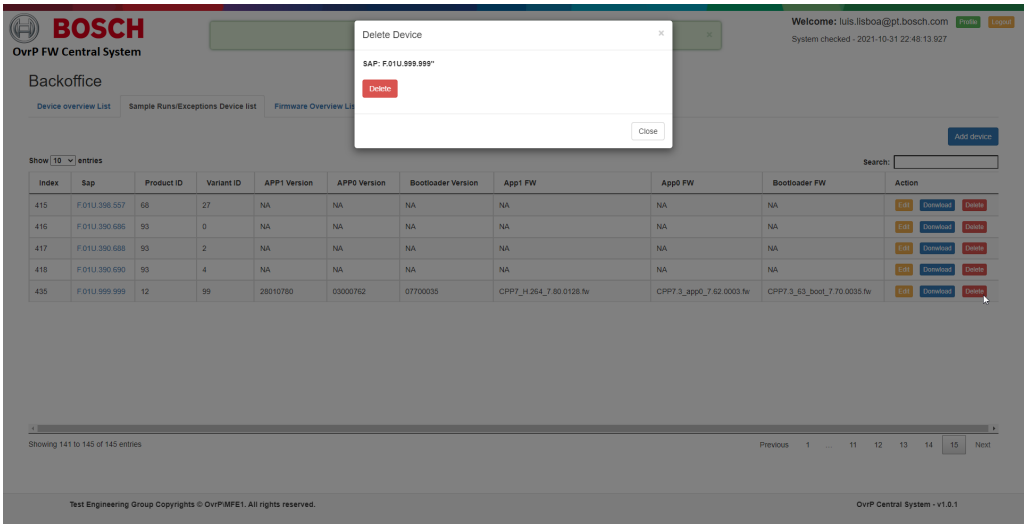


Figura 5.10: Teste Funcional Remover Produto

5.1.1.2 Menu Firmware Overview List

Neste menu serão testadas as funcionalidades de adicionar um novo ficheiro de *firmware*, de fazer o *download* e de apagar. Começando pela funcionalidade de adicionar, para cada um dos sub menus desta categoria, ou seja, sub menu `App1 Files List`, `App0 Files List` e `Bootloader Files List`, existe um botão para adicionar, sendo que no `App1` o nome do botão é `Add App1` e a mesma

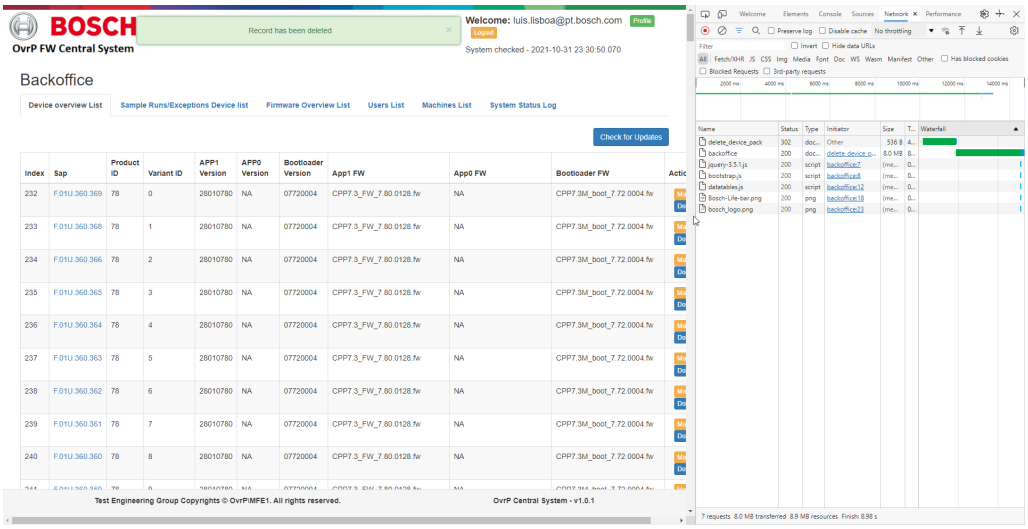


Figura 5.11: Resultado do Teste Funcional Remover Produto

lógica para os restantes. Ao clicar neste botão é lançado uma janela modal, representada na Figura 5.12, que contém um formulário em que a única entrada é do tipo ficheiro.

Para adicionar um ficheiro, como já mencionado no Capítulo 4, o nome do ficheiro tem que respeitar o formato `plataforma_FW ou !h.264_versão.fw`, por forma a manter este padrão, que é o utilizado pelo ficheiro que o sistema consulta para verificar a existência de atualizações. Este ficheiro está representado na Figura 5.13. De seguida, é adicionado o ficheiro que respeita este formato, e o resultado pode ser verificado na Figura 5.14.

Para testar a funcionalidade de *download* do ficheiro CPP7_H.264_9.99.9999.fw, é necessário carregar no botão `Download` disponível na coluna `Action`. Após carregar neste botão, abre uma janela modal que permite ao administrador descarregar o ficheiro. Ao carregar no botão de `Download`, o *back-office* através da função `download_App1` faz o pedido do ficheiro à API REST através da rota `HTTP GET API/app1/28010780/CPP7.3_H.264_9.99.9999.fw`. A API com recurso à função `send_from_directory`, envia o ficheiro para o cliente que fez o pedido. O resultado desta operação pode ser confirmado na Figura 5.15

Para apagar uma ficheiro, foi implementado o botão `Delete` que por sua vez abre uma janela modal, com o intuito de o administrador confirmar que está a apagar o ficheiro correto. Após confirmar, através da função `delete_app1` do *back-office* é feito o pedido `HTTP DELETE API/app1/56`, que correspondia ao *index* do *firmware* `CPP7_H.264_9.99.9999.fw`. O resultado da operação pode ser visualizado na Figura 5.16.

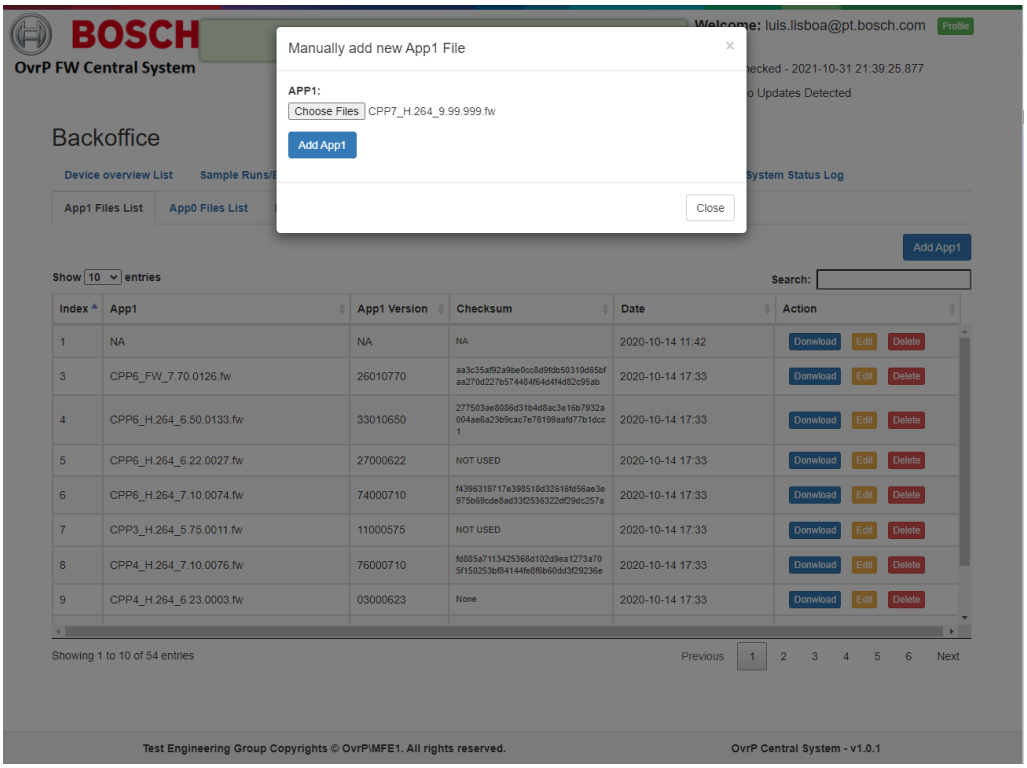


Figura 5.12: Teste Funcional da Janela Modal Adicionar `App1`

```
# ------------------------------------------------------------------------------
# Created: Thu Nov 26 15:10:15 2020
# Device Overview List Revision: 199
#
#! PRODUCT_ID, VARIANT_ID, Material Number, SAP #, Identification, APP1 Filename, APP0 Filename, Bootloader Filename, SAP description
#
# PRODUCT_ID:        Product ID that is programmed into the ID chip
# VARIANT_ID:        Variant ID that is programmed into the ID chip
# Material Number, SAP #:SAP code used for automatically selecting the device in the combo box
# Identification:    Identification if SAP code is not unique
# APP1 Filename:     Filename of APP1
# APP0 Filename:     Filename of APP0
# Bootloader Filename:Filename of Bootloader
# SAP description:   Used by FW to show in the web pages, default: missing

#SITE = 1
#Project: Hubble 1-2
48,  0x00000000,  F.01U.285.362,,         CPP6_FW_7.80.0128.fw,        ,                        ,                        "DINION IP starlight 8000 5MP IVA"
#Project: Fiedler
48,  0x00000007,  F.01U.309.129,,         CPP6_FW_7.80.0128.fw,        ,                        ,                        "DINION IP ultra 8000 MP, 12MP, IVA, MBF"
#Project: Rosen
53,  0x00000000,  F.01U.295.129,,         CPP6_FW_7.80.0128.fw,        ,                        ,                        "FLEXIDOME panoramic 7000 12MP 180"
53,  0x00000004,  F.01U.290.593,,         CPP6_FW_7.80.0128.fw,        ,                        ,                        "FLEXIDOME panoramic 7000 12MP 180 A-IVA"
#Project: Walmart
53,  0x00000000,  F.01U.359.791,,         CPP6_H.264_6.50.0133.fw,     ,                        ,                        "Dome 12MP 180° WM"
53,  0x00000004,  F.01U.359.790,,         CPP6_H.264_6.50.0133.fw,     ,                        ,                        "Dome 12MP 180° IVA WM"
#Project: Amazon
53,  0x00000004,  F.01U.349.590,,         CPP6_H.264_6.50.0133.fw,     ,                        ,                        "Fixed dome 12MP 180° IVA AMZ"
#Project: Rosen
```

Figura 5.13: Ficheiro version_table.txt

Uma vez que os restantes testes para o `App0` e `Bootlaoder` são os mesmos e foram também validados desta forma, não existe a necessidade de os demonstrar neste capitulo.

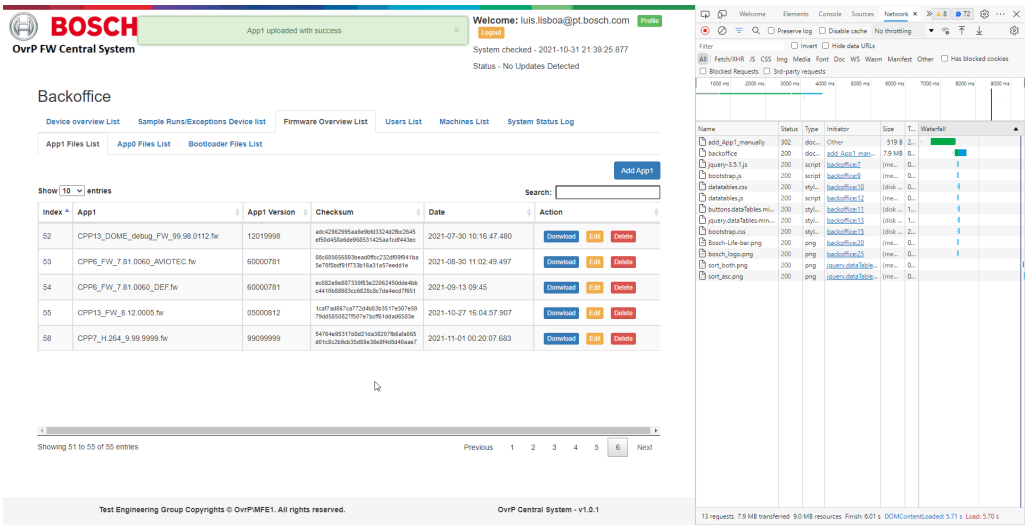


Figura 5.14: Resultado do Teste Funcional da Janela Modal Adicionar `App1`

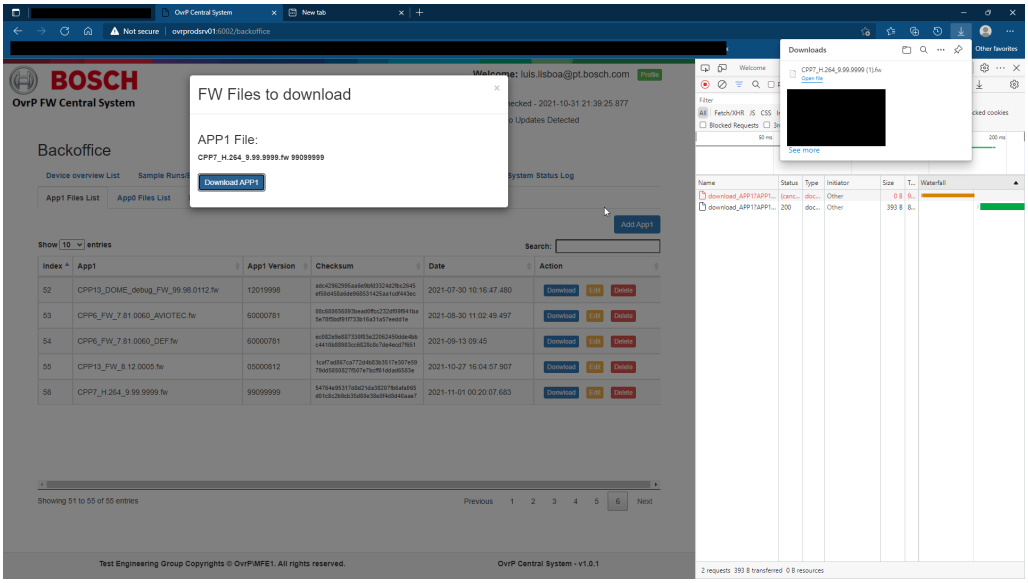


Figura 5.15: Teste Funcional da Janela Modal *Download* `App1`

5.1.1.3 Menu Users List

Para testar o menu de gestão de utilizadores, foram realizados vários testes a todas as funções existentes, como adicionar, editar e remover. O administrador ao aceder ao menu `Users List` do *back-office*, tem disponível um botão que permite criar utilizadores, `Add new user account`. Ao clicar neste botão, abre uma janela modal com um formulário onde constam as entradas, `email`, e `Access`

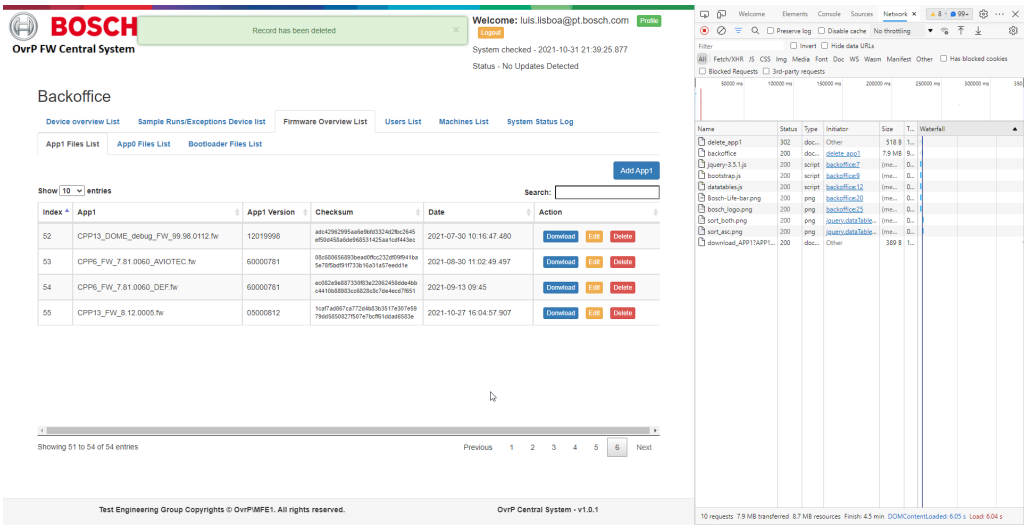


Figura 5.16: Resultado do Teste Funcional da Janela Modal Remover `App1`

`Level`. Para adicionar o utilizador é introduzido o email no primeiro campo, que para teste foi adicionado o email `teste@pt.bosch.com`. O tipo de acesso definido para este utilizador foi de `Viewer`, como se pode verificar pela Figura 5.17

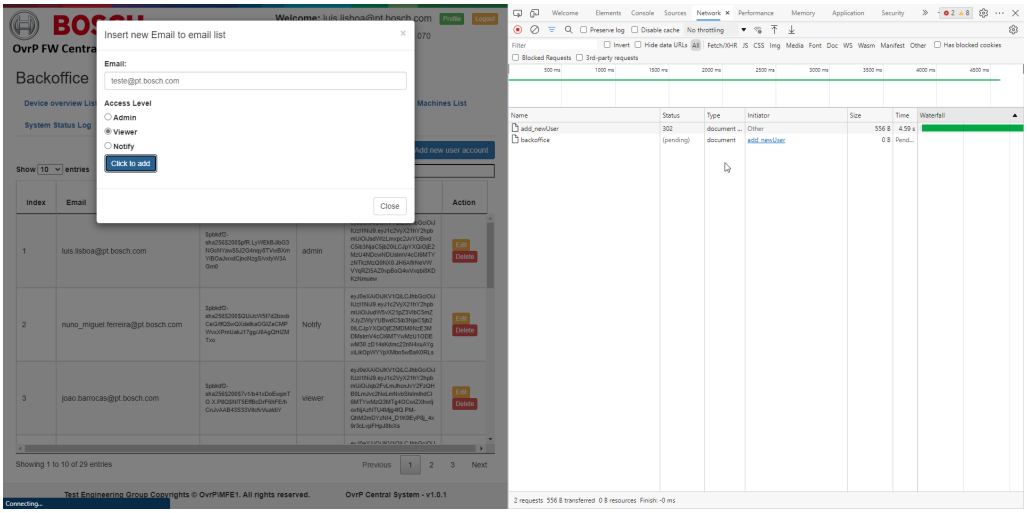


Figura 5.17: Teste Funcional Janela Modal Adicionar Utilizador

Ao clicar neste botão é invocada a função implementada no *back-office*, a `add_newUser`, que por sua vez, para processar o pedido faz o pedido `HTTP POST` à rota `API/user`. Esta processa o pedido, adiciona o utilizador em caso de não existir nenhuma entrada igual na tabela e devolve o resultado ao *back-office*, que

por sua vez mostra o resultado através da função *flash* ao administrador. O resultado da operação está disponível na Figura 5.18, onde se pode verificar que se adicionou com sucesso o utilizador teste@pt.bosch.com.

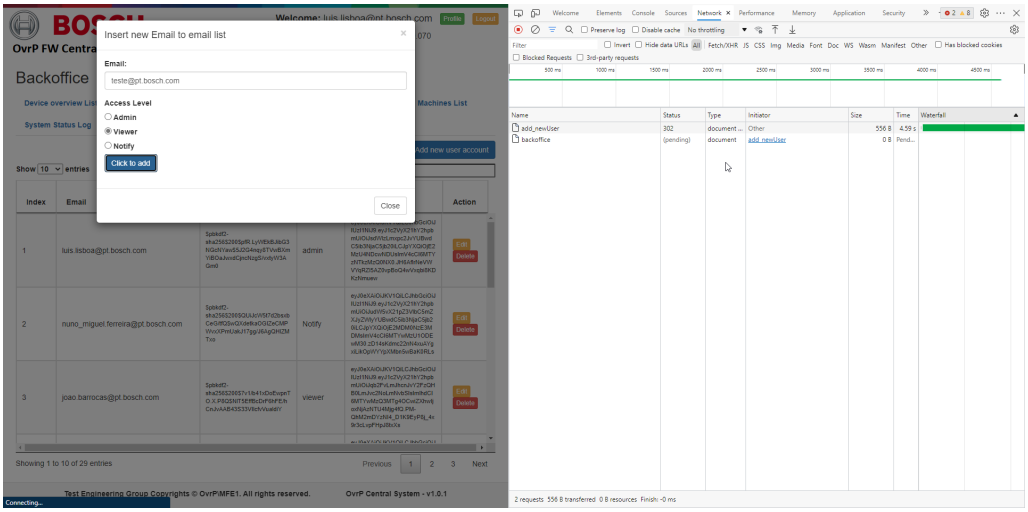


Figura 5.18: Resultado do Teste Funcional Janela Modal Adicionar Utilizador

Após adicionar o utilizador, foi testada a opção de editar os dados do utilizador, através do botão `Edit`, que por sua vez é aberta uma janela modal com os dados do utilizador, sendo eles o email, a *password* encriptada (sh256), o *token* e o nível de acesso. Para testar, alterou-se o nível de acesso do utilizador para `Notify`, alterando o acesso no formulário e submetendo o mesmo através do botão `update`. O *backoffice* através da função `update_user` lê os dados do formulário e envia-o para a API REST através da rota `HTTP PUT .../user/56`, onde o 56 corresponde ao índice do utilizador teste@pt.bosch.com. O resultado da atualização está disponível para consulta na Figura 5.19.

A última função implementada para gerir os utilizadores é a opção de apagar do sistema, sendo que para isso basta clicar no botão `Delete` e de imediato é aberta uma janela modal com os dados do utilizador para o administrador confirmar que realmente quer remover do sistema. Após confirmação, o *Backkoffice* envia um pedido à API REST através da rota `HTTP DELETE API/user/56`, por forma a que esta apague o utilizador da tabela `users`. O resultado da operação pode ser verificado na Figura 5.20.

5.1.1.4 Menu Machines List

Este menu foi dividido em dois submenus, sendo que já foram referidos no capítulo anterior. Começando pelo submenu `OvrP ITM Machines`, que tem duas funções implementadas, a de adicionar e a de remover. Para adicionar é necessá-

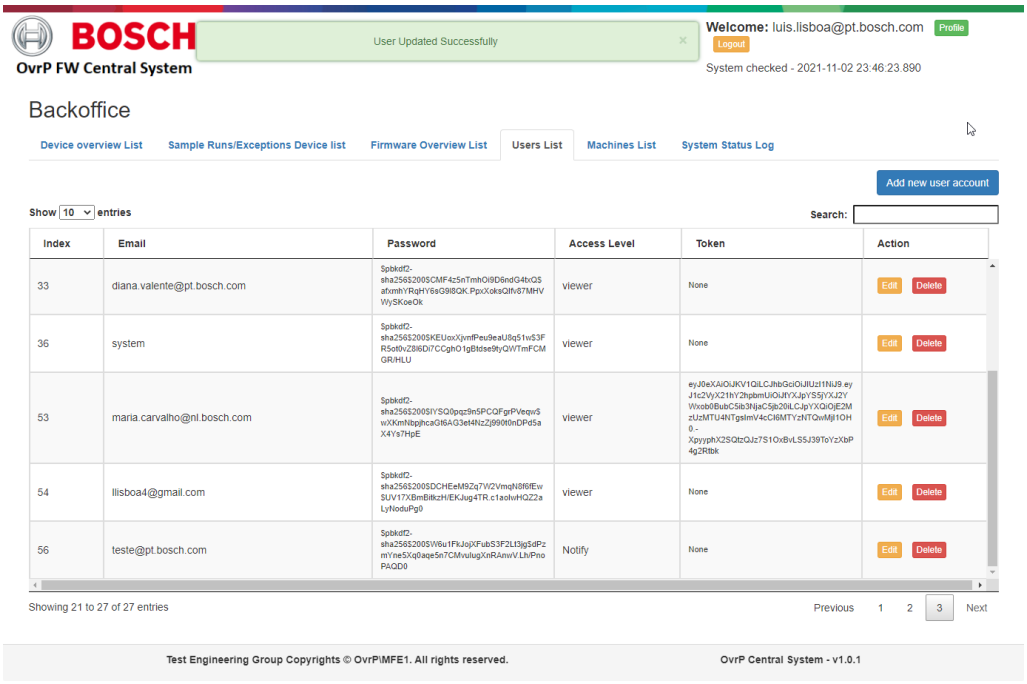


Figura 5.19: Resultado do Teste Funcional Janela Modal Editar Utilizador

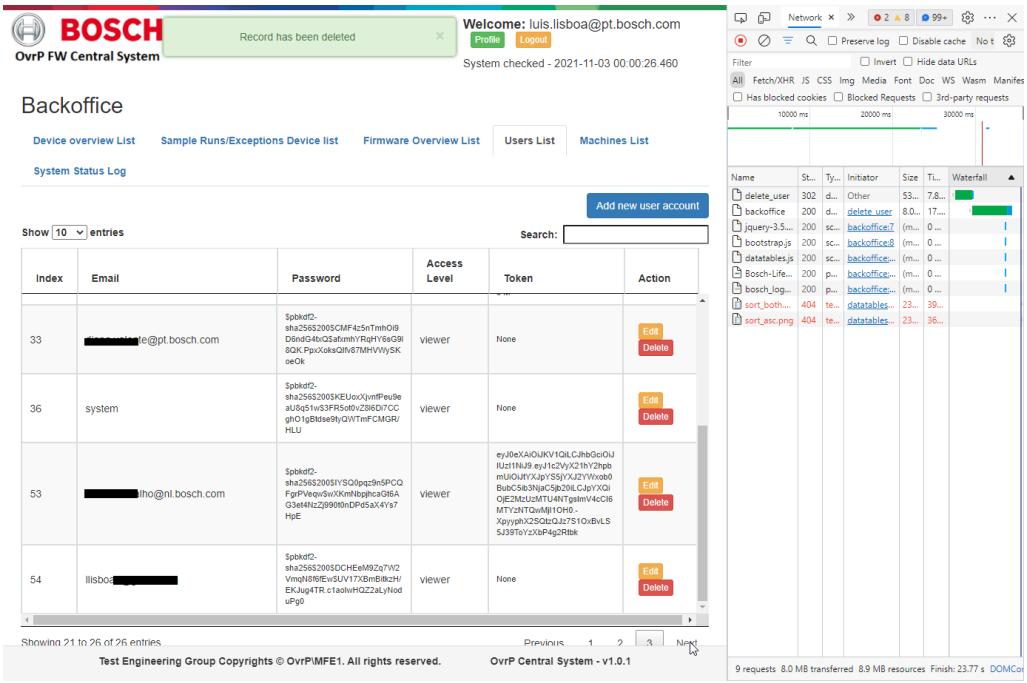


Figura 5.20: Resultado do Teste Funcional Janela Modal Remover Utilizador

rio clicar no botão `Add ITM Machine`, e uma vez que estas máquinas são geridas pelo responsável de rede da área de manufatura, o que aparece no formulário da janela modal é a lista de todas as máquinas existentes na produção, e dessa lista escolhe-se a que se pretende adicionar ao serviço. Para teste foi adicionada a máquina OVRP1109, como se pode verificar pela Figura 5.21. Ao clicar no botão `Add ITM Machine`, a máquina selecionada é processada pela função do *back-office* `addnewMachineITM`. Por sua vez, esta função faz o pedido à API para adicionar a máquina através da rota `HTTP POST api/machine`. A API REST adiciona e devolve o resultado ao *back-office*.

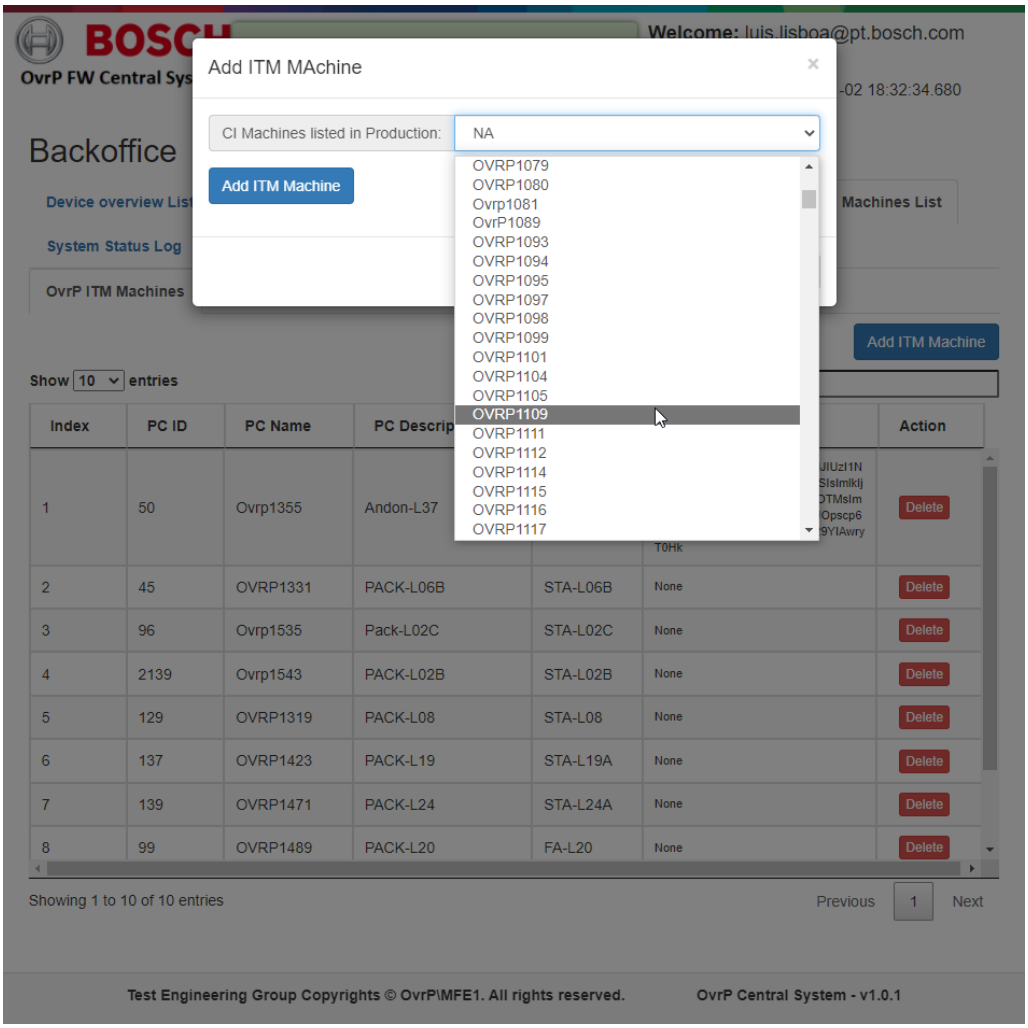


Figura 5.21: Teste Funcional Janela Modal Adicionar Máquina

Para testar a funcionalidade de remover, clicou-se no botão de `Delete` da coluna `Action` referente à estrada da máquina OVRP1109. Após clicar no botão, aparece uma janela modal a pedir ao administrador para confirmar que é

realmente a máquina que pretende remover. Após a confirmação, é enviado um pedido `HTTP DELETE API/machine/19`, sendo que o 19 corresponde ao índice da máquina OVRP1109. Na Figura 5.22 pode consultar-se a janela modal para confirmar que realmente se pretende apagar o registo.

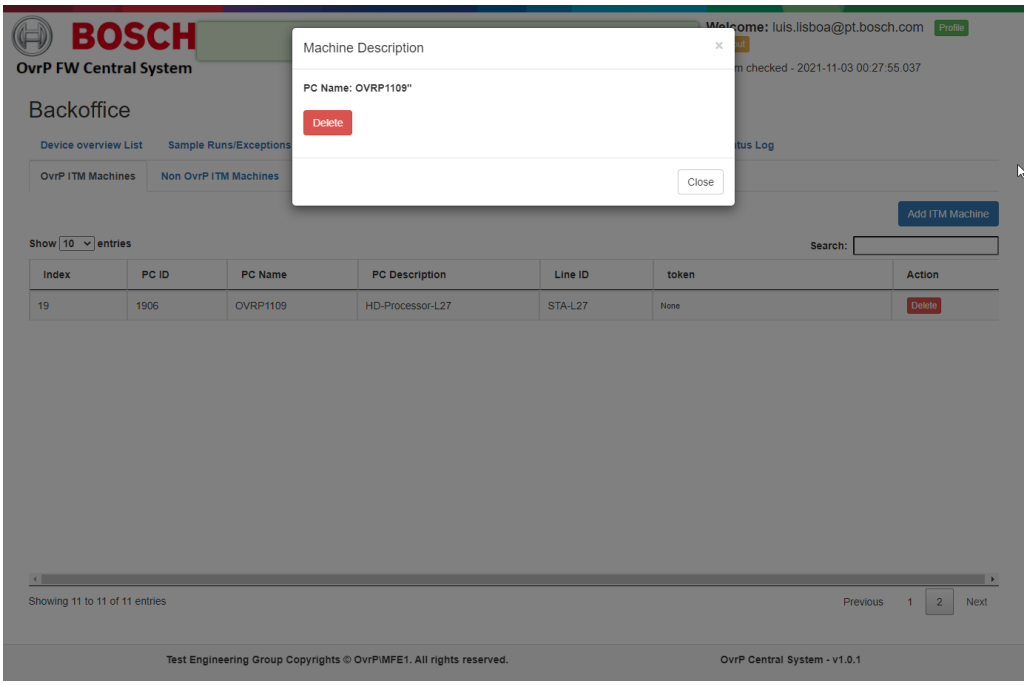


Figura 5.22: Teste Funcional Janela Modal Remover Máquina

Na Figura 5.23 é visível o resultado da operação.

Para testar as funções desenvolvidas no *back-office* para gerir as máquinas não pertencentes à gestão do responsável de IT da área de manufatura, e as respetivas rotas implementadas para processar os pedidos de todas as funções do *back-office* referentes à gestão de máquinas que não pertençam à área de ITM, foi criada uma máquina com o nome `OVRP9999`, a descrição é `TEST MACHINE` e a linha é a `L999`. Para adicionar a máquina, clicou-se no botão `Add NonITM Machine`, que após clicar validou-se que abre uma janela modal com um formulário e os respetivos parâmetros mencionados como se pode comprovar pela Figura 5.24.

A função do *back-office* `add_newMachine` lê os dados que estão no formulário e envia o pedido `HTTP POST API/mahcine`, em que no *body* do pedido são enviados os dados e a chave "CIName"com o valor"None". Desta forma, a API sabe que a máquina pertence ao grupo das não geridas pelo ITM. Após adicionar e a API devolver o resultado da operação, tendo sido este positivo, vai ser testada a funcionalidade de editar, onde para isso é necessário clicar no botão `Edit`. Após clicar neste botão, abre uma janela modal com um formulário e todos os dados

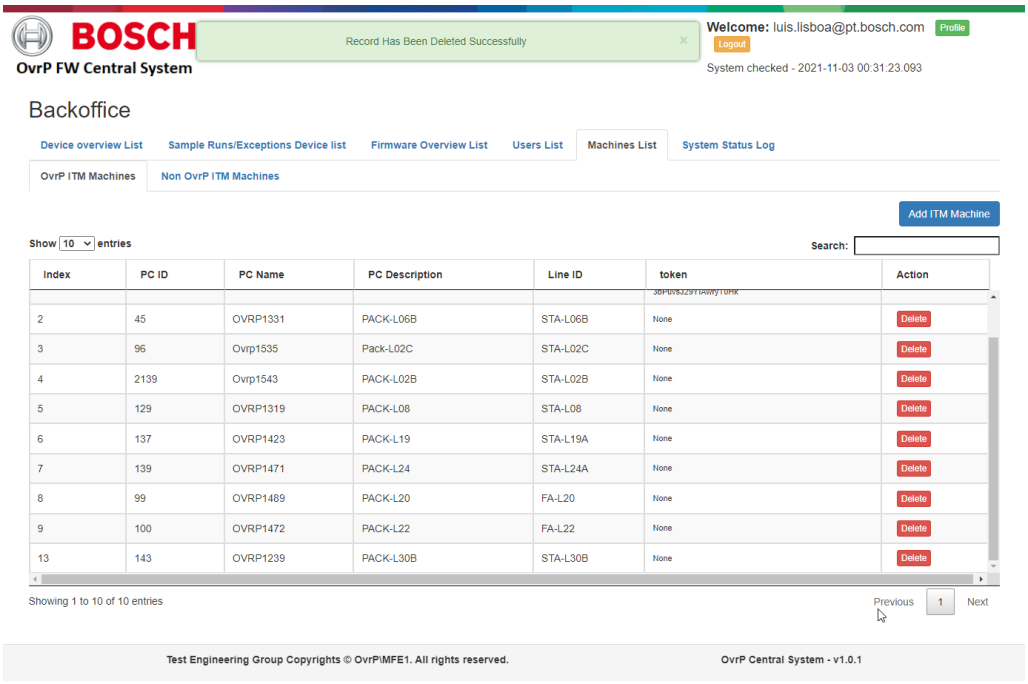


Figura 5.23: Resultado do Teste Funcional Janela Modal Remover Máquina

referentes à maquina. Para testar esta funcionalidade, vai ser alterada a linha de L999 para L9. O formulário pode ser verificado na Figura 5.25.

Após alterar o campo linha do formulário e clicar em `Update`, foi validado que a função `update_machie` do *back-office* está a funcionar. Esta função faz um pedido `HTTP PUT` à API REST através da rota `API/machine/51`, em que no *body* envia em JSON os dados para atualizar. Na figura 5.26 pode ser verificado o resultado da operação.

Por último, para apagar a máquina, clica-se no botão `Delete`, abre a janela modal e após confirmar que se pretende realmente apagar o registo, é executado a função do *back-office* `delete_machine_from_machine_list`, que por sua vez faz um pedido à API REST para pagar o registo através da rota `.../machine/<machine_table>/<machine_id>`, neste caso o `machine_table` corresponde ao à tabela, e o machine_id ao índice da máquina. O resultado da operação de apagar a máquina OVRP9999 está disponível na Figura 5.27

5.1.2 Firmware Upload

Neste sub-capítulo, estão descritos os testes executados na produção durante a validação do novo *software* integrado no posto de embalagem. Após o responsável por este *software* ter integrado o VI `svi_FW_Upload.vi` desenvolvido neste

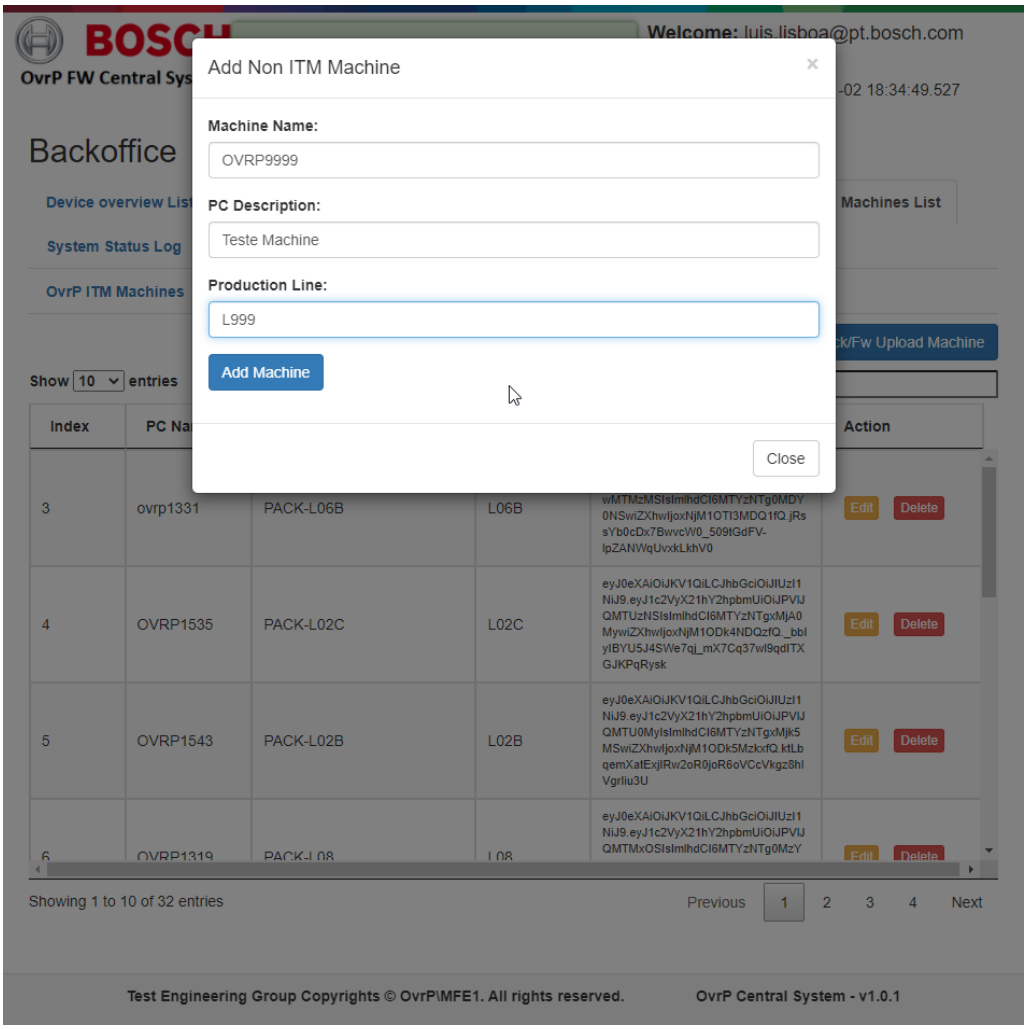


Figura 5.24: Teste Funcional Janela Modal Adicionar Máquina Não ITM

projeto, o passo seguinte foi escolher uma linha piloto e validar todas as novas alterações. Para isso, sempre que se pretende validar alguma alteração no código é necessário compilar o mesmo, pois nos computadores de produção o ambiente de desenvolvimento do LabVIEW não está instalado. Desta forma, para iniciar o posto de embalagem, o operador lê a diversidade que está a ser produzida com recurso a um leitor de código de barras e ao cartão *kanban* de *setup*. Ao ler, o posto de embalagem consulta a base de dados e carrega todos os parâmetros necessários, como os códigos únicos que identificam o produto e as respetivas etiquetas. Para iniciar o teste, basta conetar a câmara via *ethernet* ao *switch POE* que por sua vez está ligado ao computador. A câmara arranca com o DHCP ligado, e é sempre atribuído o IP 192.168.0.1. Quando a câmara é detetada, o posto de embalagem verifica quando a mesma faz o *boot* a 100 %,

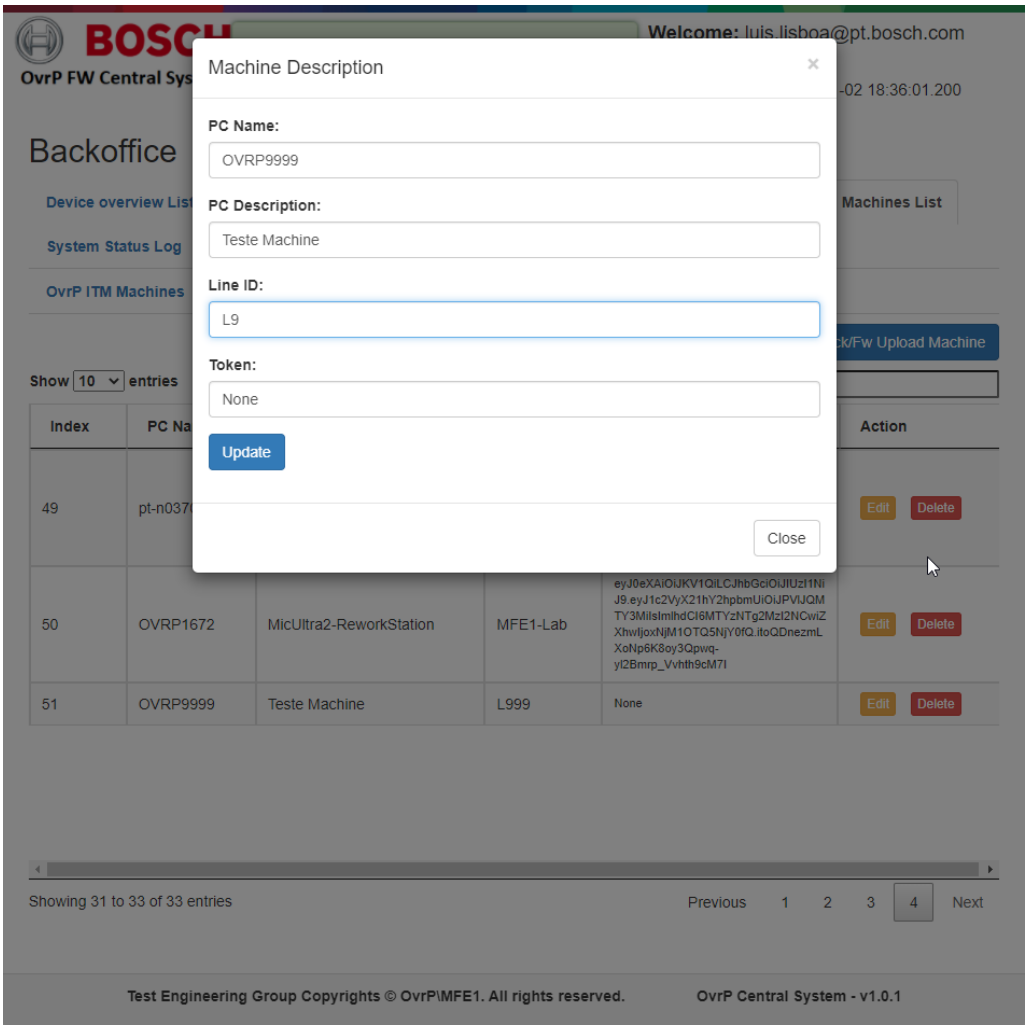


Figura 5.25: Teste Funcional Janela Modal Editar Máquina Não ITM

com recurso a comandos disponíveis no protocolo de comunicação do *firmware* da câmara. Após a câmara arrancar é feita a validação da diversidade através dos IDs únicos, sendo eles o `variant ID` e `product ID`. Validada esta parte, é então executado o VI que faz a verificação, o descarregamento e a atualização do *firmware* na câmara. No teste executado, foi utilizada uma câmara cujo o SAP é o `F.01U.365.453`. Esta câmara é produzida com a versão de *firmware* `84500710(7.10.0084)`, sendo que tanto o App1 como o App0 têm esta versão. A versão mais recente de *firmware* que está disponível no mercado através da `Bosch Download Store` é a `18010760(7.60.0118)`. Quando o VI é executado, a primeira operação é consultar através da API REST qual a versão que o produto deve ter, através da rota `HTTP GET API/device/F.01U.365.453`. A API REST só responde se o posto de embalagem enviar como parâmetro do *header* um *to-*

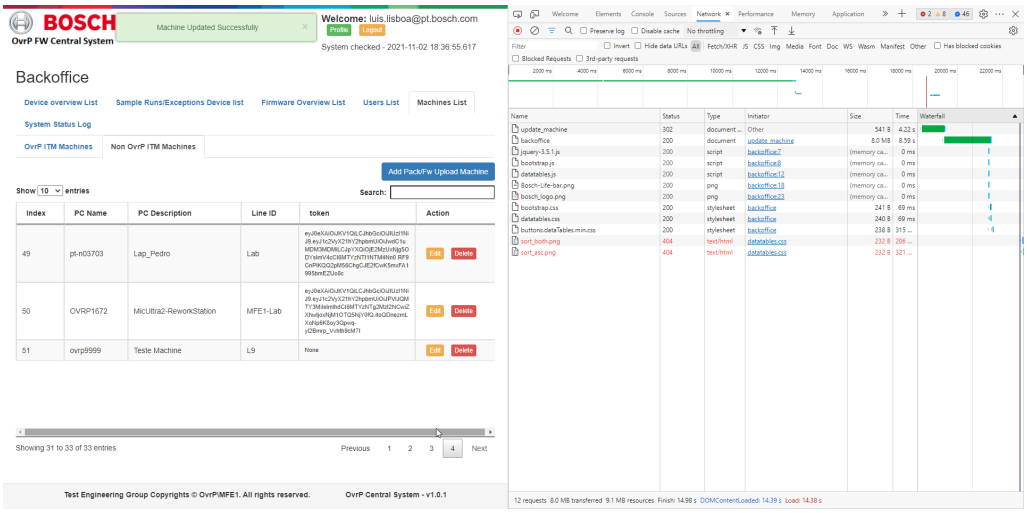


Figura 5.26: Resultado do Teste Funcional Janela Modal Adicionar Máquina Não ITM

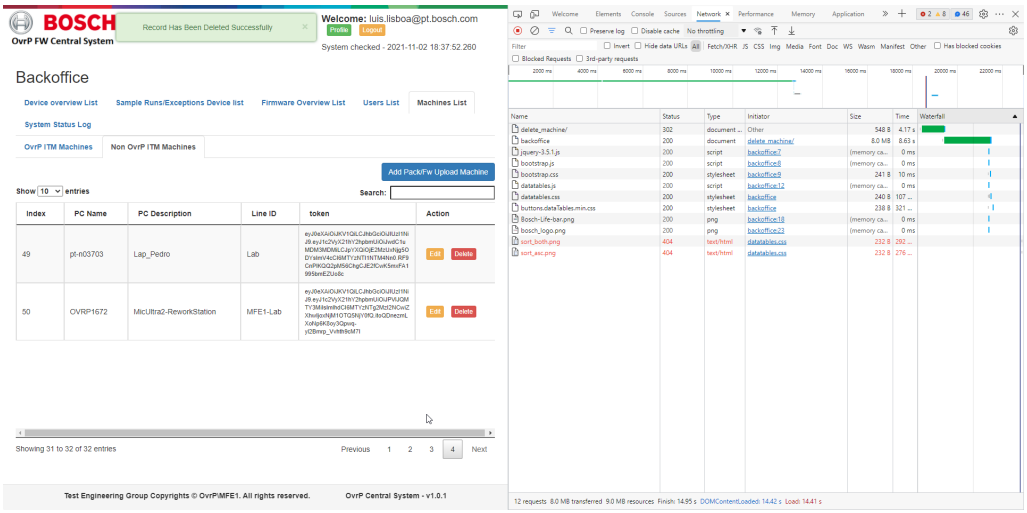


Figura 5.27: Resultado do Teste Funcional Janela Modal Remover Máquina Não ITM

ken válido. Depois de obter as versões das partes constituintes do *firmware*, o VI verifica se os ficheiros de *firmware* já estão disponíveis localmente. Se não estiverem, faz o descarregamento do que está em falta, se não, procede para a verificação de qual versão a câmara em questão tem instalada. Após verificar que não está concordante com a que está no serviço, procede à sua atualização. O interface do posto de embalagem com a indicação de que está a atualizar o `App1` está disponível na Figura 5.28

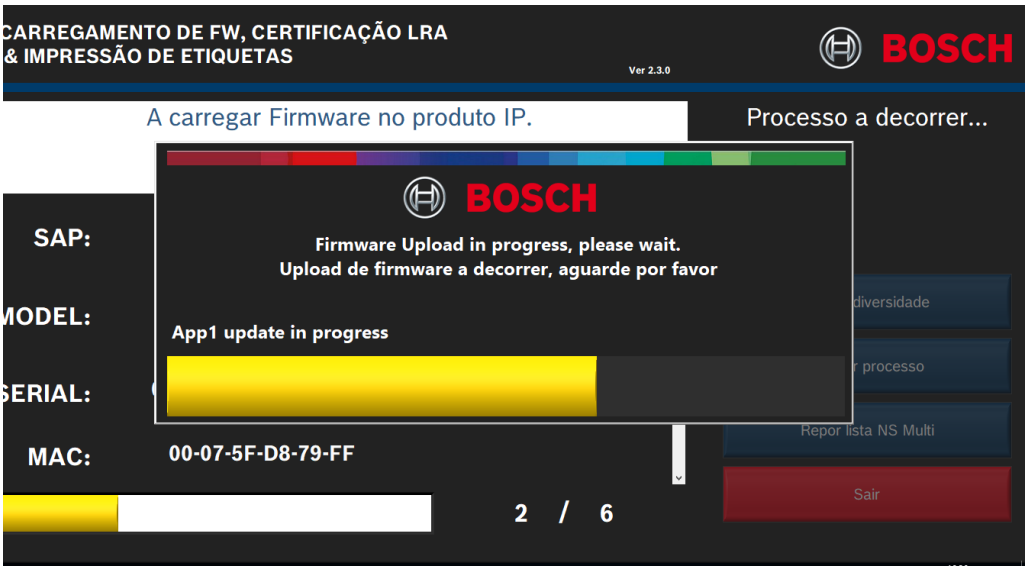


Figura 5.28: Teste Funcional Carregamento Automático de *Firmware App1* no Posto de Embalagem

Para este código, é possível consultar o *back-office* do serviço, para obter as versões corretas, sendo que para o código F.01U.365.453, as versões corretas estão na para consulta na Figura 5.29.

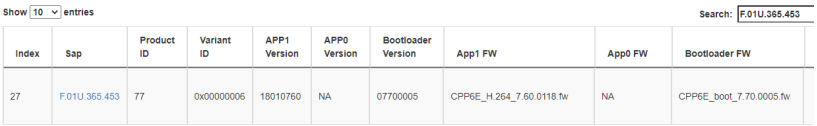


Figura 5.29: Consultar Dados no *Back-office* do Produto F.01U.365.453

É possível verificar que o `App0` não necessita de ser atualizado e o `Bootloader` já consta com a versão mais atual, ou seja, apenas é necessário atualizar o `App1`. Como já referido, o *software* do posto de embalagem guarda todos os dados do produto, como a versão de *firmware* com que o produto foi ligado e a versão com que o produto foi enviado para o mercado. Neste teste, a câmara atualizada entrou como já referido com a versão `84500710(7.10.0084)` e foi enviada com a versão `18010760(7.60.0118)`, onde apenas o `App1` foi atualizado. Na maioria dos casos, o `App0` e `Bootloader` não se atualizam, quando existe uma versão nova e afeta alguns produtos é para corrigir algum bug, pois em 90 % dos casos apenas o `App1` é atualizado. A câmara utilizada para testar o *software* foi atualizada e a atualização registada na base dados, como se pode verificar na Figura 5.30. O *MAC Address* corresponde ao da Figura 5.28.

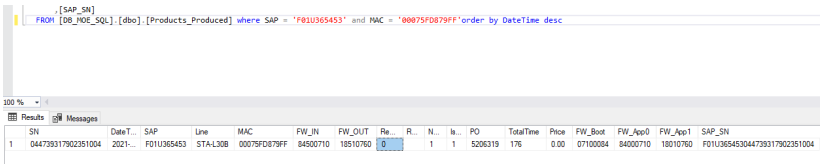


Figura 5.30: Consulta Registos na Base da Dados do Produto F.01U.365.453 Testado

5.1.3 Estação de Carregamento de Firmware

Esta estação permite ligar até quatro câmaras em paralelo, sendo um requisito devido ao estrangulamento de produção causado pelo posto de embalagem quando produz códigos que atualizam mais do que uma parte de *firmware*. Caso isso aconteça, a câmara obriga a um *reboot* após cada atualização, no caso de atualizar o `App1`, `App0` e `Bootloader`, significa que a câmara tem que fazer *reboot* três vezes. Desta forma, com recurso a esta estação de carregamento, é possível atualizar até 4 câmaras em simultâneo, sendo desta forma possível aumentar a capacidade de produção, pois se uma câmara demorar no total 400 s, quer dizer que o tempo de ciclo do posto é de 100 s, ou seja, é possível ligar uma câmara ao posto de embalagem a cada 100 s. Este posto de carregamento de *firmware* foi implementado nas linhas com os requisitos de produção mais elevados. Na Figura 5.31 temos a imagem da utilização de uma estação de carregamento. Esta estação é constituída por um PC com cinco portas de rede, sendo que uma das portas é para ligar à rede Bosch e as restantes são para ligar às várias câmaras. Cada placa de rede tem uma gama de IP distinta, que podem ser consultadas na Tabela 5.1. O segundo IP é utilizado para detetar a câmara quando se liga

Tabela 5.1: Gamas de IP da Estação de Carregamento de *Firmware*

Placa de rede	IP Address 1	IP Address 2
Baía 1	192.168.0.100/255.255.255.0	169.254.0.200/255.255.0.0
Baía 2	192.168.1.100/255.255.255.0	169.254.1.200/255.255.0.0
Baía 3	192.168.2.100/255.255.255.0	169.254.2.200/255.255.0.0
Baía 4	192.168.3.100/255.255.255.0	169.254.3.200/255.255.0.0

a uma porta, pois o *firmware* da Bosch tem um algoritmo que automaticamente calcula um IP com base no número de série e `MAC Address`, sendo que os IPs estão sempre na gama 169.254.*.*. Quando se ligou as câmaras às baías, após serem detetadas, o que o *software* desenvolvido faz é atribuir o IP a cada câmara dependendo da baía em que está ligado, ou seja, na baía 1 é atribuído o IP 192.168.0.1, e a mesma lógica para as restantes baías. Isto é necessário, pois esta estação para além do carregamento de *firmware*, também se implementou o carregamento dos certificados e respetiva verificação. Uma vez que para carregar os certificados, é enviado um comando à câmara, que faz com que a mesma faça um

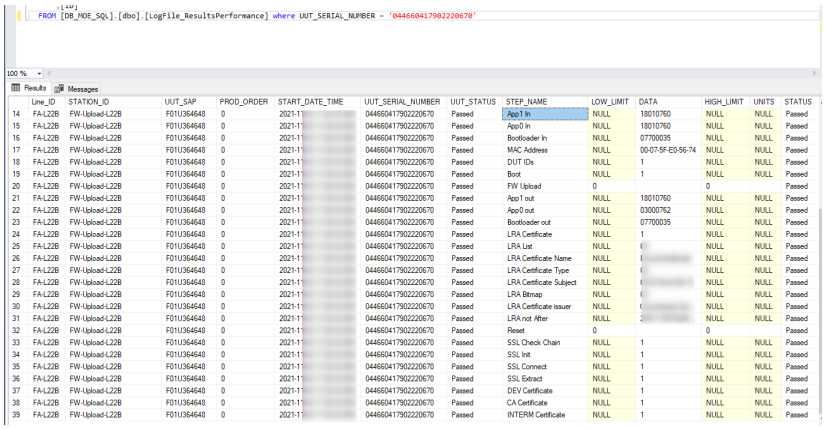
pedido de certificado a um serviço central da Bosch, para que o certificado seja devolvido para a câmara correta, é necessário utilizar um *reverse proxy*, sendo que neste caso foi utilizado o nginx [21]. Para o nginx funcionar corretamente, é necessário que o IP da câmara pertença à gama de IP da placa de rede, caso contrário falha sempre neste passo, como se verificou durante os testes. Desta forma, foram instalados com recurso ao nssm [22], quatro serviços de nginx, um para cada baía, com as configurações referentes a cada uma das gamas de IP. Desta forma é possível utilizar quatro instâncias do nginx a correr em paralelo na mesma máquina, permitindo que cada câmara possa fazer o pedido e a instalação dos certificados.

Figura 5.31: Estação de Carregamento de *Firmware*

Após carregar os certificados, inicia-se o carregamento de *firmware*, utilizando o mesmo VI do posto de embalagem. Apenas nesta situação, a versão de *firmware* é verificada quando se faz *setup* ao teste através do cartão **kanban**, e durante a produção este não verifica se existem ou não atualizações. Só quando se volta a fazer novo *setup* é que a estação de carregamento volta a verificar. Isto foi definido assim, para evitar possíveis problemas durante a atualização, como parar a meio de uma atualização ou até mesmo iniciar uma nova atualização enquanto o ficheiro está a ser descarregado a API REST e desta forma poder corromper a câmara, obrigando a rejeitar a mesma. Uma vez que as ordens de produção nunca são elevadas, ou seja, uma ordem em média tem 24 câmaras, o pior dos casos é atrasar atualização durante 1 h. Para o teste em questão, apenas o `App0` é atualizado, passando de `18010760(7.60.0118)` para a versão `03000762(7.62.0003)`. Depois de atualizar corretamente, o passo seguinte é definir os parâmetros de fábrica e posteriormente validar os certificados instalados. Todos os passos foram validados com sucesso, como se pode verificar na Figura 5.32. A estação regista todos os passos na base de dados e todos os registos estão associados ao número de série da câmara.

Figura 5.32: Resultado do Teste à Estação de Carregamento de *Firmware* do `App0` Resultado

Na base de dados estão visíveis todos os passos efetuados e as respetivas versões com que cada parte de *firmware* entra no posto e a versão com que é enviado para posto seguinte. O registo na base de dados da câmara com o número de série 044660417902220670 está na Figura 5.33 e o registo do número de série 044660417902220745 está representado na Figura 5.34.

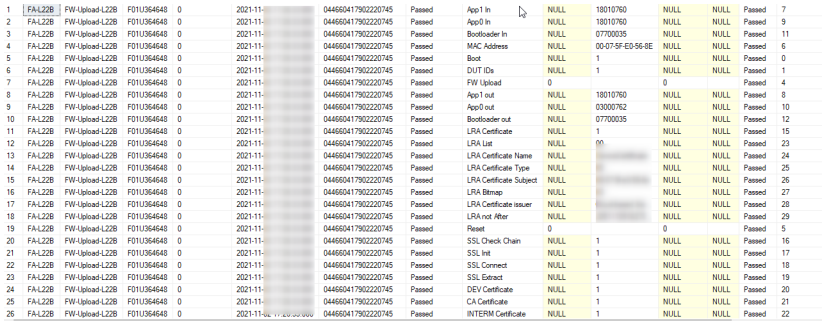


```
FROM [DB_HOE_SQL].[dbo].[LogFile_ResultsPerformance] where UUT_SERIAL_NUMBER = '04466041790222067O'
```

Results | Messages

	Line_ID	STATION_ID	UUT_SAP	PROD_ORDER	START_DATE_TIME	UUT_SERIAL_NUMBER	UUT_STATUS	STEP_NAME	LOW_LIMIT	DATA	HIGH_LIMIT	UNITS	STATUS
14	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	App1 In	NULL	18010760	NULL	NULL	Passed
15	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	App0 In	NULL	18010760	NULL	NULL	Passed
16	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	Bootloader In	NULL	07700035	NULL	NULL	Passed
17	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	MAC Address	NULL	00-07-5F-ED-56-74	NULL	NULL	Passed
18	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	DUT IDs	NULL	1	NULL	NULL	Passed
19	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	Boot	NULL	1	NULL	NULL	Passed
20	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	FW Upload	0		0		Passed
21	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	App1 out	NULL	18010760	NULL	NULL	Passed
22	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	App0 out	NULL	03000762	NULL	NULL	Passed
23	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	Bootloader out	NULL	07700035	NULL	NULL	Passed
24	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	LRA Certificate	NULL	1	NULL	NULL	Passed
25	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	LRA List	NULL	0	NULL	NULL	Passed
26	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	LRA Certificate Name	NULL	[redacted]	NULL	NULL	Passed
27	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	LRA Certificate Type	NULL	0	NULL	NULL	Passed
28	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	LRA Certificate Subject	NULL	[redacted]	NULL	NULL	Passed
29	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	LRA Bitmap	NULL	0	NULL	NULL	Passed
30	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	LRA Certificate issuer	NULL	[redacted]	NULL	NULL	Passed
31	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	LRA not After	NULL	[redacted]	NULL	NULL	Passed
32	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	Reset	0		0		Passed
33	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	SSL Check Chain	NULL	1	NULL	NULL	Passed
34	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	SSL Init	NULL	1	NULL	NULL	Passed
35	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	SSL Connect	NULL	1	NULL	NULL	Passed
36	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	SSL Extract	NULL	1	NULL	NULL	Passed
37	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	DEV Certificate	NULL	1	NULL	NULL	Passed
38	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	CA Certificate	NULL	1	NULL	NULL	Passed
39	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222067O	Passed	INTERM Certificate	NULL	1	NULL	NULL	Passed

Figura 5.33: Resultado da Estação de Carregamento de Firmware para o Número de Série 044660417902220670



1	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	App1 In	NULL	18010760	NULL	NULL	Passed	7
2	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	App0 In	NULL	18010760	NULL	NULL	Passed	9
3	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	Bootloader In	NULL	07700035	NULL	NULL	Passed	11
4	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	MAC Address	NULL	00-07-5F-ED-56-8E	NULL	NULL	Passed	6
5	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	Boot	NULL	1	NULL	NULL	Passed	0
6	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	DUT IDs	NULL	1	NULL	NULL	Passed	1
7	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	FW Upload	0		0		Passed	4
8	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	App1 out	NULL	18010760	NULL	NULL	Passed	8
9	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	App0 out	NULL	03000762	NULL	NULL	Passed	10
10	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	Bootloader out	NULL	07700035	NULL	NULL	Passed	12
11	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	LRA Certificate	NULL	1	NULL	NULL	Passed	15
12	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	LRA List	NULL	00	NULL	NULL	Passed	23
13	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	LRA Certificate Name	NULL	[redacted]	NULL	NULL	Passed	24
14	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	LRA Certificate Type	NULL	[redacted]	NULL	NULL	Passed	25
15	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	LRA Certificate Subject	NULL	[redacted]	NULL	NULL	Passed	26
16	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	LRA Bitmap	NULL	[redacted]	NULL	NULL	Passed	27
17	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	LRA Certificate issuer	NULL	[redacted]	NULL	NULL	Passed	28
18	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	LRA not After	NULL	[redacted]	NULL	NULL	Passed	29
19	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	Reset	0		0		Passed	5
20	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	SSL Check Chain	NULL	1	NULL	NULL	Passed	16
21	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	SSL Init	NULL	1	NULL	NULL	Passed	17
22	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	SSL Connect	NULL	1	NULL	NULL	Passed	18
23	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	SSL Extract	NULL	1	NULL	NULL	Passed	19
24	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	DEV Certificate	NULL	1	NULL	NULL	Passed	20
25	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	CA Certificate	NULL	1	NULL	NULL	Passed	21
26	FA-L22B	FW-Upload-L22B	F01U364648	0	2021-11-	04466041790222074S	Passed	INTERM Certificate	NULL	1	NULL	NULL	Passed	22

Figura 5.34: Resultado da Estação de Carregamento de Firmware para o Número de Série 044660417902220745

5.2 Testes de Desempenho

Nesta secção apresentam-se os testes não funcionais executados na API desenvolvida, que se dividem em testes de latência e de carga.

5.2.1 Latência

Os testes de latência à API REST foram executados com recurso ao *software* Postman, que permite a quem desenvolve interagir com a API e criar uma diversificada variedade de testes.

5.2.1.1 API REST

Para testar a latência, foi decidido utilizar a rota `.../device`, e para além do testes de latência também são verificados os parâmetros do *header* e do *body*,

nomeadamente o `HTTP Status Code` e o conteúdo JSON da reposta.

O teste consistiu numa série de dez pedidos para os métodos `HTTP GET` e `PUT` e cinco para os métodos `HTTP POST` e `DELETE`. Os resultados dos testes estão compilados na Tabela 5.2.

Tabela 5.2: Teste Latência à Rota *Device*

Teste de Latência à rota *Device*				
Pedido				
HTTP	GET	POST	PUT	DELETE
URI	/device/F.01U.321.597	/device	/device/450	/device/455...460
Header	x-access-token : "...." Content-Type: "application/json" U:"luislisboa@pt.bosch.com.com"			
Resposta				
HTTP Status Code	200	200	200	200
Verificar Body JSON	OK	OK	OK	OK
Latência média (ms)	36,2	58,2	52,2	36,8
Latência mínima (ms)	14,0	31,0	26,0	17,0
Latência máxima (ms)	111,0	91,0	84,0	93,0
Desvio Padrão	34,2	21,7	19,9	40.0

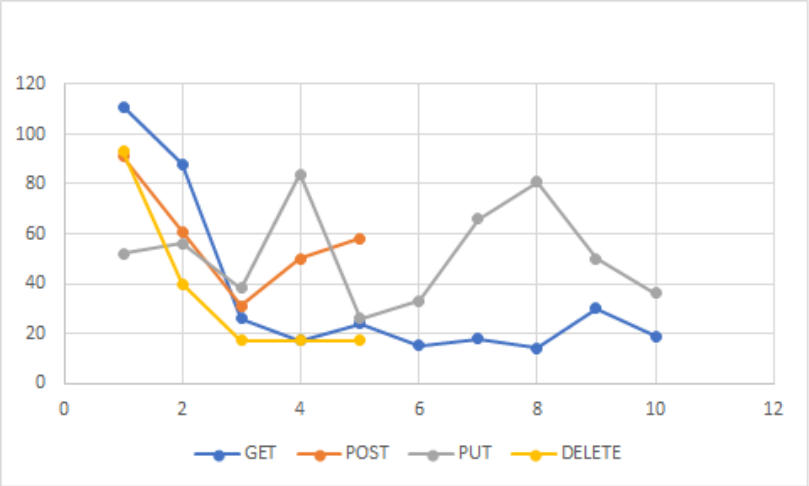


Figura 5.35: Dispersão Teste de Latência à Rota `Device`

Os pedidos cuja latência é superior, corresponde sempre ao primeiro pedido HTTP de cada um dos testes executados. Nos pedidos seguintes a latência é baixa e constante. Como se pode verificar pela tabela, a latência é sempre muito inferior a 200 ms, sendo que em média, os pedidos andam na ordem dos 50 ms. Pode concluir-se que a infraestrutura está preparada para conseguir gerir uma elevada quantidade de pedidos. Em anexo, estão as Figuras D.1, D.2, D.3, D.4, com os resultados obtidos através dos testes executados no Postman. Quanto ao desvio padrão, pode concluir-se que para os pedidos `HTTP PUT` e `POST` o valor é próximo de zero, logo está a dizer-nos que existe pouca dispersão em relação

à latência obtida em cada pedido. Nos restantes, este valor é superior e pode concluir-se que a dispersão é maior, existindo uma diferença significativa dos tempos de resposta de cada pedido `HTTP` executado nestes testes. Na Figura 5.35 é possível consultar o gráfico com a dispersão de cada pedido.

5.2.1.2 Back-office

Os testes de latência realizados ao *back-office* foram realizados também com o Postman. Os testes consistiram em fazer 10 pedidos de *Sign In*, através do URL `http://...:6002` e 10 pedidos para forçar a verificação de atualizações de *firmware* através do URL `http://...:6002/update_pack_version_table`. Para fazer autenticação o pedido é um `HTTP POST` e para forçar uma verificação é `HTTP GET`. Os resultados obtidos estão visíveis na Tabela 5.3. Para autenticar na página de *Sign In*, o Postman permite enviar dados de formulário no pedido. Para o restante, é utilizado o *session cookie* gerado no *Sign In*.

Tabela 5.3: Teste Latência ao *Back-office*

Teste de Latência ao Back-office		
Pedido		
HTTP	GET	POST
URL	...:6002/	...:6002/backoffice
Resposta		
Latência média (ms)	670.4	600.6
Latência mínima (ms)	487.0	474.0
Latência máxima (ms)	824.0	713.0
Desvio Padrão	112.0	64.0

Os valores de latência obtidos no teste de *Sign In* não são próximos de zero, pois o teste verifica os dados de acesso do utilizador e devolve a página *Web* do *back-office*, na qual constam muitos dados e para os obter são realizadas várias *queries* à base de dados. A quantidade de dados recebidos em cada pedido é cerca de 8 MB. O que se pode concluir é que para a quantidade de dados processados neste pedido, o valor da latência é aceitável. O desvio padrão mostra que o tempo de resposta aos pedidos tem alguma variação. Quanto aos dados do teste de forçar a verificação de *firmware*, seguem o mesmo padrão do teste anterior, no entanto os valores obtidos são ligeiramente melhores, pois como se pode confirmar, o desvio padrão é metade do obtido no primeiro teste. Na Figura 5.36, é possível verificar a dispersão dos dois testes realizados. No Anexo D.2 podem ser consultadas os resultados dos testes executados no Postman.

5.2.2 Carga

Os testes de carga foram executados com o propósito de verificar o comportamento da API e do *back-office* face a uma grande quantidade de pedidos. Para tal, foi utilizado o *software* Apache JMeter [23].

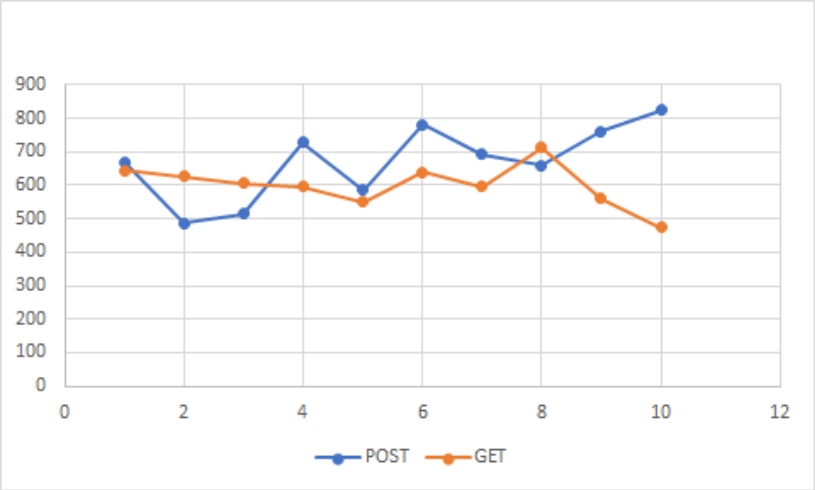


Figura 5.36: Dispersão dos dados obtidos nos Testes de Latência do *Back-office*

5.2.2.1 API REST

A Tabela 5.4 contém os valores da média e desvio padrão obtidos no teste de carga efectuado à API REST.

Tabela 5.4: Teste Carga API REST

Teste de Carga API REST		
Amostras/Segundo	35	
HTTP	GET	
URL	.../device/F.01U.321.595	
Resultado		
	Média	Desvio Padrão
Latência (ms)	20,4	18,44
Tempo da Amostra (ms)	20,3	18,55
Tempo da Ligação (ms)	4,9	0,60
Throughput	1745,6 pedido/min	

Na Figura 5.37 pode verificar-se a dispersão dos resultados obtidos.

O teste consistiu em simular 35 pedidos `HTTP GET` à API REST durante um período de 1 s, ou seja, 35 pedidos por segundo. Foi definido este valor pois é o número máximo de acessos expectável dado o número de utilizadores registados e o número de máquinas adicionadas ao serviço. Do resultado, pode concluir-se que a latência media do teste corresponde a 20 ms e do desvio padrão calculado é 18. Uma vez que para processar o pedido a API tem que consultar várias tabelas da base dados e fazer o `JOIN` entre elas para tratar os dados e responder ao pedido, logo o valor da latência obtido é excelente e significa que a API consegue responder e processar muitos pedidos por minuto. O *throughput* com o valor de 1745.6 pedidos por minuto, está a comprovar que a API tem uma performance muito superior comparativamente com os pedidos que são expectáveis existirem

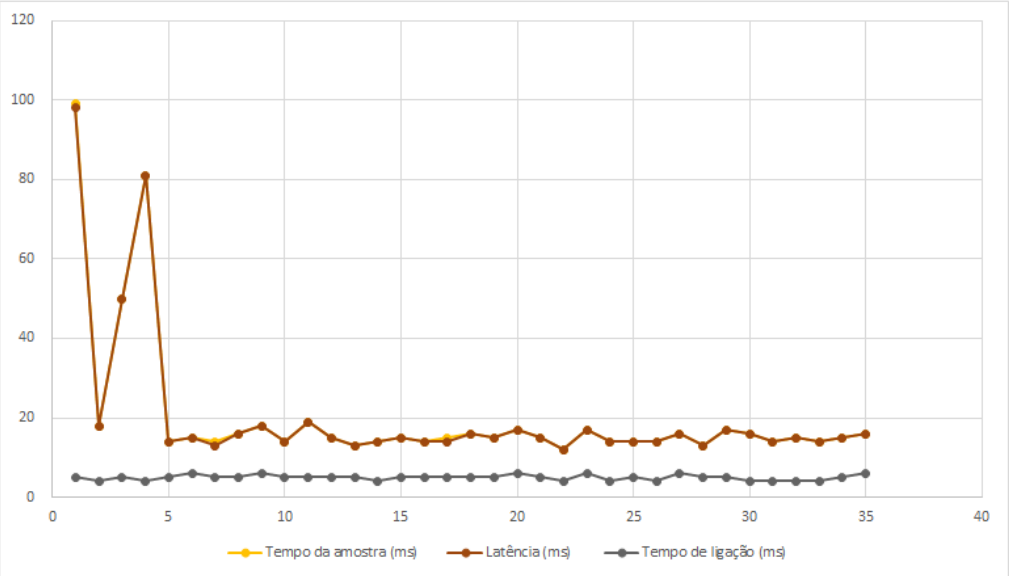


Figura 5.37: Teste de Carga `HTTP GET` à rota `Device` com 35 Pedidos

em simultâneo. Por forma a garantir que a performance não é afetada significativamente quando o número de pedidos aumenta, foi decidido fazer um teste a simular uma situação extrema, que consistiu em simular 250 acessos durante 1 s. Este é o número máximo de *threads* definidas na configuração do `Waitress` da API REST. Pode verificar-se na Tabela 5.5, que o tempo médio correspondente à latência e o desvio padrão aumentaram significativamente, sendo que a latência passou de 20 ms para 850.1 ms e o desvio padrão de 18 para 207.2. No entanto, dada a complexidade do processamento do pedido, estes valores são aceitáveis. O *throughput* é o dobro comparativamente com o número de pedidos efetuados, logo, o serviço consegue responder de forma positiva mesmo quando colocado numa situação extrema e improvável.

Tabela 5.5: Teste Carga 2 API REST

Teste de Carga 2 API REST		
Amostras/Segundo	250	
HTTP	GET	
URL	.../device/F.01U.321.595	
Resultado		
	Média	Desvio Padrão
Latência (ms)	850,1	207,2
Tempo da Amostra (ms)	850,1	207,2
Tempo da Ligação (ms)	4,0	1,6
Throughput	596,5 pedido/min	

Na Figura 5.38, está disponível o gráfico da dispersão das amostras referentes a este pedido. É possível verificar que a tendência do valor da latência conso-

ante o aumento do número de amostras é subir mais ou menos de forma linear, comportamento expectável para este tipo de situação.

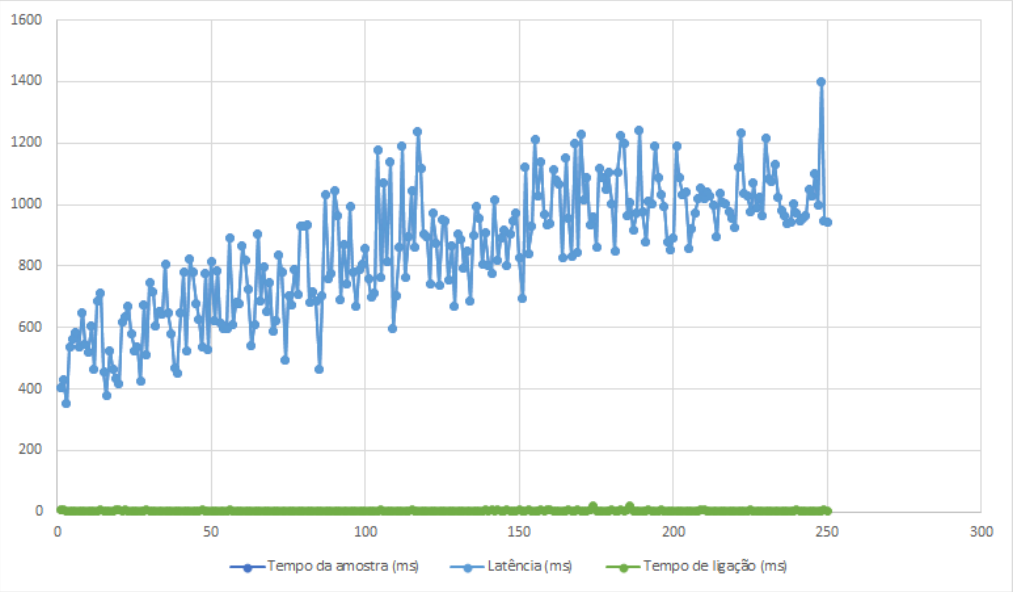


Figura 5.38: Teste de Carga `HTTP GET` à rota `Device` com 250 Pedidos

5.2.2.2 Back-office

Quanto ao Teste de Carga realizado ao *back-office*, consistiu na simulação de 10 pedidos por segundo, ou seja, dado o número de utilizadores registados no sistema, no máximo por segundo podem existir 20 pedidos. O resultado deste teste está visível na Tabela 5.6.

Tabela 5.6: Teste Carga *Back-Office*

Teste de Carga Back-Office		
Amostras/Segundo	10	
URL	.../backoffice	
Resultado		
	Média	Desvio Padrão
Latência (ms)	628,5	643,5
Throughput	212,6 pedido/min	

A latência média de resposta aos pedidos é de 628.5 ms e o desvio padrão é 643.5. São valores elevados, mas este teste consiste em fazer *Sign In* e carregar a página do *back-office* com os dados de todas as tabelas, logo é uma quantidade elevada de dados. O valor do desvio padrão indica que existe uma grande variação entre o processamento de cada pedido, o que pode sugerir alguma instabilidade e baixa performance. Através do *throughput* verificamos que para esta situação

o *back-office* consegue responder a 212.6 pedidos por minuto. Na Figura 5.39 é possível verificar a dispersão entre cada pedido.

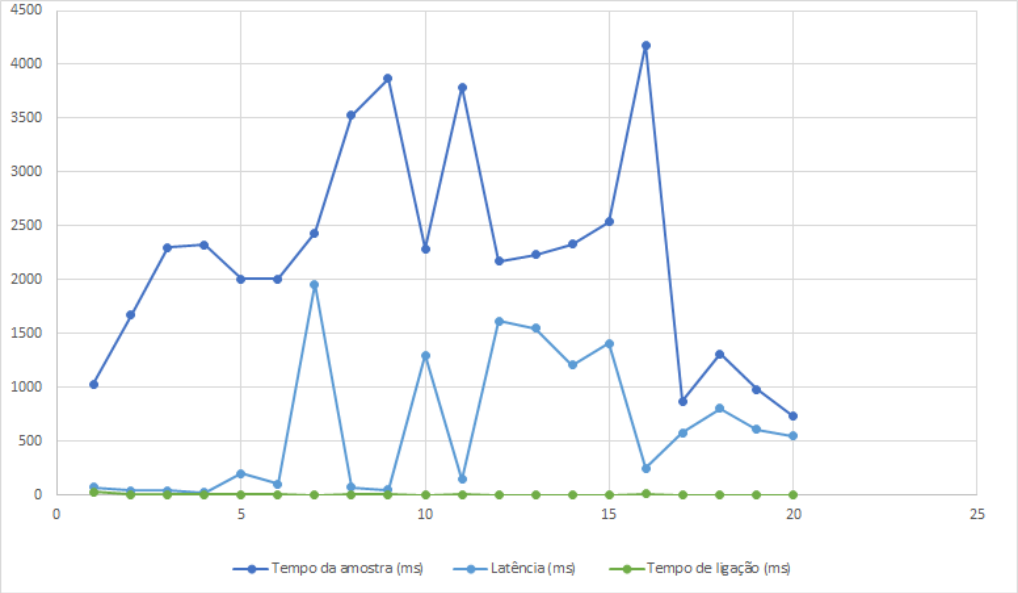


Figura 5.39: Teste de Carga *Back-office*

Para colocar *back-office* à prova, foi executado um teste que consistiu em fazer 25 pedidos, com tempo inicial igual a 5, dando desta forma 5 pedidos por segundo, durante 5 s, O resultado deste teste está disponível na Tabela 5.7.

Tabela 5.7: Teste Carga 2 *Back-Office*

Teste de Carga Back-Office		
Amostras/Segundo	25	
URL	.../backoffice	
Resultado		
	Média	Desvio Padrão
Latência (ms)	945,1	1495,4
Throughput	105,9 pedido/min	

É possível verificar que a performance baixou, sendo que o *throughput* passou para menos de metade. Desta forma pode concluir-se que o *back-office* consegue gerir a quantidade de acessos para os utilizadores existentes, sendo que apenas em casos mais extremos e pouco prováveis é que o sistema pode ficar comprometido. Na Figura 5.40, é possível verificar a dispersão dos dados do teste de carga realizado

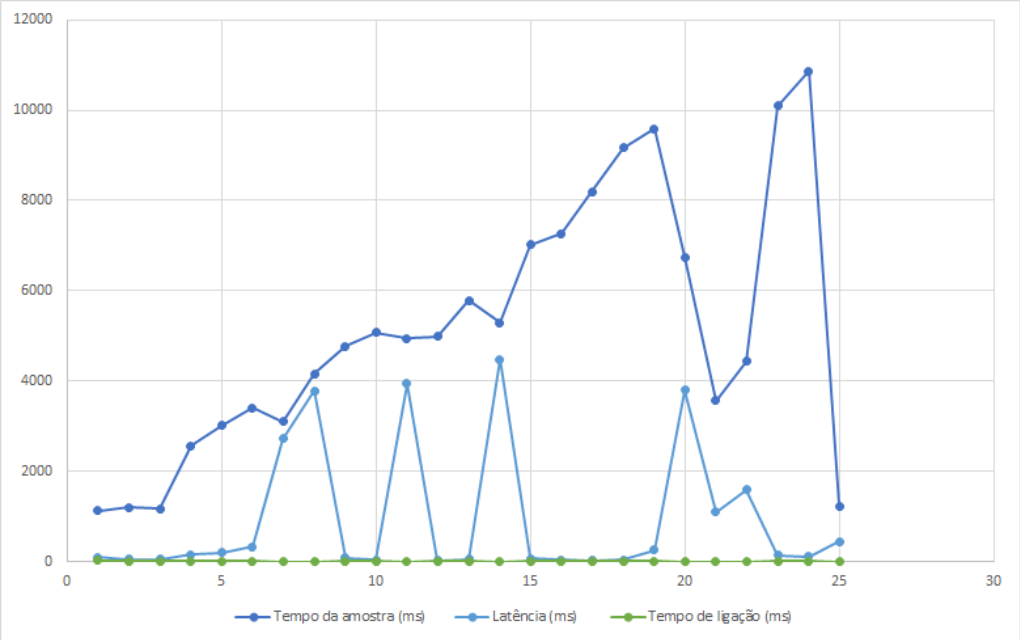


Figura 5.40: Teste de Carga *Back-office* 25 *Threads*

5.3 Usabilidade e Impacto

Após a disponibilização em ambiente de produção, foi elaborado um formulário com sete perguntas, referentes à usabilidade e impacto da solução desenvolvida. Apresentam-se de seguida as perguntas e respostas obtidas.

1. O *back-office* tem um interface intuitivo para o utilizador?
   - 100 % - Concordo Totalmente
2. No geral, o *back-office* possui todas as funcionalidades necessárias para a gestão do sistema?
   - 100 % - Concordo Totalmente
3. Como classifica a pesquisa de informação no *back-office*?
   - 83.3 % - Muito Boa
   - 16.7 % - Boa
4. No geral, como classifica o impacto que teve a automatização de todo o processo de carregamento de *firmware*?
   - 100 % - Muito Positivo

5. No geral, o sistema conseguiu resolver as lacunas existentes no sistema antigo, para a introdução de novos *firmware* em produção?
    - 100 % - Conseguiu Totalmente
6. Antes da implementação do novo sistema, qual era o número de horas despendidas por semana na implementação de novos *firmware* na produção/-posto de embalagem?
    - 3 h - 16.7 %
    - 4 h - 66.6 %
    - 5 h - 16.7 %
7. Após a implementação do novo sistema, qual o número de horas despendidas por semana na implementação de novos *firmware* na produção (posto de embalagem/ estação de carregamento)?
    - 0 h - 100 %

Ao formulário responderam 6 colaboradores da Bosch Security Systems que são utilizadores regulares do *back-office*. Pela análise das respostas, conclui-se que a solução cumpre os objetivos. No Anexo E pode ser consultado o inquérito realizado através do Google Forms.

5.4 Sumário

Os testes funcionais permitiram verificar o correto funcionamento da solução. A possibilidade de atualização das versões de *firmware* do posto de embalagem durante a produção e a criação da estação de carregamento de *firmware* tiveram um impacto positivo na capacidade de produção horária de algumas linhas, ultrapassando o *bottleneck* pré-existente. O *back-office* de gestão do serviço demonstrou a eficácia e correto funcionamento das funcionalidades implementadas.

Os testes de latência da API REST mostraram uma latência baixa, indicando um bom desempenho. Os testes de carga da API REST em funcionamento normal, ou seja, para 35 por minuto, baseado no número de máquinas e utilizadores existentes, revelou uma latência média baixa e um desvio padrão baixo. No caso extremo, 250 pedidos num segundo, o desempenho baixou, com o aumento da latência média e o desvio padrão. Apesar da complexidade dos pedidos, a API processou todos com sucesso. Quanto aos resultados obtidos nos testes realizados ao *back-office*, relativamente aos testes de latência, os valores obtidos são muito superiores aos da API REST, no entanto os resultados são aceitáveis, uma vez que todos os pedidos foram processados corretamente. No teste de carga, para o funcionamento normal, o *back-office* teve um resultado aceitável, contudo a

performance é baixa, dada a instabilidade do tempo de processamento de cada pedido, como se comprova pelo desvio padrão obtido. Para o caso extremo, os valores de latência aumentaram e o *Throughput* baixou, no entanto todos os pedidos foram processados. Por último, a análise das respostas ao inquérito distribuído entre os utilizadores permitiu concluir que os respondentes atribuem elevada utilidade e usabilidade à solução.

Capítulo 6

Conclusões

Esta dissertação de mestrado teve como objetivo desenvolver um serviço para atualização e gestão automática de firmware para câmaras de videovigilância em shop floor.

6.1 Resultados

Com este projeto foi possível automatizar o processo de atualização após a industrialização, garantindo o carregamento das versões corretas e mais recentes de *firmware* para cada produto. Com a implementação da API REST, definiu-se uma interface comum a todos os pedidos dos Postos de Embalagem, Estação de Carregamento e *Back-office*. Assim, a interação com a base de dados é sempre processada de igual forma e apenas os administradores podem alterar a base de dados. Para além de rotas dedicadas à fase de produção, foram igualmente implementadas na API rotas para a fase de industrialização e para forçar o carregamento de versões específicas em produtos. Esta opção permite responder a requisitos de produção especificados por clientes.

O *software* criado em LabVIEW para o carregamento do *firmware* através da API REST foi validado e integrado no Posto de Embalagem e na Estação de Carregamento de *firmware*. A atualização de versões de *firmware* no Posto de Embalagem é totalmente automática. Desde a implementação em produção, o responsável não voltou a fazer qualquer introdução manual de *firmware*. A Estação de Carregamento de *Firmware* ultrapassou as limitações de produção do posto de embalagem, em linhas de produção com requisitos elevadas. Para além de fazer as atualizações de *firmware*, também realiza o carregamento e verificação de certificados. Assim, nestas linhas a única função do posto de embalagem é imprimir as etiquetas de produto ou caixa. Com esta solução, foi possível

aumentar significativamente a produção diária.

Com o inquérito realizado, verificou-se que a solução cumpre os requisitos iniciais, é fácil de utilizar e impacta positivamente a produção.

6.2 Sugestões

Uma das fragilidades do *back-office*,centra-se no facto de este recarregar integralmente a página do *front-end* sempre que se realiza uma operação. Indiretamente aumenta-se o número de ligações à API REST, o tráfego e a carga de dados na rede. Uma sugestão seria utilizar um *framework* de refrescamento assíncrono da página, como o Django [24], que é direccionado para a criação de serviços de *back-office*, com uma boa interatividade entre o *front-end* e o *back-end*.

Bibliografia

[1] F. Community, "Http requests with the httpcaller." https://community.safe.com/s/article/HTTP-Requests-With-The-HTTPCaller/, 2021. [citado na p. vii, 8]

[2] Oracle, "The Java EE 5 Tutorial." https://docs.oracle.com/cd/E19879-01/819-3669/bnbhj/index.html, 2010. [citado na p. vii, 10]

[3] Comtech, "SOAP vs REST." http://comtech2.com/web-services-architecture-when-to-use-soap-vs-rest2/. [citado na p. vii, 20]

[4] AUVESY GmbH, "Automotive manufacturer speeds up S7 firmware updates with versiondog factory floor status." https://auvesy.com/en/case-studies-factory-floor-status-auvesy, 2019. [citado na p. 5]

[5] H.-Y. Paik, A. L. Lemos, M. C. Barukh, B. Benatallah, and A. Natarajan, *Web Service Implementation and Composition Techniques*. Springer, 2017. [citado na p. 7, 10]

[6] Service Ojects, "Why REST is so popular." https://www.serviceobjects.com/resources/articles-whitepapers/why-rest-popular. [citado na p. 8]

[7] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web services description language (WSDL) 1.1." https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.460.467&rep=rep1&type=pdf, 2021. [citado na p. 10]

[8] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the web services: an introduction to soap, wsdl, and uddi." https://ieeexplore.ieee.org/abstract/document/991449, 2002. [citado na p. 11]

[9] R. Bitter, T. Mohiuddin, and M. Nawrocki, *LabVIEW: Advanced Programming Techniques, Second Edition*. CRC Press, 2007. [citado na p. 14]

[10] M. Lutz and D. Ascher, *Learning Python, Second Edition*. O'Reilly Media, 2003. [citado na p. 15]

[11] Pallets, "Jinja." `https://jinja.palletsprojects.com/en/3.0.x/`, 2021. [citado na p. 16]

[12] B. Mehta and M. Kalali, *Developing RESTful Services with JAX-RS 2.0, WebSockets, and JSON: A complete and practical guide to building RESTful Web Services with the latest Java EE7 API*. Packt Publishing, 2013. [citado na p. 16]

[13] P. S. Foundation, "Pep 3333 – python web server gateway interface v1.0.1." `https://www.python.org/dev/peps/pep-3333/`. [citado na p. 16]

[14] M. Makai, "Flask - full stack python." `https://www.fullstackpython.com/flask.html`, 2021. [citado na p. 16]

[15] B. C. Rocha, "What the flask? pt-1 introdução ao desenvolvimento web com python." `http://pythonclub.com.br/what-the-flask-pt-1-introducao-ao-desenvolvimento-web-com-python.html`, 2014. [citado na p. 16]

[16] J. Myers and R. Copeland, *Essential SQLAlchemy, 2nd Edition: Mapping Python to Databases*. O'Reilly Media, 2015. [citado na p. 17]

[17] A. Becker, "Heidisql." `https://www.heidisql.com/`, 2020. [citado na p. 19]

[18] JWT, "Json web tokens." `https://jwt.io/`, 2021. [citado na p. 47]

[19] Internet Engineering Task Force, "JSON Web Token (JWT) - rfc7519." `https://tools.ietf.org/html/rfc7519`, 2020. [citado na p. 47]

[20] Flask, "The application context – flask documentation (2.0.x)." `https://flask.palletsprojects.com/en/2.0.x/appcontext/`, 2010. [citado na p. 72]

[21] nginx, "Introducing nginx 1.18 and 1.19." `https://www.nginx.com/blog/nginx-1-18-1-19-released/`, 2021. [citado na p. 126]

[22] NSSM, "Nssm - the non-sucking service manager." `https://nssm.cc/`, 2020. [citado na p. 126]

[23] T. A. S. Foundation, "Apache jmeter." `https://jmeter.apache.org/`, 2021. [citado na p. 130]

[24] D. S. Foundation, "Django." `https://www.djangoproject.com/`, 2021. [citado na p. 140]

Anexos

Anexo A

Classes de Mapeamento das Tabelas da Base de Dados

A.1 Classe Version_Table

Para a tabela `version_table`, a classe criada é a `DB_VersionTable` e está representada no Excerto de Código A.1

Excerto de Código A.1 Classe DB_VersionTable

```
class DB_VersionTable(base):
    __tablename__ = ''version_Table''
    index = Column(Integer, primary_key=True)
    VARIANT_ID = Column(String(255))
    PRODUCT_ID = Column(String(255))
    SAP = Column(String(13))
    IDENTIFICATION = Column(String(255))
    idApp1 = Column(Integer, ForeignKey(App1.idApp1))
    idApp0 = Column(Integer, ForeignKey(App0.idApp0))
    idBootloader = Column(Integer, ForeignKey(Bootloader.idBootloader))
    __table_args__ = {''schema'': ''OvrP_CS''}
```

Esta tabela é uma réplica em termos de colunas do ficheiro *version_tabele.txt*. É este ficheiro que fornece ao sistema a relação entre produtos e versões de *firmware* a utilizar. De forma a não duplicar dados, uma vez que algumas famílias de produtos utilizam as mesmas versões de *firmware*, foi decidido separar o *firmware*, ou seja, os ficheiros `App1`, `App0` e `Bootloader` em tabelas separadas. Desta forma é necessário criar *foreignKeys* para ligar as várias linhas da `version_table` à respetiva versão de *firmware*. Quanto a variável `__tabele__`, faz a correspondência à tabela da base de dados. A variável `__table_args__` serve para selecionar o esquema da base de dados ao qual a tabela pertence.

A.2 Classe Firmware_in_Production

Excerto de Código A.2 Classe dbOvrpCS_Device_Fw_Production

```
class dbOvrpCS_Device_Fw_Production(Base):
    __tablename__ = ''Fw_In_Production"
    index = Column(Integer, primary_key=True)
    PRODUCT_ID = Column(String(255))
    VARIANT_ID = Column(String(255))
    SAP = Column(String(13))
    IDENTIFICATION = Column(String(255))
    idApp1 = Column(Integer, ForeignKey(App1.idApp1))
    idApp0 = Column(Integer, ForeignKey(App0.idApp0))
    idBootloader = Column(Integer, ForeignKey(Bootloader.idBootloader))
    History = Column(String(max))
    Manual_Mode = Column(SMALLINT)
    __table_args__ = {''schema": ''OvrP_CS"}
```

A classe correspondente à tabela `Fw_In_Production` está representada no Excerto de Código A.2. Esta tabela é uma réplica da tabela `version_table` à exceção de duas colunas, sendo estas a `History`, cuja a função é guardar todo o histórico de atualizações de um produto e a `Manual`, que tem como função retirar um determinado produto do processo de atualização automático de *firmware*. Esta tabela é atualizada após a atualização da tabela `version_table`. Desta forma garante-se que só se procede à atualização da informação referente à produção se a atualização da tabela version_table não retornar qualquer tipo de erro.

A.3 Classes App1, App0 e Bootloader

Uma vez que o *firmware* de uma câmara é composto por `Bootloader`, `App0` e `App1`, como já referido, e um produto pode ter como requisito uma versão distinta de cada parte do *firmware*, foram criadas três tabelas de forma a fazer uma gestão independente dos vários ficheiros de *firmware*. Para estas tabelas foram criadas uma *primary key* com auto-incremento, que serve para fazer a ligação às tabelas `verion_table` e `fw_in_production` via *foreign key*.

Excerto de Código A.3 Classe App0

```
class App1(Base):
        __tablename__ = ''app0"
    idApp0 = Column(Integer, primary_key=True)
    App0 = Column(String(255))
    App0Version = Column(String(255))
    Checksum = Column(String(255))
    Date = Column(DATETIME(255))
    __table_args__ = {''schema": ''OvrP_CS"}
```

Excerto de Código A.4 Classe App1

```
class App1(Base):
        __tablename__ = ``app1"
        idApp0 = Column(Integer, primary_key=True)
        App1 = Column(String(255))
        App1Version = Column(String(255))
        Checksum = Column(String(255))
        Date = Column(DATETIME(255))
        __table_args__ = {``schema": ``OvrP_CS"}
```

Excerto de Código A.5 Classe Bootloader

```
class Bootloader(Base):
        __tablename__ = ``bootloader"
        idBootloader = Column(Integer, primary_key=True)
        Bootloader = Column(String(255))
        BootloaderVersion = Column(String(255))
        Checksum = Column(String(255))
        Date = Column(DATETIME(255))
        __table_args__ = {``schema": ``OvrP_CS"}
```

Nos Excertos de Código A.3, A.4 e A.5 estão representadas as classes dos três tabelas da base de dados correspondentes às três partes de *firmware*. A diferença entre as tabelas reside apenas no nome das colunas, sendo que o `App1`, `App0` e `Bootloader` correspondem ao nome do ficheiro de *firmware*, as colunas `App1Version`, `App0Version` e `BootloaderVersion` corresponde à versão de cada *firmware*, o `checksum` corresponde ao calculo de *checksum* de cada ficheiro e a coluna `Date` tem como função gravar a data na qual o ficheiro foi introduzido no sistema.

A.4 Classes Users e UsersType

Neste sub-capitulo irá ser apresentado as tabelas que fazem a gestão de acessos de utilizadores ao *back-office* do sistema. Para estas tabelas foram criadas as classes `DB_BCS_Uers` `DB_BCS_User_Type`.

Excerto de Código A.6 Classe Users

```
class Users(Base):
    __tablename__ = "Users"
    idUser = Column(SMALLINT, primary_key=True)
    email = Column(String(255))
    password = Column(String(255))
    token = Column(String(max))
    idUserType = Column(SMALLINT, ForeignKey(UserType.idUserType))
    __table_args__ = {"schema": "OvrP_CS"}
```

No Excerto de Código A.6 está representada a classe `Users`. A tabela correspondente à classe tem como função permitir a gestão de todos os utilizadores do *back-office*. Nesta classe, a variável `idUserType` guarda o nível de acesso de cada utilizador. Os vários tipos de acessos estão definidos na tabela `User_Type`. Para ligar as duas tabelas foi criada uma relação `foreign key`.

Excerto de Código A.7 Classe UsersType

```
class UserType(Base):
    __tablename__ = "user_type"
    idUserType = Column(SMALLINT, primary_key=True)
    user_type = Column(String(50))
    admin = Column(SMALLINT)
    access = Column(SMALLINT)
    __table_args__ = {"schema": "OvrP_CS"}
```

A classe que faz a ligação à tabela `User_Type` pode ser concultada no Excerto de Código A.7. A ligação à tabela `Uers` é feita pelo `idUserType`. Nesta tabela é possível adicionar os tipos de utilizador, que para este serviço foram definidos apenas três, sendo eles, o `admin`, `viewer` e o `notify`.

A.5 Classe Pack_Machines

A classe `NonOvrpMachine` está a mapear a tabela `Pack_Machines`. Nesta tabela são adicionadas todas as máquinas de posto de embalagem e da estação de carregamento de *firmware*. De salientar que esta tabela é utilizada para adicionar máquinas que não tenham qualquer registo na lista de máquinas presentes na tabela `machine` do esquema de ITM, desta forma será utilizada para adicionar máquinas de outras fábricas, que tenham a necessidade de consumir o serviço.

Excerto de Código A.8 Classe NonOvrpMachine

```
class NonOvrpMachine(Base):
    __tablename__ = "Pack_Machines"
    index = Column(SMALLINT, primary_key=True)
    machine_name = Column(String(255))
    pc_description = Column(String(255))
    line = Column(String(255))
    token = Column(String(max))
    __table_args__ = {"schema": "OvrP_CS"}
```

No Excerto de Código A.8 é possível visualizar a classe que faz o mapeamento da tabela `Pack_Machines`. Os objetos `machine_name`, `pc_description` e `line`, são preenchidos quando se adiciona uma máquina nova. O objeto `token` é utilizado para guardar o referido *token*, quando este é pedido. Desta forma, caso o *token* não tenha expirado e o cliente pedir outro, a API irá retornar o *token* guardado neste objeto.

A.6 Classes Pack_Machines2 e Machine

A tabela `Machine`, mapeada na classe `dbITM_Machine`, é uma tabela controlada pelo responsável de ITM da fábrica da Bosch Ovar. Nesta tabela, estão adicionados todos os computadores existentes na produção. A classe `dbOvrpCS_Machines` contém a tabela `Pack_Machines2`. Esta tabela é utilizada para adicionar as máquinas com permissão para comunicar com a API. Estas máquinas são as que estão mapeadas na tabela `Machine`. Desta forma, não há duplicação de informação.

Excerto de Código A.9 Classe dbITM_Machine

```
class dbITM_Machine(Base):
    __tablename__ = "Machine"
    ID = Column(BIGINT, primary_key=True)
    CIname = Column(String(255))
    Status = Column(SMALLINT())
    Description = Column(String(100))
    Line = Column(String(255))
    ITMnetwork = Column(String(50))
    IP = Column(String(15))
    __table_args__ = {"schema": "ITM"}
```

A tabela `Machine` pertence ao esquema `ITM`, como se pode verificar no Excerto de Código A.9. Este esquema é gerido pelo responsável de IT da área de produção. Na tabela `Pack_MAchines`, foi adicionada uma *foreign key* à coluna `machine_id`. Esta por sua vez está ligada à coluna `ID` da tabela `Machine`. As colunas `token` e `token_status`, são utilizadas para armazenar o *token* e o estado do mesmo. Isto é possível verificar no Excerto de Código A.10

Excerto de Código A.10 Classe OvrpMachine

```
    __tablename__ = "Pack_Machines2"
    id = Column(SMALLINT, primary_key=True)
    machine_id = Column(BIGINT, ForeignKey(dbITM_Machine.ID))
    token = Column(String(max))
    token_status = Column(BINARY)
    __table_args__ = {"schema": "OvrP_CS"}
```

A.7 Classes OvrP_CS_Log e Log_Status

Os dados referentes às transacções mais importantes são guardados na tabela `OvrP_CS_Log`. Desta forma, é possível saber a cause de problemas que possam surgir durante a actualização automática do sistema, bem como em algumas das transacções executadas no *back-office*. Como também é possível saber em que dia foi detetada uma atualização de novos *firmware*.

Excerto de Código A.11 Classe dbOvrpCS_Log_Status e classe dbOvrpCS_Log

```
class dbOvrpCS_Log_Status(Base):
    __tablename__ = "Log_Status"
    id = Column(SMALLINT, primary_key=True)
    Status = Column(String(255))
    __table_args__ = {"schema": "OvrP_CS"}

class dbOvrpCS_Log(Base):
    __tablename__ = "OvrP_CS_Log"
    id = Column(Integer, primary_key=True)
    Date = Column(DATETIME(255))
    ErrorMessage = Column(String(5000))
    ErrorMessageStatus = Column(BINARY())
    idStatus = Column(SMALLINT(), ForeignKey(dbOvrpCS_Log_Status.id))
    Message = Column(String(max))
    idUser = Column(SMALLINT(), ForeignKey(Users.idUser))
    __table_args__ = {"schema": "OvrP_CS"}
```

No Excerto de Códigoa A.11 estão disponíveis as classes que mapeiam as tabelas `OvrP_CS_Log` e `Log_Status`. Na tabela `Log_Status`, apenas estão definidos os vários eventos por omissão. Na tabela `OvrP_CS_Log`, é onde são registadas os vários acontecimentos. A coluna `idStatus` está ligada à tabela `Log_Status` via *foreign key*. Os códigos por defeito são os seguintes:

1. “Updates Detected”;
2. “No Updates Detected”;
3. “Error while updating”;
4. “Backoffice Error/Bug”;
5. “Backoffice operation”

Sempre que existe um evento, é armazenada a data do evento. Em caso de erro, é utilizada uma das mensagens de erro por defeito e o relatório do erro é armazenado na coluna `ErrorMessage`. Em caso de detetar alteração, os código afectados bem com as versões de *firmware* são armazenada na coluna `Message`.

Anexo B

Rotas

B.1 Função de Validação do Token

A função que valida o *token* está representada no Excerto de Código B.1. Com recurso ao módulo `functools.wraps`, é possível adicionar as funcionalidades de copiar o nome da função, a lista de argumenstos, etc..., e sendo o `wraps` um *decorator*, é possivel fazer com que antes de executar qualquer função de um *endpoint*, seja executada a validação do *token* antes de ser processado o pedido. Isto apenas é possível devido aos *decorators*, que são funções que permitem modicar a funcionalidade de outras funções. Como se pode verificar no Excerto de Código B.1, antes de executar a função `decorated`, foi adicionado `@wraps(f)`. Assim, é possível aceder às propriedades da função *pre-decoreted* na função `decorated`. Para validar um utilizador normal (`@token_requird`), quando esta função é invocada, esta chama outra função que apenas faz uma pesquisa da existência do utilizador na tabela `Users` base de dados. Caso positivo, a função retorna o decorador e o *endpoint* é executado.

No caso de ser um *endpoint* crítico, que apenas pode ser executado por utilizadores com nível de administrador, é executado o *decroter* `admin_token_required`, cujo código pode ser consultado no Excerto de Código B.2. Neste caso, é verificado o tipo de utilizador em questão. Caso seja administrador, o valor retornado é `1`. Em caso positivo, executa a *route*. Caso não se verifique que é administrador, a função retorna um erro e não executa a *route*. Desta forma, garante-se que apenas as pessoas com privilégios elevados podem fazer alterações criticas ao serviço. Os *decorators* mencionados neste subcapítulo, para além de adicionarem a segurança necessário à API, também validam o cabeçalho de cada pedido.

Excerto de Código B.1 Função token_required

```
def token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        token = None
        if 'x-access-token' in request.headers:
            token = request.headers['x-access-token']
        if not token:
            return Response(json.dumps({'message': 'Token is\
            missing!'}), status=401, mimetype='application/json')
        try:
            data = jwt.decode(token, app.secret_key, 'HS256')
            user_machine = Token.check_token_required(data)
            if not user_machine:
                return Response(json.dumps({'message': 'Token is\
                invalid!'}), status=401, mimetype='application/json')
            else:
                pass
        except jwt.ExpiredSignatureError:
            return Response(json.dumps({'message': 'Token is\
            invalid!'}), status=401, mimetype='application/json')
        return f(*args, **kwargs)
    return decorated
```

Excerto de Código B.2 Função admin_token_required

```
def admin_token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        token = None
        if 'x-access-token' in request.headers:
            token = request.headers['x-access-token']
        if not token:
            return Response(json.dumps({'message': 'Token is\
            missing!'}), status=401, mimetype='application/json')
        try:
            data = jwt.decode(token, app.secret_key, 'HS256')
            admin = Token.check_admin_token_required(data)
            if int(admin) == 1:
                pass
            else:
                return Response(json.dumps({'message': 'Token is\
                invalid!'}), status=401, mimetype='application/json')
        except jwt.ExpiredSignatureError:
            return Response(json.dumps({'message': 'Token is\
            invalid!'}), status=401, mimetype='application/json')
        return f(*args, **kwargs)
    return decorated
```

B.2 Rotas

B.2.1 Device

B.2.1.1 GET

Na Figura B.1 está representada o diagrama de sequência deste pedido. O utilizador ou máquina iniciam o pedido à API. A API tem implementada a classe

`Device` que por sua vez tem implementada várias funções, neste caso a função selecionada é a função `get`. Esta função tem como parâmetro de entrada a variável `sap` e retorna os dados referentes ao produto/sap pedido. Para consultar os dados é feita uma *query* à base dados, que está descrita no diagrama. É possível verificar que esta *query* está a consultar e a juntar informação de várias tabelas, sendo que não é a forma mais eficiente de fazer a *query*, mas devido a limitações e regras definidas pela organização, não foi possível criar visualizações de tabelas na base de dados. Quando é necessário obter dados de várias tabelas na mesma *query*, é necessário recorrer ao `JOIN`. Neste caso, é executado o `JOIN` entre a tabela `firmware_in_production` e as respetivas tabelas das várias partes do *firmware*. Recorrendo ao SQLalchemy ORM, torna-se uma tarefa menos suscetível a erros de sintaxe.

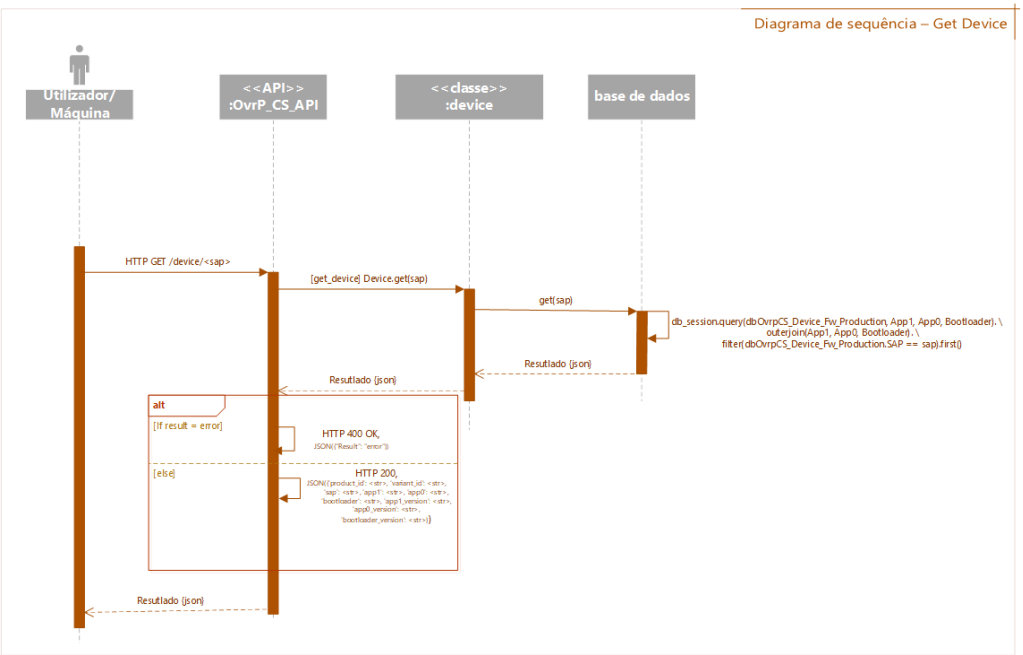


Figura B.1: Diagrama de Sequência UML HTTP GET Device

B.2.1.2 POST

Na Figura B.2 está visível o diagrama de sequência do pedido HTTP POST. O administrador do sistema, através do *back-office*, pode adicionar manualmente um `device` à lista de dispositivos. Como é possível verificar no diagrama, é o administrador que dá inicio ao processo. Este acede ao *back-office* e ao clicar na função de adicionar, o *back-office* vai mostrar um menu para preencher todos os dados nos respetivos campos de entrada. Ao clicar adicionar, vai ser enviado um pedido POST à API através da rota HTTP POST /device0. Na API foi criada

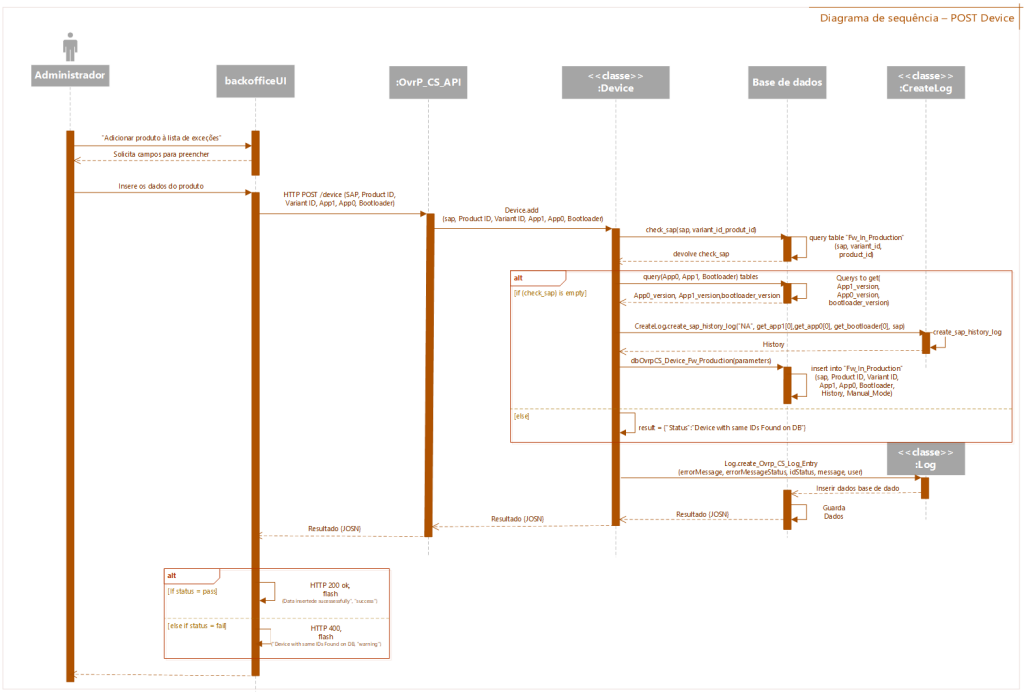


Figura B.2: Diagrama de Sequência UML HTTP POST Device

a classe *Device* como já mencionado. Nesta classe vai ser executada a função `add`. A função verifica se existe algum `device` já criada e em caso de não existir, adiciona o novo produto à tabela da base de dados "fw_in_production". Durante este processo é criado um registo da operação, através da classe `Log` e também são adicionados as respetivas versões de *firmware* no formato JSON à coluna `history` com recurso à função `create_sap_history_log` da classe `CreateLog`.

B.2.1.3 PUT

Este recurso é importante, pois permite atualizar o *firmware* durante a fase de industrialização de novos produtos. Para atualizar a versão de *firmware* de um determinado produto, é necessário que o *firmware* seja adicionado ao sistema de forma automática ou de forma manual, uma vez que o *back-office* e a API apenas permitem associar uma versão de uma parte do *firmware* de um produto a uma versão que já está adicionada nas tabelas `App0`, `App1` ou `Bootloader`. Para além de permitir a atualização das partes constituintes do *firmware*, também permite a alteração das restantes colunas, nomeadamente, `SAP`, `Variant ID`, `Product ID` e `manual_mode`. Foi também adicionado a funcionalidade de permitir passar para gestão manual dispositivos que são geridos pelo desenvolvimento. Este foi um requisito da fábrica, sendo que em situação especiais poderá ser necessário alterar determinado dispositivo manualmente. Para isto, através do *backkoffice*, é apenas

necessário procurar pelo dispositivo na *tab* `Device Overview List`, e através do botão `Manual` é possível remover determinado dispositivo da atualização automática. Caso apenas seja necessário atualizar um dispositivo cujo modo manual está ativado, basta aceder à *tab* `Sample Runs/Exceptions Device List` e através do botão edit é possível editar todos os parâmetros definidos. Em ambas as situação, a rota utilizada é HTTP PUT /device/“<device_id>”.

B.3.

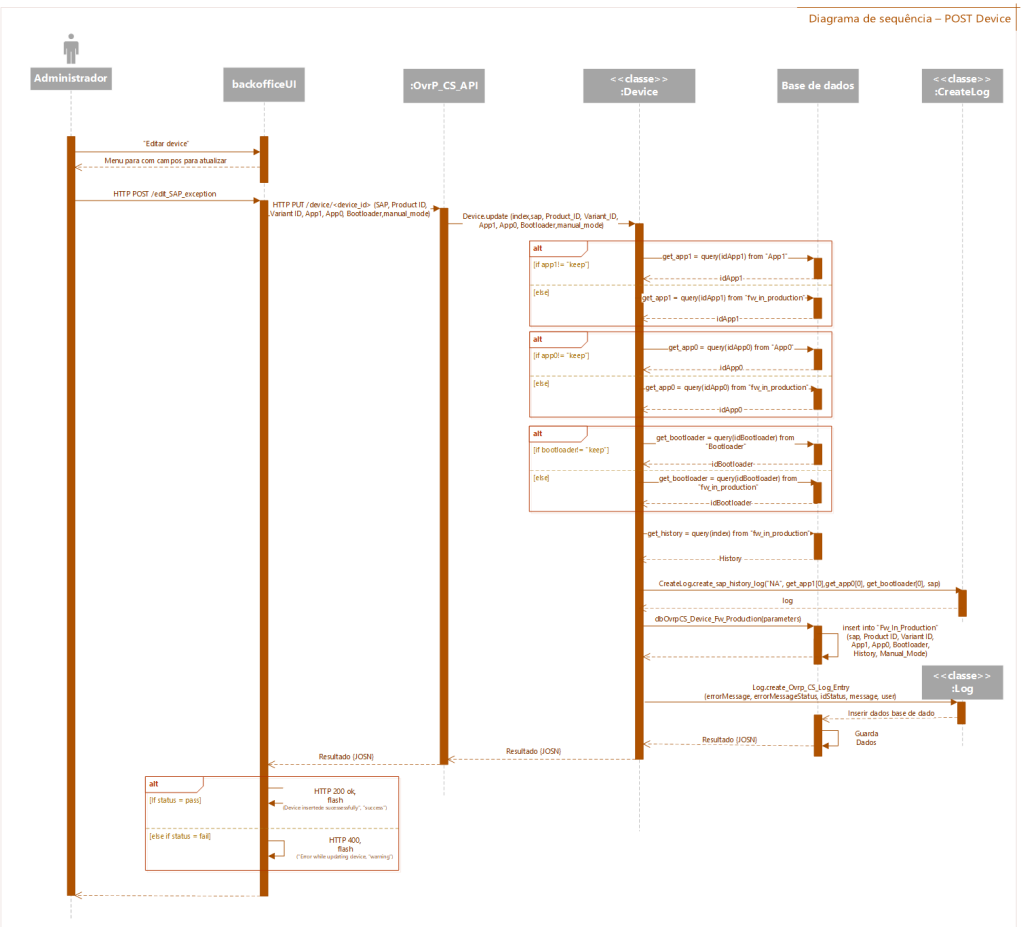


Figura B.3: Diagrama de Sequência UML HTTP PUT Device

B.2.1.4 DELETE

Na Figura B.4, é possível consultar o Diagrama de Sequência UML deste pedido.

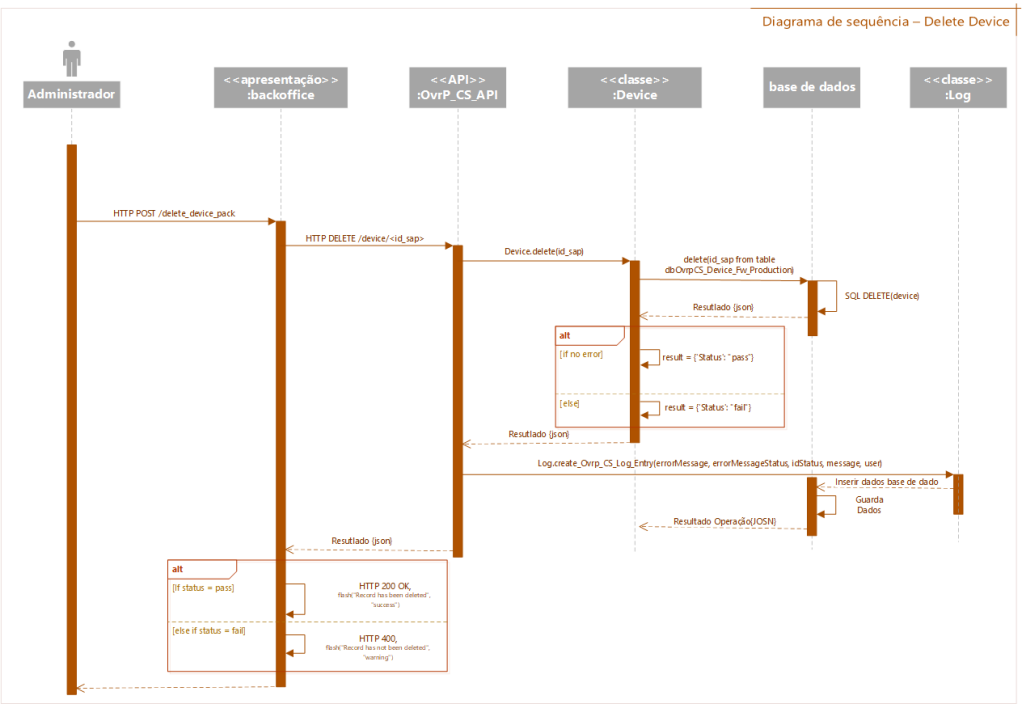


Figura B.4: Diagrama de Sequência UML HTTP DELETE Device

B.2.2 Users

Neste Capítulo, estão disponíveis os Diagramas de Sequência UML referentes à rota Users.

B.2.2.1 POST

Para gerar as *passwords* dos utilizadores, foi implementado uma função, enc_pass(), como se pode verificar no diagrama de sequência UML da Figura B.5 . Ao adicionar um utilizador, é enviado o pedido HTTP POST à API. A API tem a classe `Users` com as funções de `add`, `update` e `delete` implementadas. Para o método `POST`, é utilizada a função `add`, que antes de adicionar um utilizador, verifica se o mesmo já consta na tabela `Users`, através da variável `check_email`. Caso não exista, adiciona o utilizador. Só após ser adicionado com sucesso, é enviado um email com o *link* do `back-office` e com a *password* temporária do utilizador.

B.2.2.2 PUT

Para um melhor entendimento deste método, o Diagrama de Sequência UML deste pedido pode ser consultado na Figura B.6 O identificador único do utilizador é passado através do URI. No *body* do pedido, são enviados todos as variáveis.

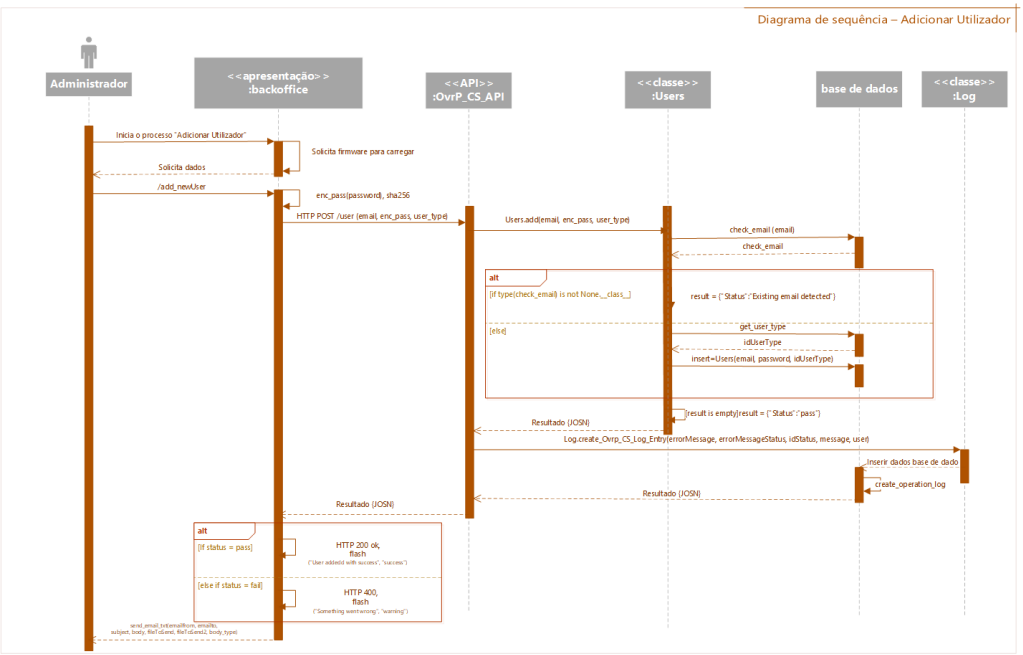


Figura B.5: Diagrama de Sequência UML HTTP POST User

Por defeito, ao atualizar um utilizador via *back-office*, ao abrir o menu com os dados, são preenchidos todos os campos com os valores atuais, sendo que apenas se altera o campo pretendido. Desta forma, apesar de se atualizar todos os campos ao executar a *query* à base de dados, apenas o campo alterado é atualizado sendo que os restantes são substituídos pelo mesmo valor.

B.2.2.3 DELETE

Como especificado pela arquitetura REST, para o método `HTTP DELETE`, o utilizador que se pretende apagar está referenciado no URI do pedido. Ao receber o pedido, a API automaticamente sabe qual o utilizador que tem que remover. Na API, como já mencionado, foi criada uma classe `Users`, na qual se implementou a função `delete`. Esta função apaga o utilizador através do ID único passado e não através do email. Desta forma garante-se que se apaga o utilizador correto. Todas as operação, validações e respostas implementados, estão visíveis no diagrama de sequência UML da Figura B.7.

B.2.3 Machine

B.2.3.1 POST

Como a rota é a mesma para ambas as situações, no caso de uma máquina pertencente à fábrica de Ovar, a rota verifica se o campo "CIname"é diferente

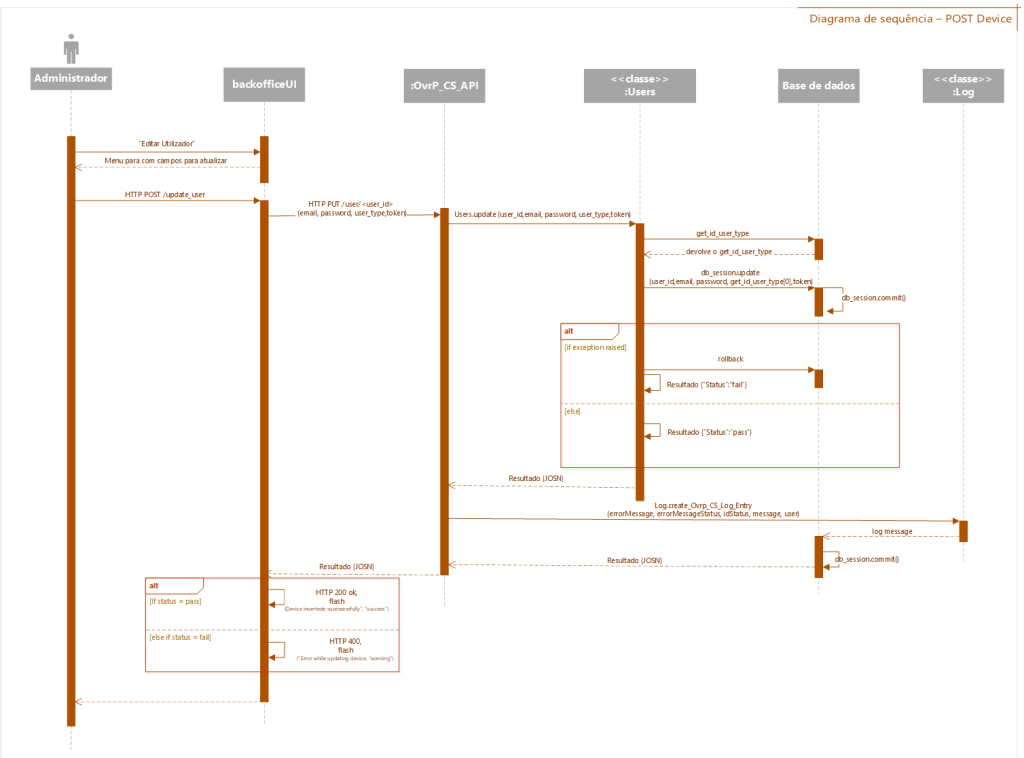


Figura B.6: Diagrama de Sequência UML HTTP PUT User

de "None". Nesta situação, é executado a função `add` da classe `OvrpMachine`, que como o nome indica, tem como função adicionar uma máquina à tabela ■`Pack_Machines2"`. Antes de adicionar a máquina é feita a verificação se a máquina já existe ou não. Apenas adiciona se não existir, e em case de existir avisa o administrador. Caso a rota receba o campo "CIname"igual a "None", quer dizer que a máquina não pertence à fábrica de Ovar, e nesta situação é executado a função `add` da classe `NonOvrpMachine`. É também feita a verificação de duplicados na base de dados, mas nesta situação a tabela consultada é a ■`Pack_Machines"`. Apenas insere os dados na base dados em caso de não existirem duplicados, caso contrário o administrador recebe um aviso. O Diagrama de Sequência UML deste pedido está descrito na Figura B.8.

B.2.3.2 PUT

Na Figura B.9, pode-se verificar de uma forma mais detalhada todos as funções executadas para processar este pedido.

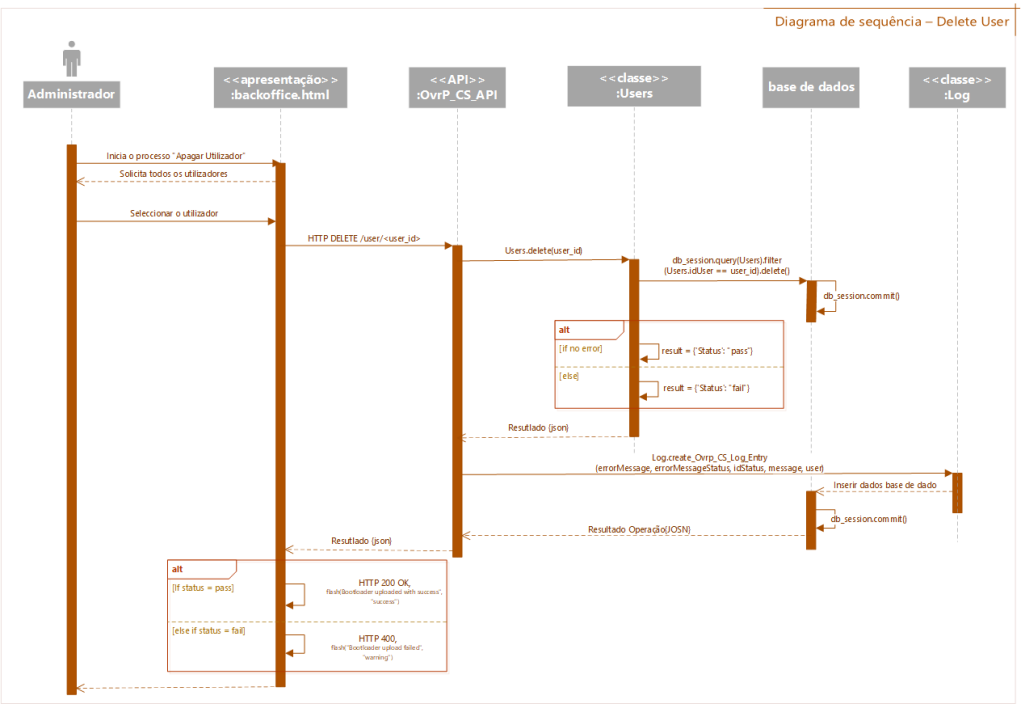


Figura B.7: Diagrama de Sequência UML HTTP DELETE User

B.2.3.3 DELETE

O *index* da máquina cuja intenção seja remover do sistema, é passado através do URI. Em caso de sucesso, o pedido devolve uma mensagem no formato JSON, com o estado da operação. Em caso de erro ao apagar o registo, a rota foi desenvolvido com mecanismos de identificação e tratamento de erro, ou seja, para esta situação, perante uma exceção durante a *query* para apagar o registo, a API cancela a operação e faz o *rollback*. De seguida é possível visualizar o Diagrama de Sequência UML deste pedido na Figura B.10. A identificação da máquina é feita através do campo `machine_id` enviado no URI.

B.2.4 App0

B.2.4.1 GET

O utilizador ou máquina necessitem de identificar no URI qual a versão e o nome do respetivo *firmware*. Com estes dados a API consegue facilmente enviar o ficheiro com recurso à biblioteca send_from_directory(). O pedido GET apenas é terminado após ser enviado o ficheiro.

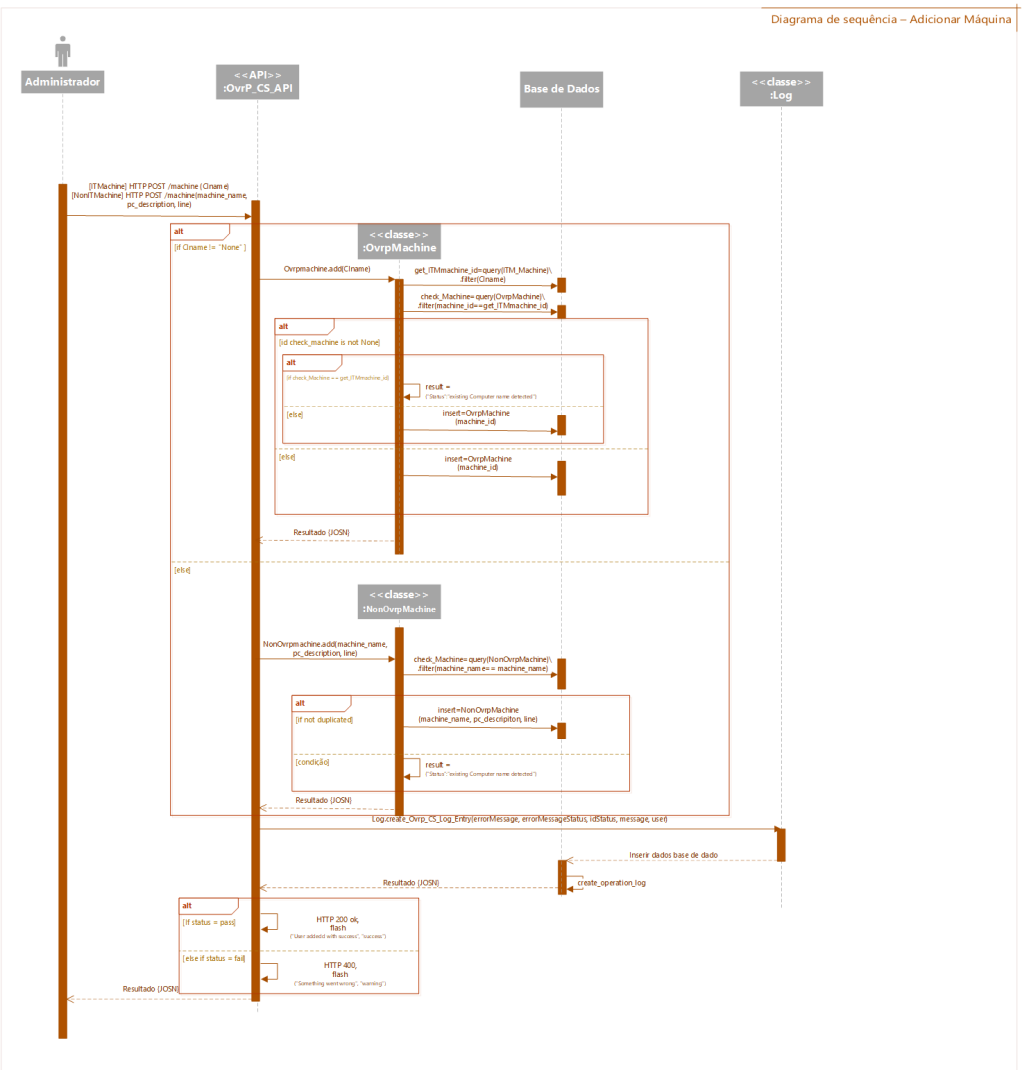


Figura B.8: Diagrama de Sequência UML HTTP POST Machine

B.2.4.2 POST

A função criada para decompor o nome é a `get_fw_version` e pertence à classe `database_data_manipulation`. Após o *parse* do nome, é executada a função `fw_manually_addedd` da classe `FW_Manage`. Esta função verifica se o ficheiro já existe no servidor, calcula o `checksum` (sh256) e copia o ficheiro para o servidor que contém a estrutura de pastas de armazenamento dos vários tipos de *firmware*. Assim, sempre que alguma máquina fizer o *download* deste ficheiro, pode calcular o *checksum* e comparar com o calculado pela API, garantido que o ficheiro não está corrompido. Para a operação de adicionar é feito o registo da operação na tabela "OvrP_CS_Log". Para se entender de forma detalhada a sequência de

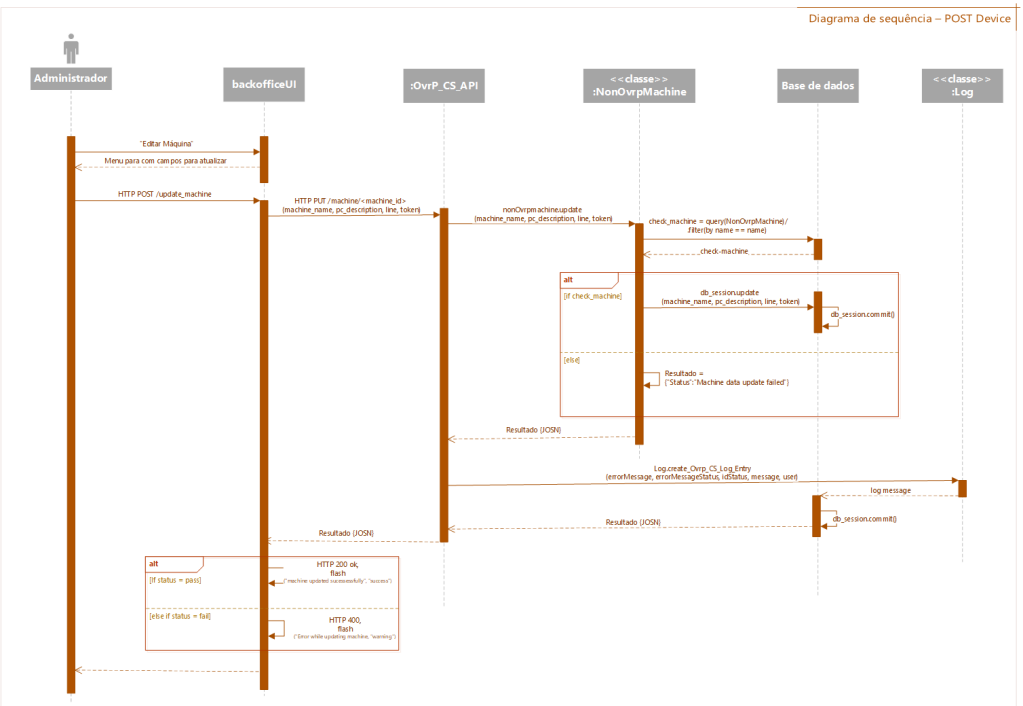


Figura B.9: Diagrama de Sequência UML HTTP PUT Machine

adiconar um ficheiro, pode ser consultada o diagrama de sequência UML descrito na Figura B.12

B.2.4.3 PUT

B.2.4.4 DELETE

B.2.5 App1

Neste Capítulo serão apresentados os respetivos Diagramas de Sequência UML referentes a cada método implementado para esta rota.

B.2.5.1 GET

O utilizador ou máquina necessitem de identificar no URI qual a versão e o nome do respetivo *firmware*. Com estes dados a API consegue facilmente enviar o ficheiro com recurso à biblioteca send_from_directory(). O pedido GET apenas é terminado após ser enviado o ficheiro. Na Figura B.15 pode ser verificado o Diagrama de Sequência UML.

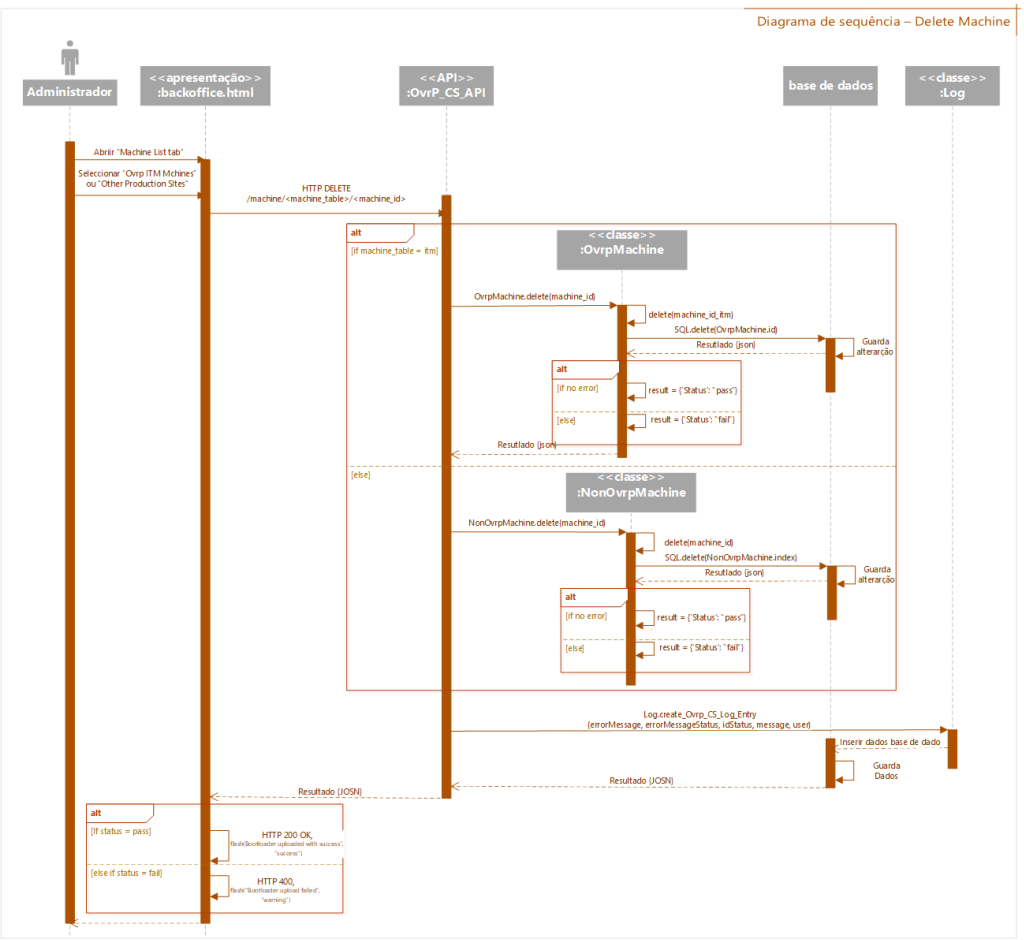


Figura B.10: Diagrama de Sequência UML HTTP DELETE Machine

B.2.5.2 POST

função criada para decompor o nome é a `get_fw_version` e pertence à classe `database_data_manipulation`. Após o *parse* do nome, é executada a função `fw_manually_addedd` da classe `FW_Manage`. Esta função verifica se o ficheiro já existe no servidor, calcula o `checksum` (sh256) e copia o ficheiro para o servidor que contém a estrutura de pastas de armazenamento dos vários tipos de *firmware*. Assim, sempre que alguma máquina fizer o *download* deste ficheiro, pode calcular o *checksum* e comparar com o calculado pela API, garantido que o ficheiro não está corrompido. Para a operação de adicionar é feito o registo da operação na tabela "OvrP_CS_Log". Na Figura B.16, está detalhado o Diagrama de Sequência deste pedido.

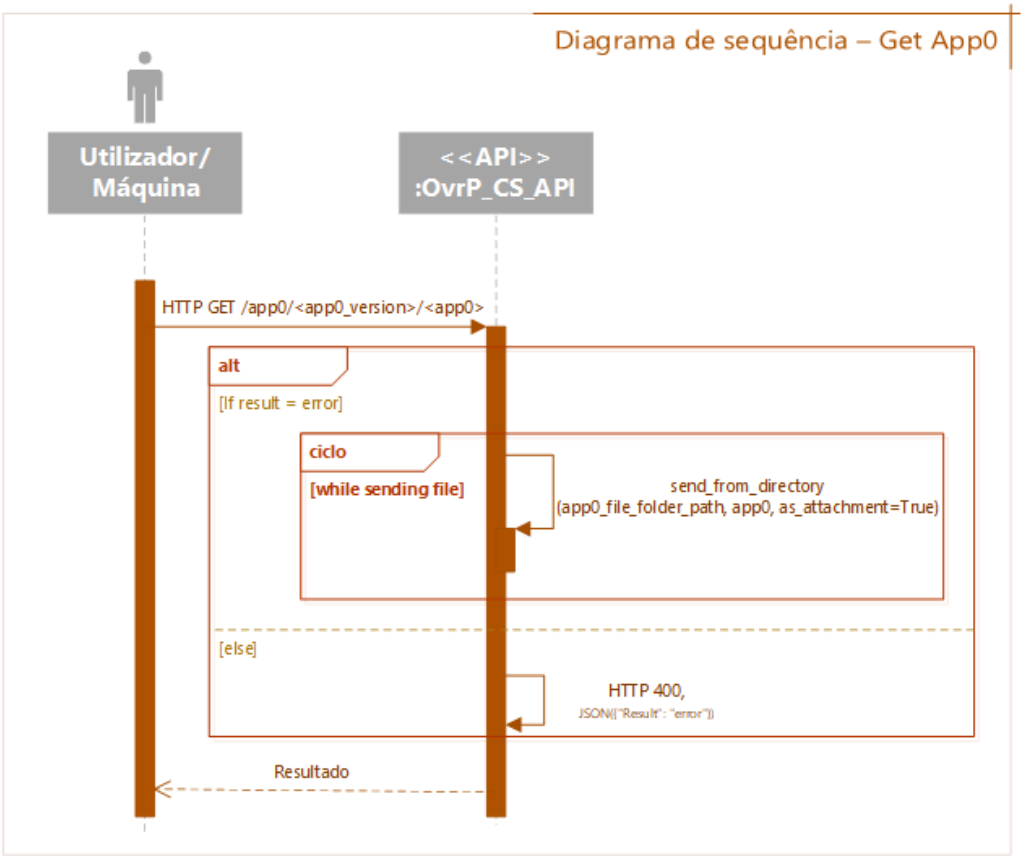


Figura B.11: Diagrama de Sequência UML HTTP GET App0

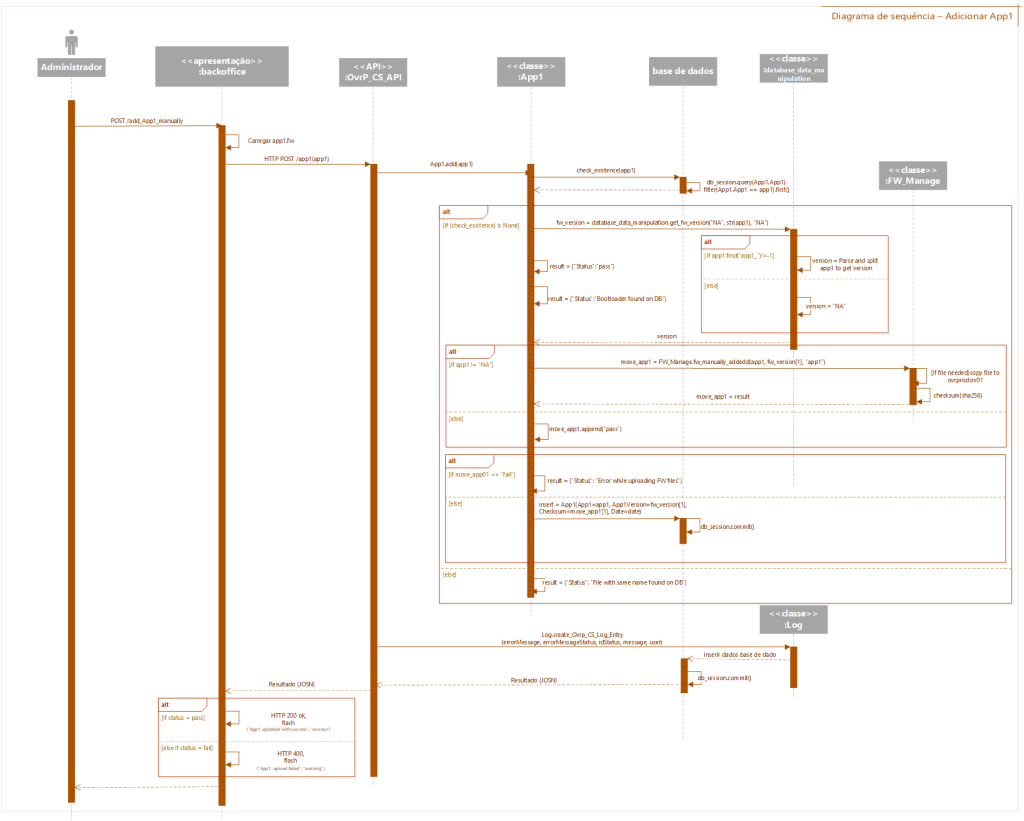


Figura B.16: Diagrama de Sequência UML HTTP POST App1

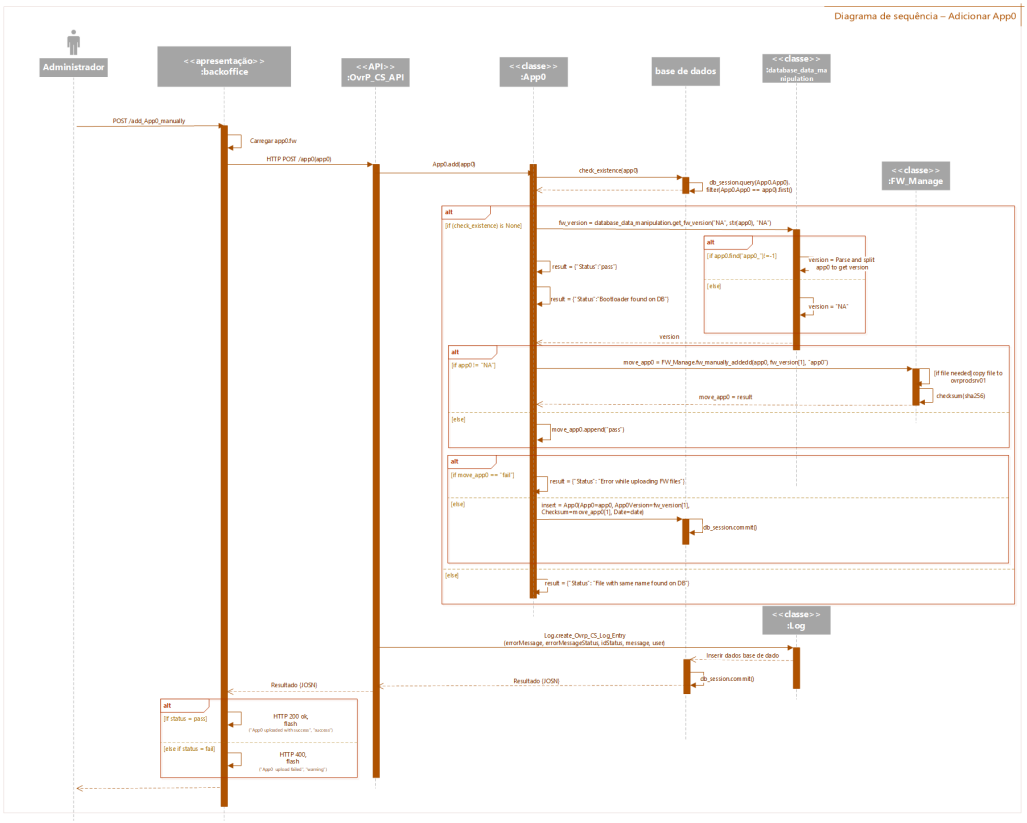


Figura B.12: Diagrama de Sequência UML HTTP POST App0

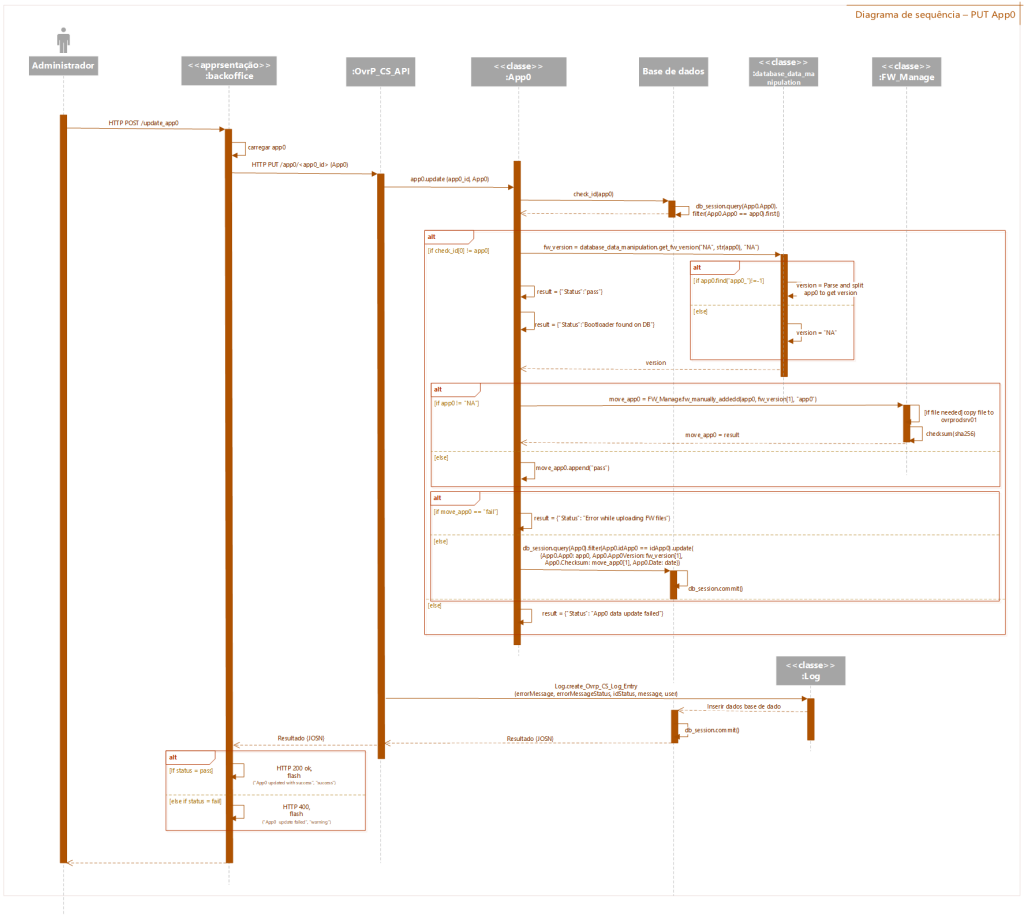
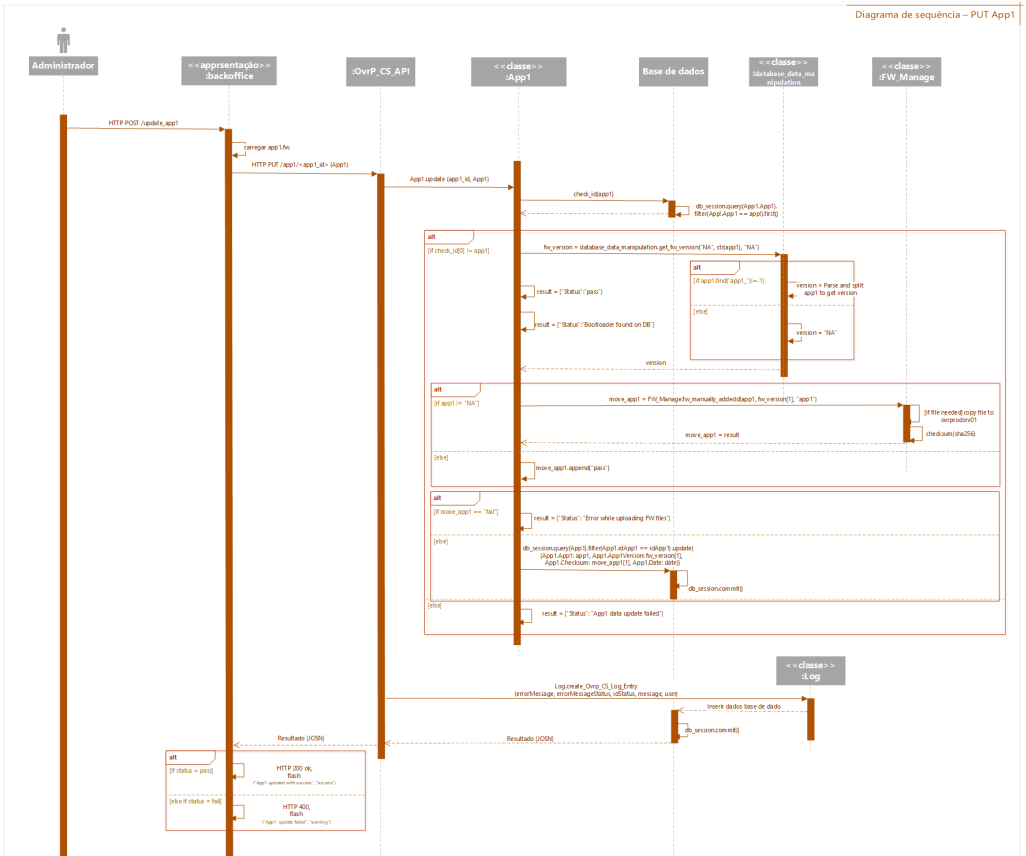


Figura B.13: Diagrama de Sequência UML HTTP PUT App0

B.2.5.3 PUT



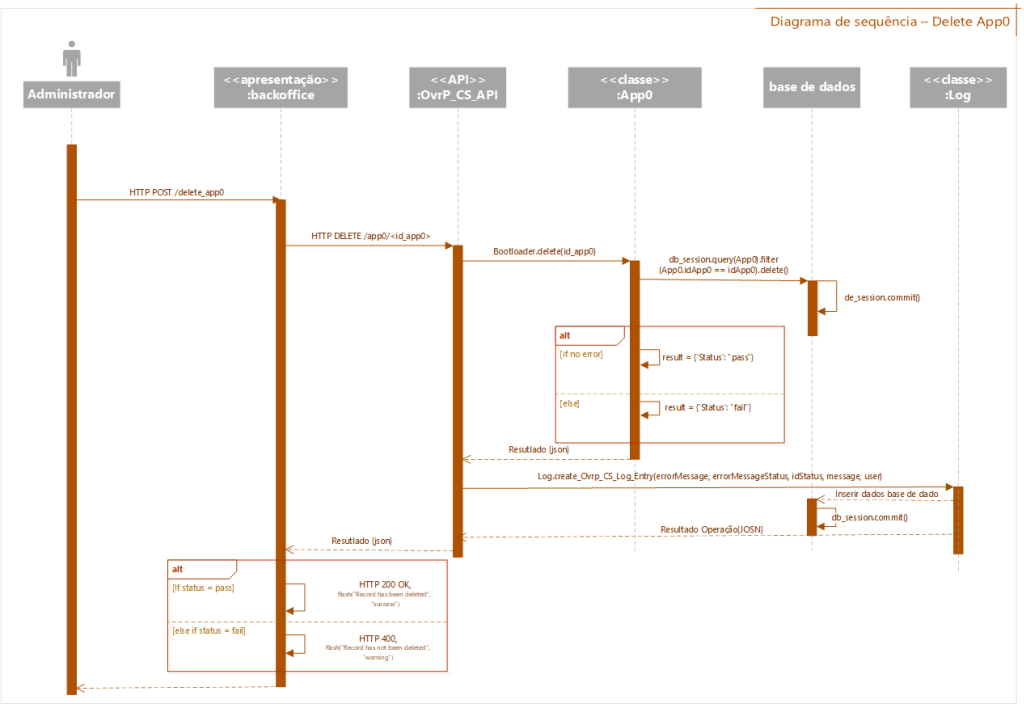


Figura B.14: Diagrama de Sequência UML App0 Delete

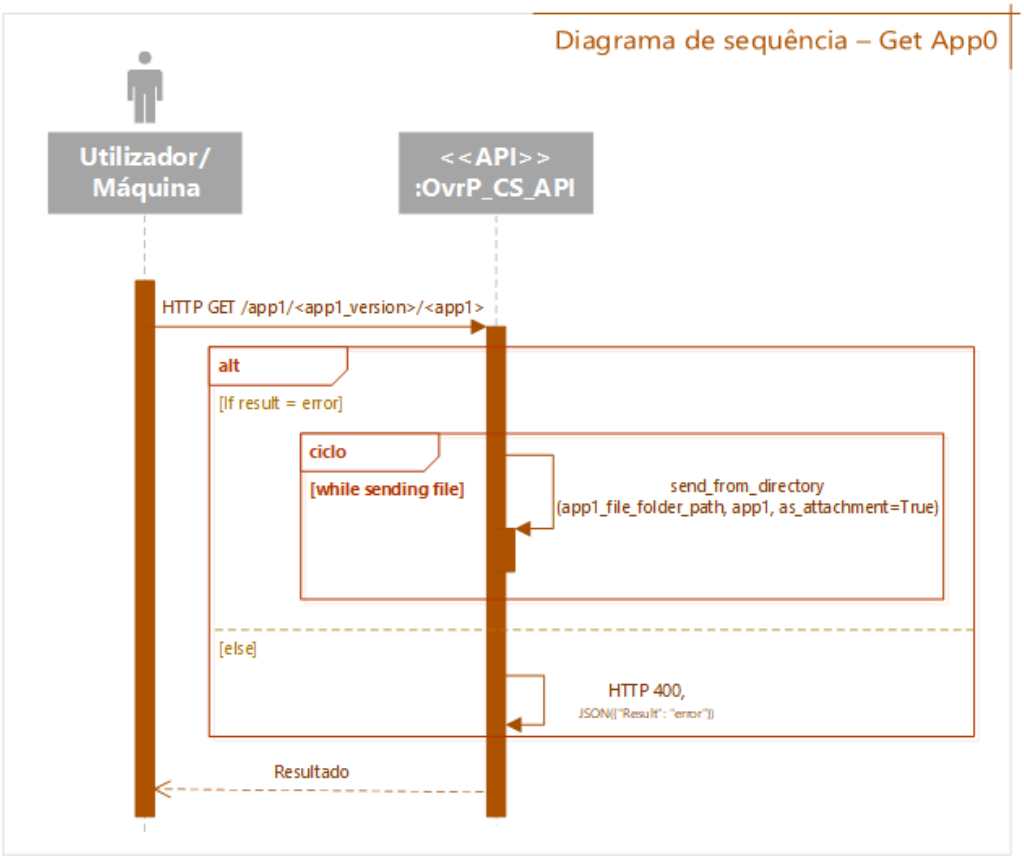


Figura B.15: Diagrama de Sequência UML HTTP GET App1

B.2.5.4 DELETE

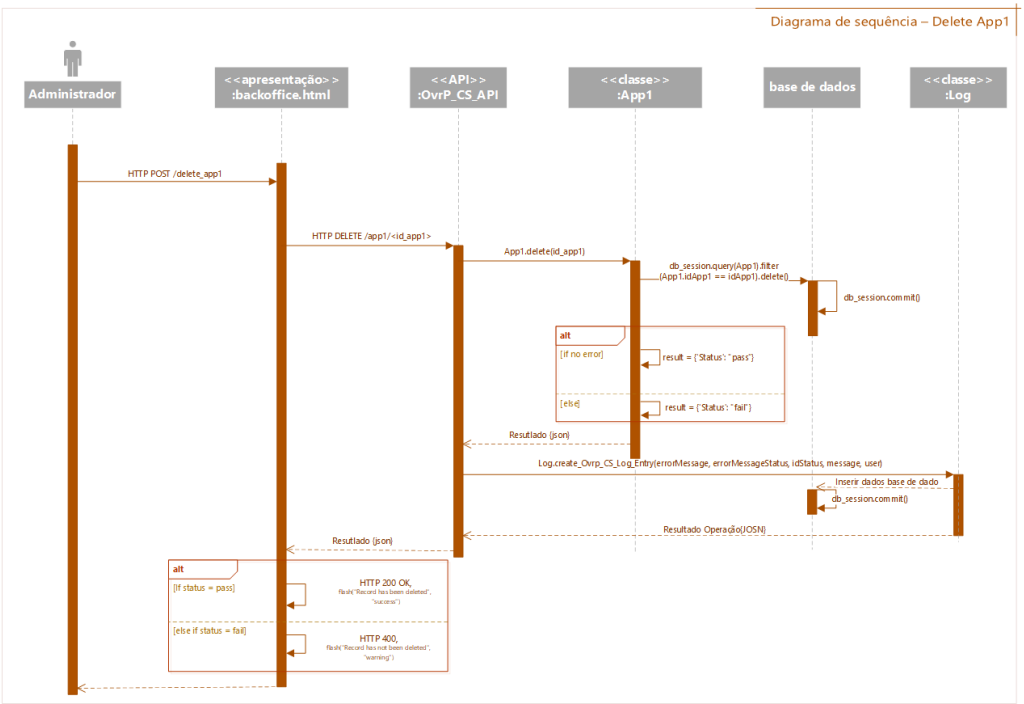


Figura B.18: Diagrama de Sequência UML HTTP DELETE App1

B.2.6 Bootloader

B.2.6.1 GET

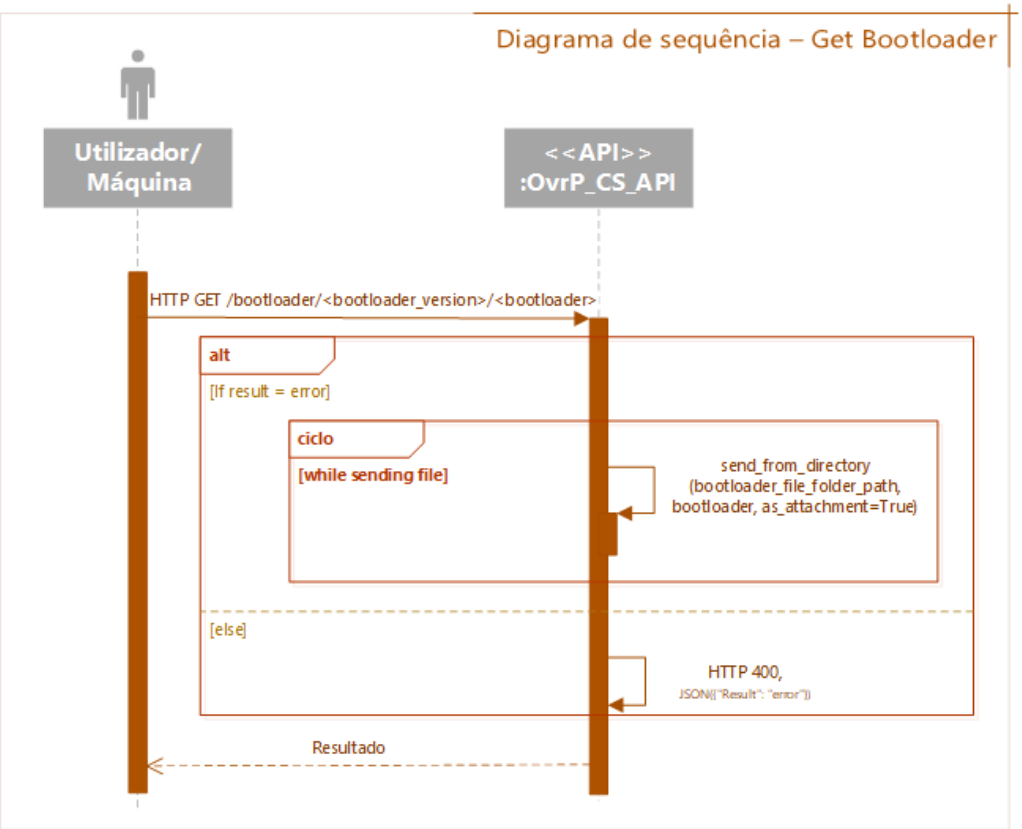


Figura B.19: Diagrama de Sequência UML HTTP GET Bootloader

B.2.6.2 POST

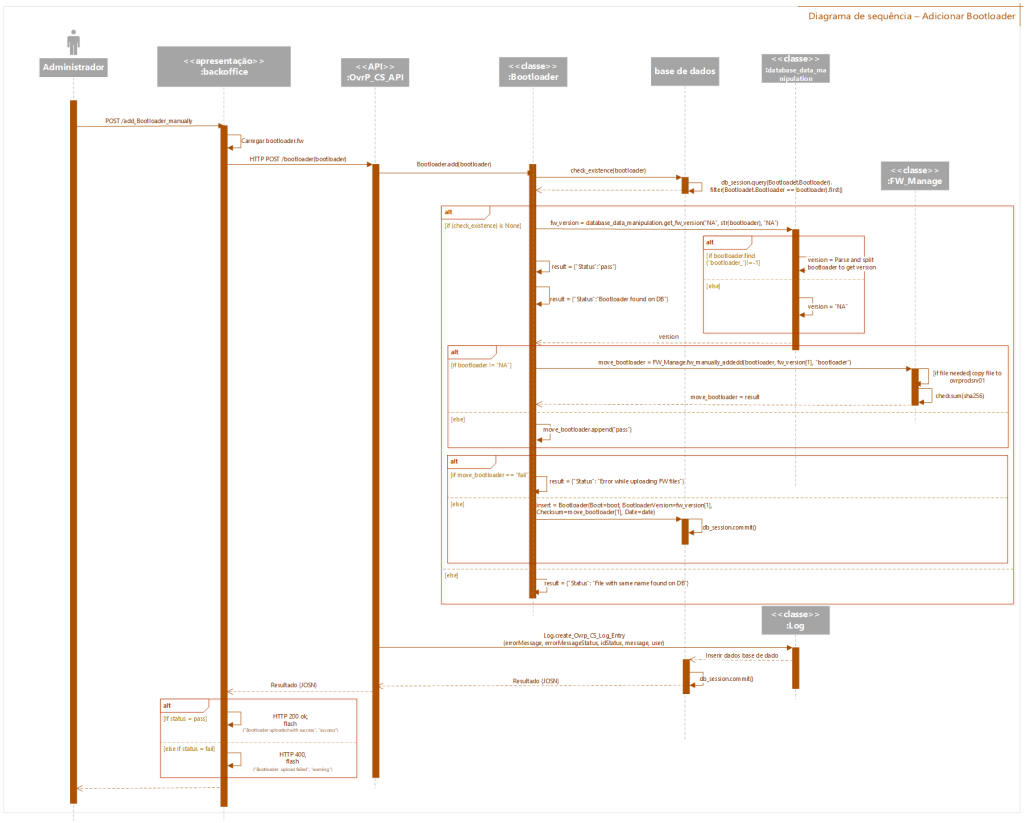


Figura B.20: Diagrama de Sequência UML HTTP POST Bootloader

B.2.6.3 PUT

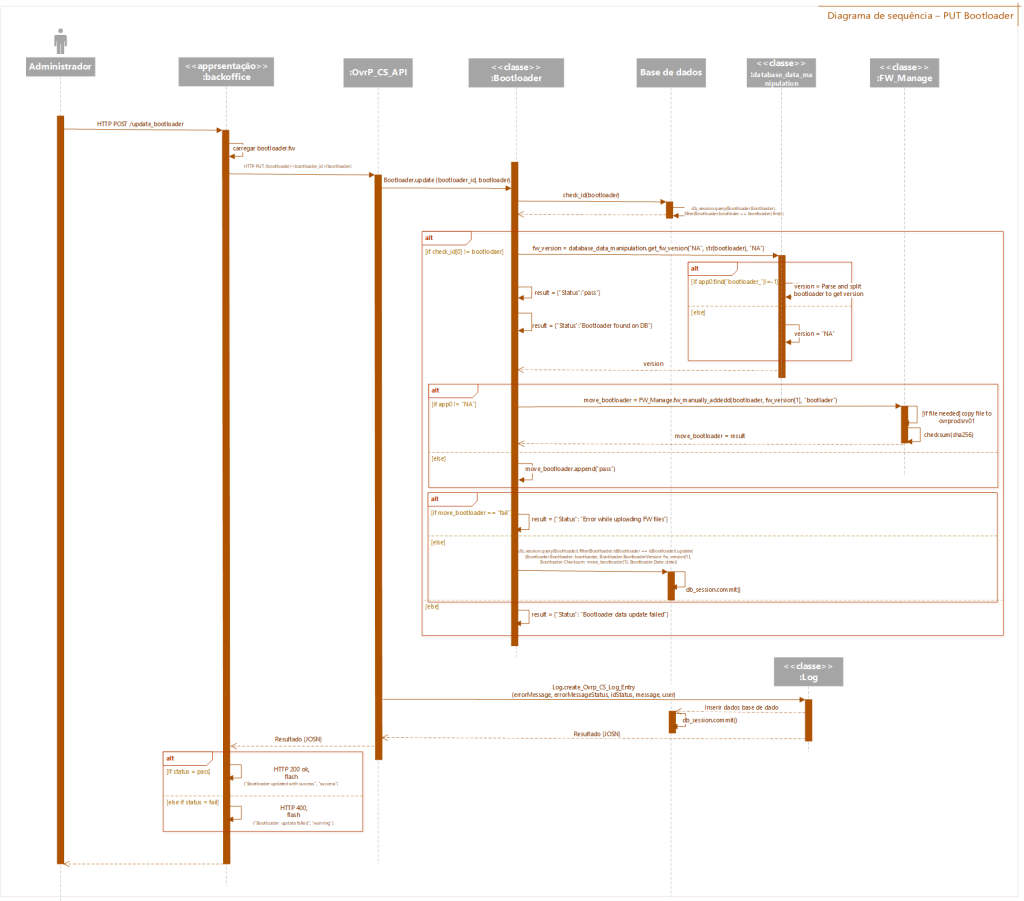


Figura B.21: Diagrama de Sequência UML HTTP PUT Bootloader

B.2.6.4 DELETE

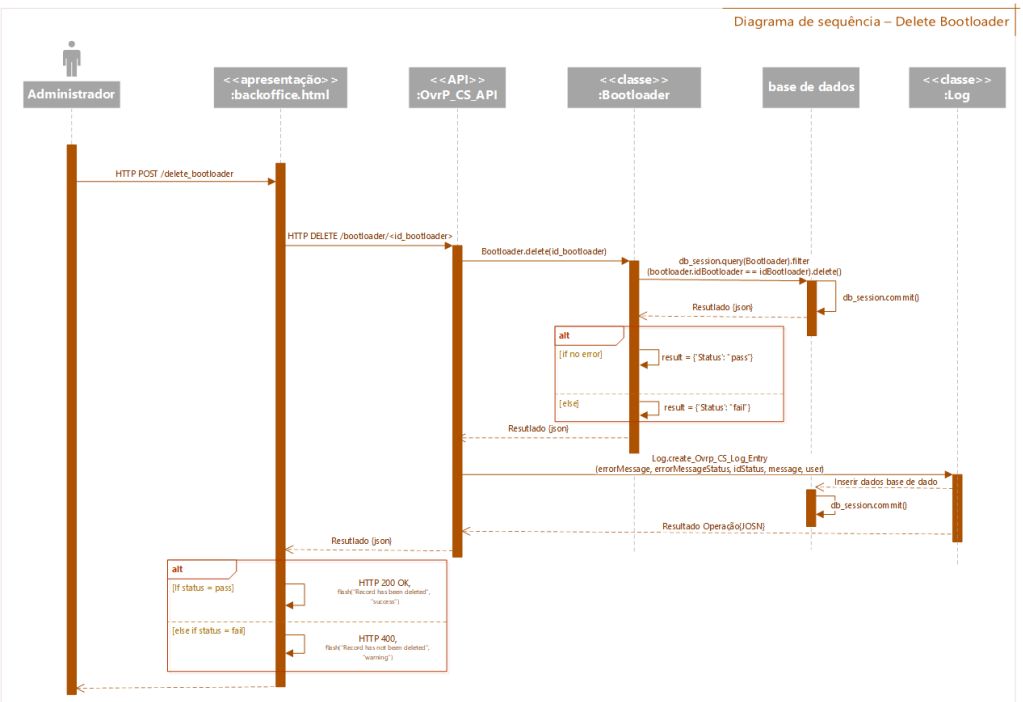


Figura B.22: Diagrama de Sequência UML HTTP DELETE Bootloader

B.2.7 Pack_Version_Table

B.2.7.1 PUT

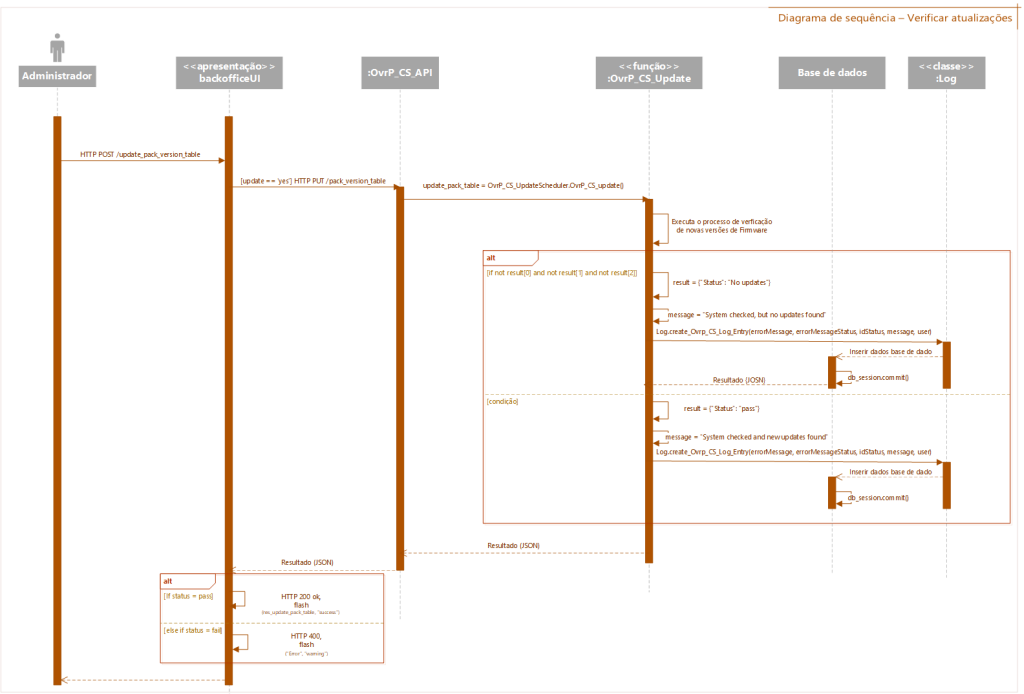


Figura B.23: Pack_Version_Table - HTTP PUT - Sequência UML

Anexo C

Funções e Menus do Backoffice

C.1 Back-office

C.1.1 Back-End

Excerto de Código C.1 Render e passagem de variáveis para a página HTML

```
return render_template('backoffice.html',
pack_versionTable=data_packversiontable, machines=data_machines_list,
bcs_users=data_users,administrator=admin,
pack_versionTable_viewer=result_version_viewer,
system_status_data=system_status,
last_update=last_update1[0],changes_detetcted=changes_detect[0],
app1_list=data_idApp1, app0_list=data_idApp0,
bootloader_list=data_idBootloader,
users_type=data_users_type,ciname_machines_list=data_ciname,
machines_other_sites=data_machines_other_sites)
```

No excerto de código C.1, é possível verificar como é feito o *rendering* da página `backoffice.html` e a passagem dos dados necessários.

Excerto de Código C.2 Objeto Flask.G

```
@app.before_request
def before_request():
    g.user = None
    if 'email' in session:
        g.user = session['email']
```

Excerto de Código C.3 Email de registo de utilizador

```
<html>
<head></head>
<body>
<p>Ovar FW Central System - New account was created by the
administrator.</p>
<p>Your temporary password is the following: + str(temp_pass) + </p>
<p>click on the link below to login and change your password as
soon as possible</p>
<p><a href="http://ovrprodsrv01:6002/">click here</a></p>
</body>
</html>
```

C.1.2 Front-End

Excerto de Código C.4 Utilização das variáveis

```
{% include "header.html" %}
<body>
 {% block body %}
 <div class="container" style="width:95%;">
 <div class="row">
 <div class="col-md-12">
 {% if administrator %}
 <h2>Backoffice</h2>
 <ul class="nav nav-tabs">
```

Como se pode ver pelo código apresentado no excerto de código C.4, para aceder às variáveis passadas é necessário utilizar {% variável %}. Também é aplicável para incluir outras páginas HTML, como se pode verificar na primeira linha de código apresentada, em que se insta a incluir o `header.html`, e permite criar condições como `if` e `else` e até percorrer os dados de uma lista recorrendo ao ciclo `for`.

C.2 Diagramas de Sequência UML

C.2.1 Sign Up

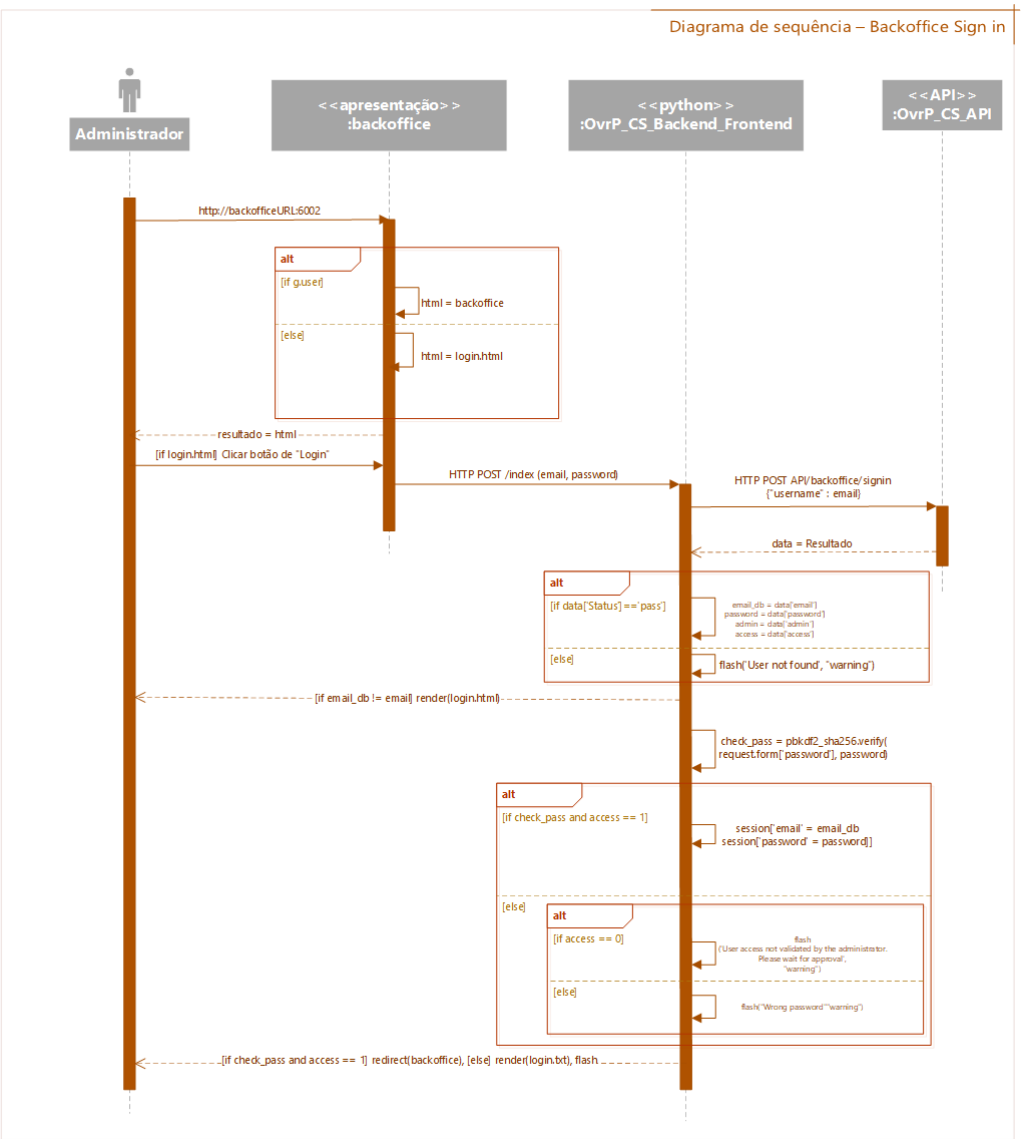


Figura C.1: Diagrama de sequência UML para fazer*sign in*

C.2.2 backoffice.html

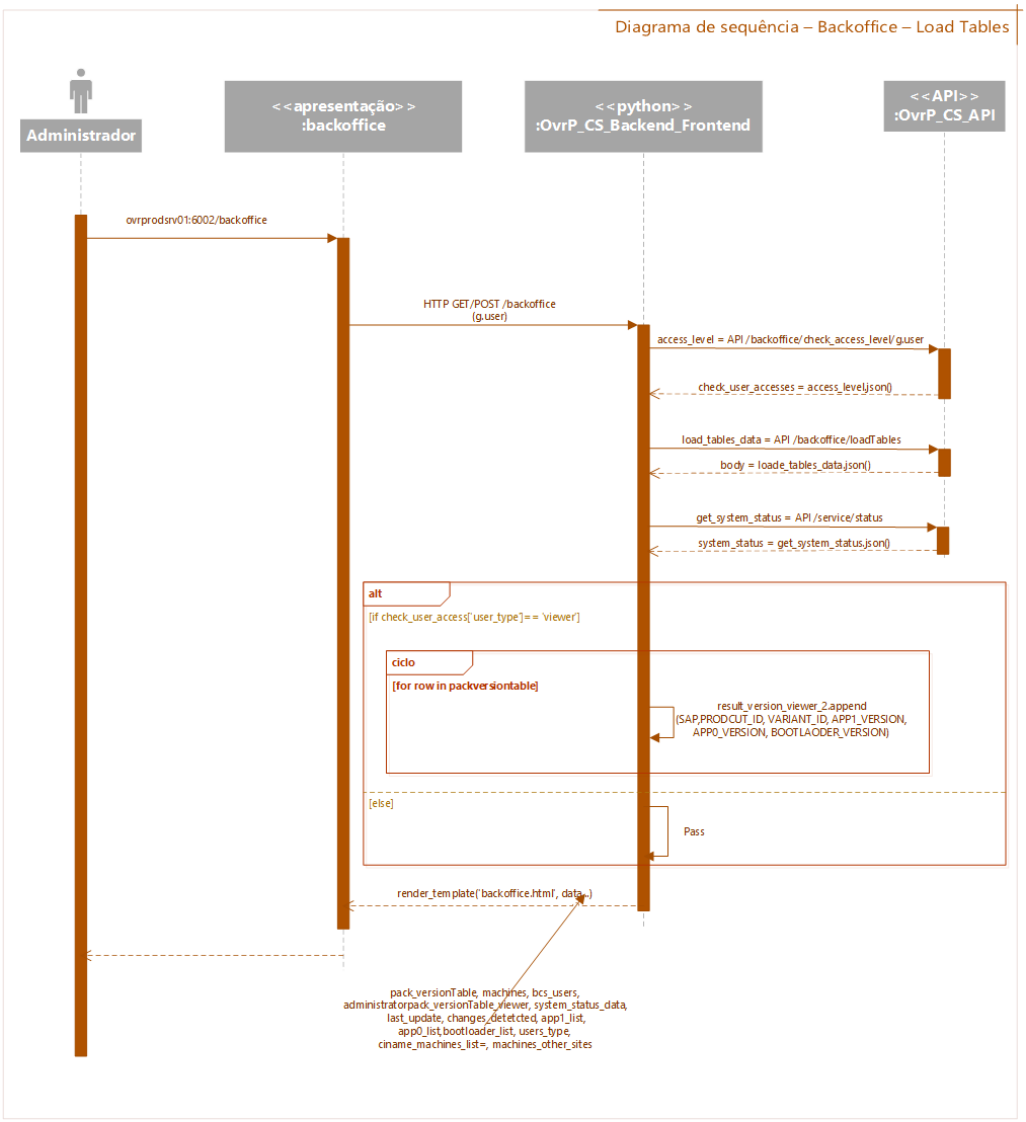


Figura C.2: Diagrama de Sequência UML `loadTables`

C.2.3 Menu Device Overview List

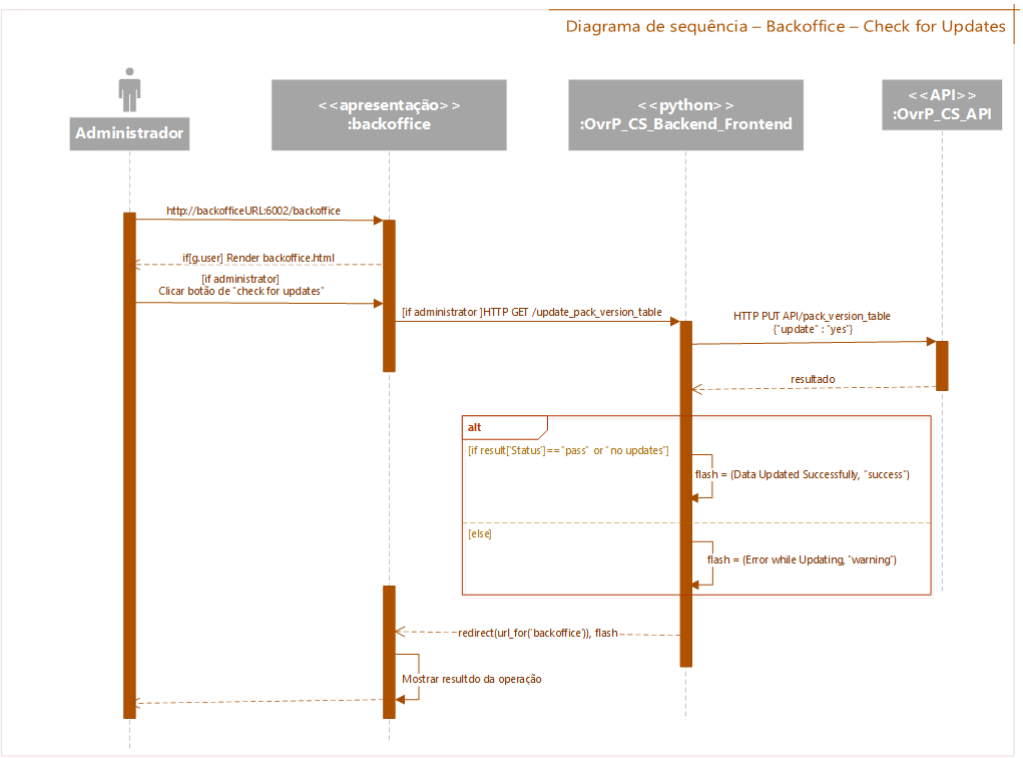


Figura C.3: Diagrama de sequência UML `Check for Updates`

C.2.4 Menu Sample Run/Exceptions Device List

C.2.4.1 Edit

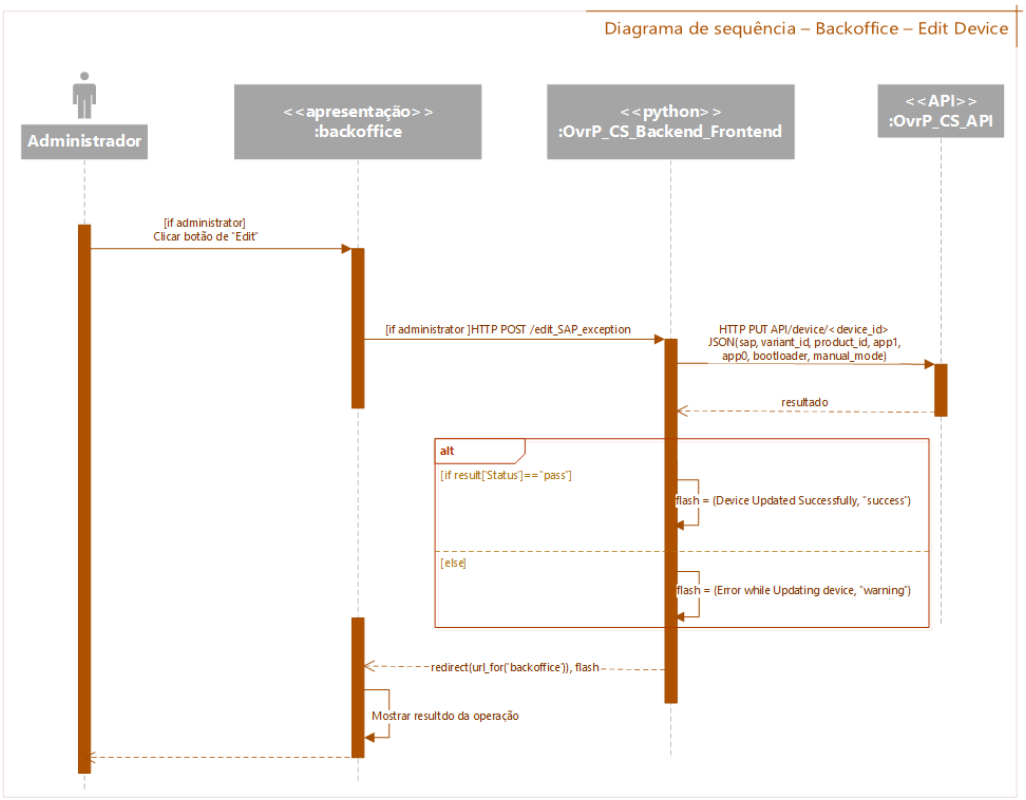


Figura C.4: Diagrama de Sequência UML Edit `Sample Runs/Exceptions Device List`

C.2.4.2 Add

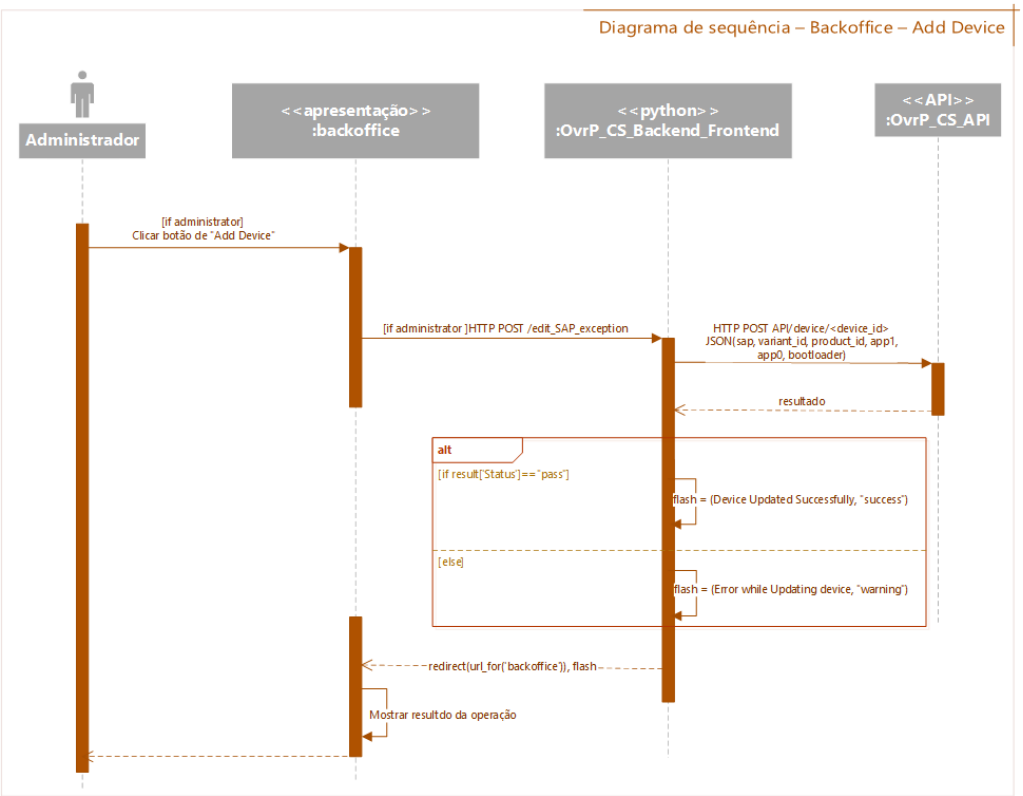


Figura C.5: Diagrama de Sequência UML Add `Sample Runs/Exceptions Device List`

C.2.5 Menu Firmware Overview List

C.2.5.1 Add

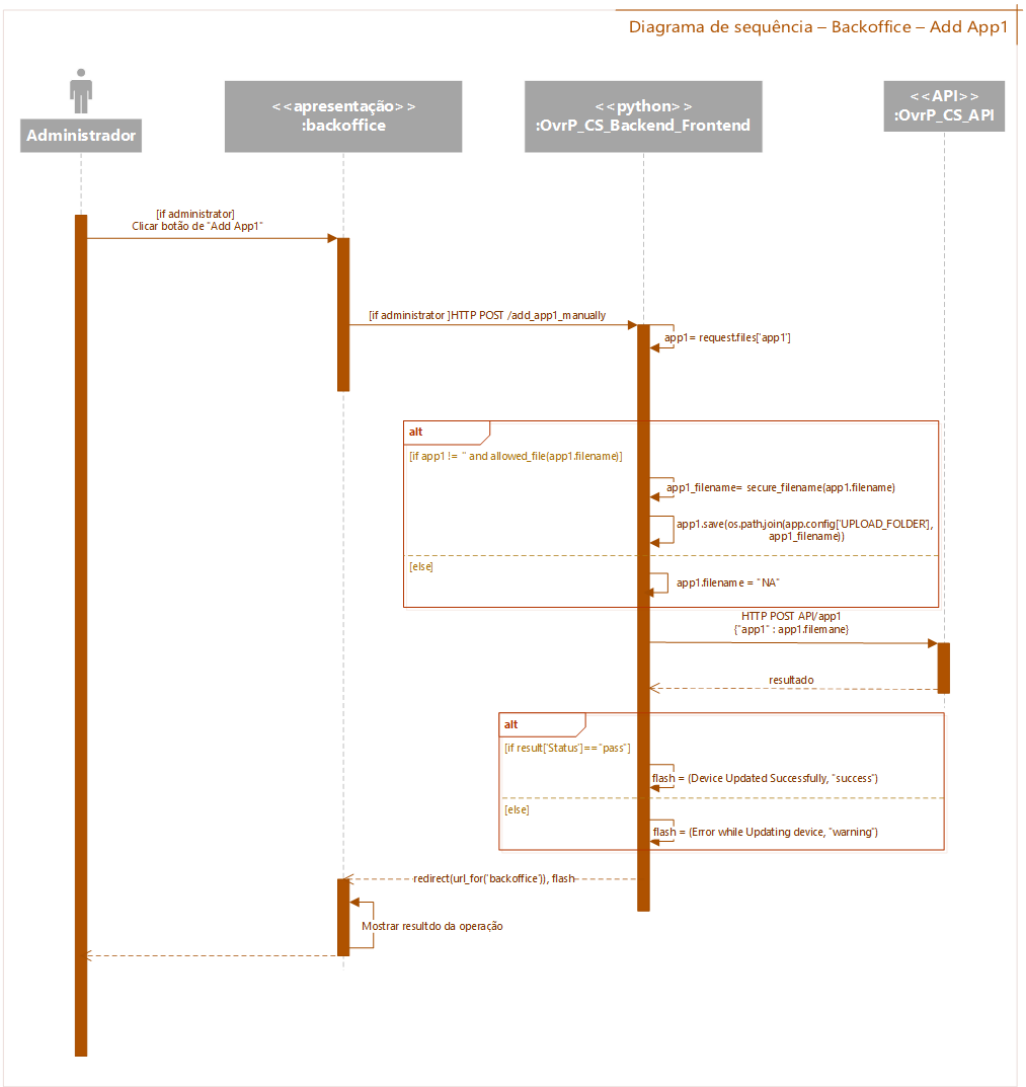


Figura C.6: Diagrama de Sequência UML Add `App1`

C.2.5.2 Edit

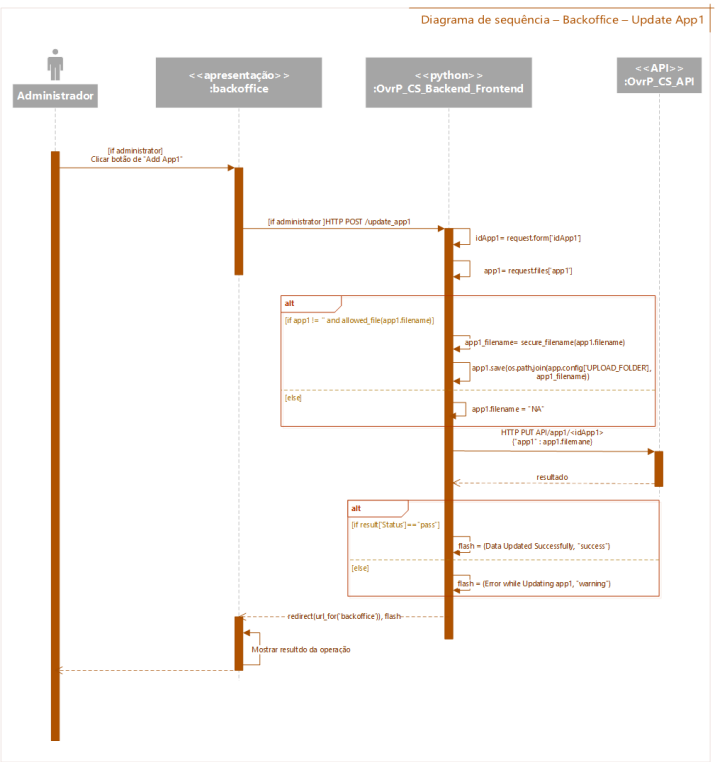


Figura C.7: Diagrama de Sequência UML Edit `App1`

C.2.5.3 Delete

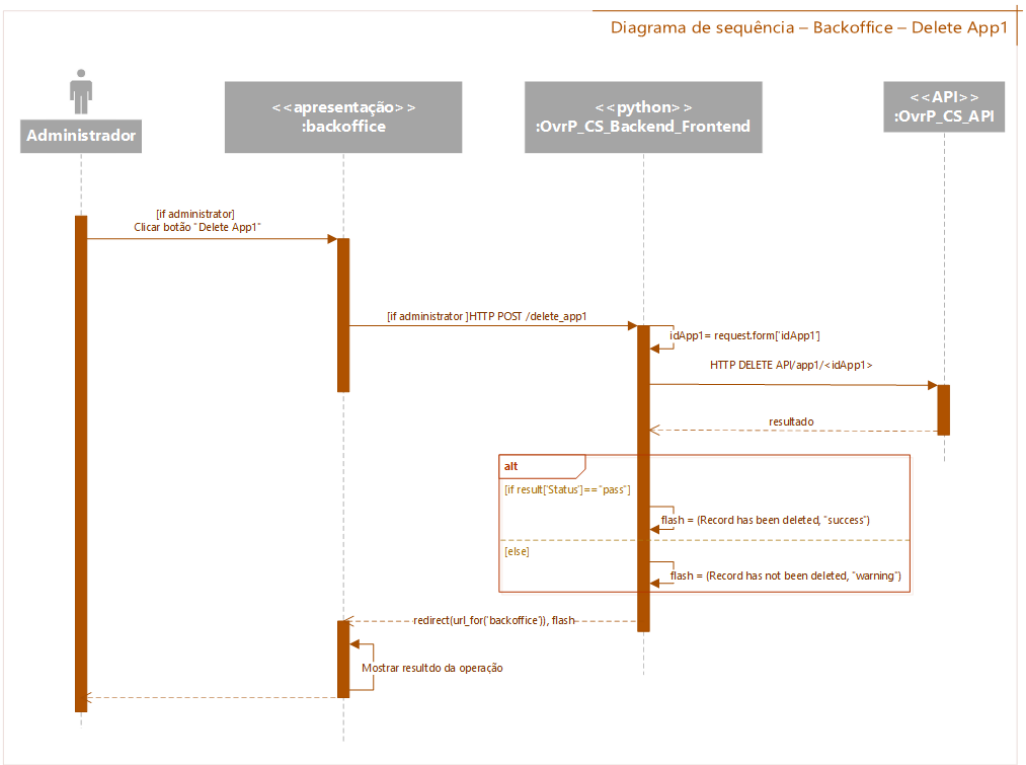


Figura C.8: Diagrama de Sequência UML Delete `App1`

C.2.6 Menu Users List

C.2.6.1 Add

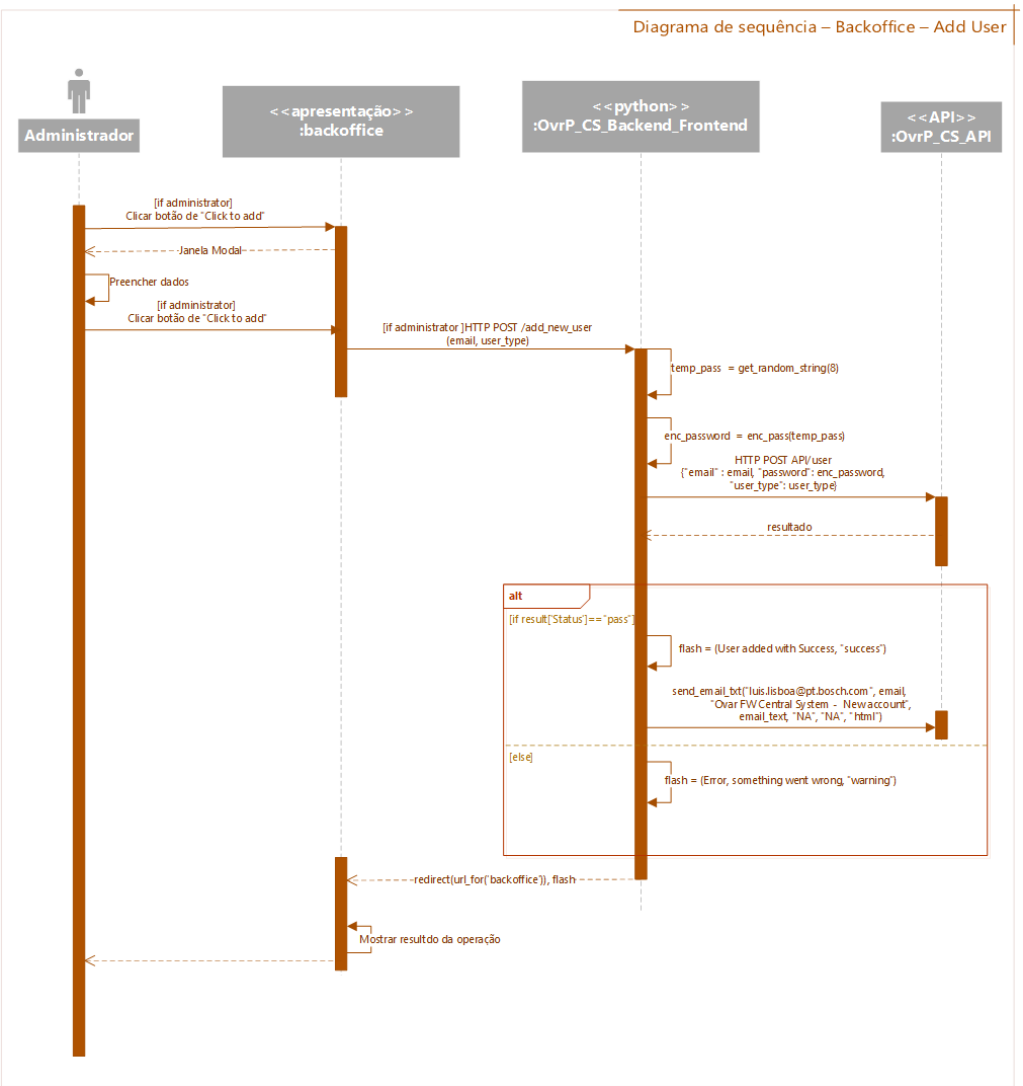


Figura C.9: Diagrama de sequência UML Add User `Users List`

C.2.6.2 Edit

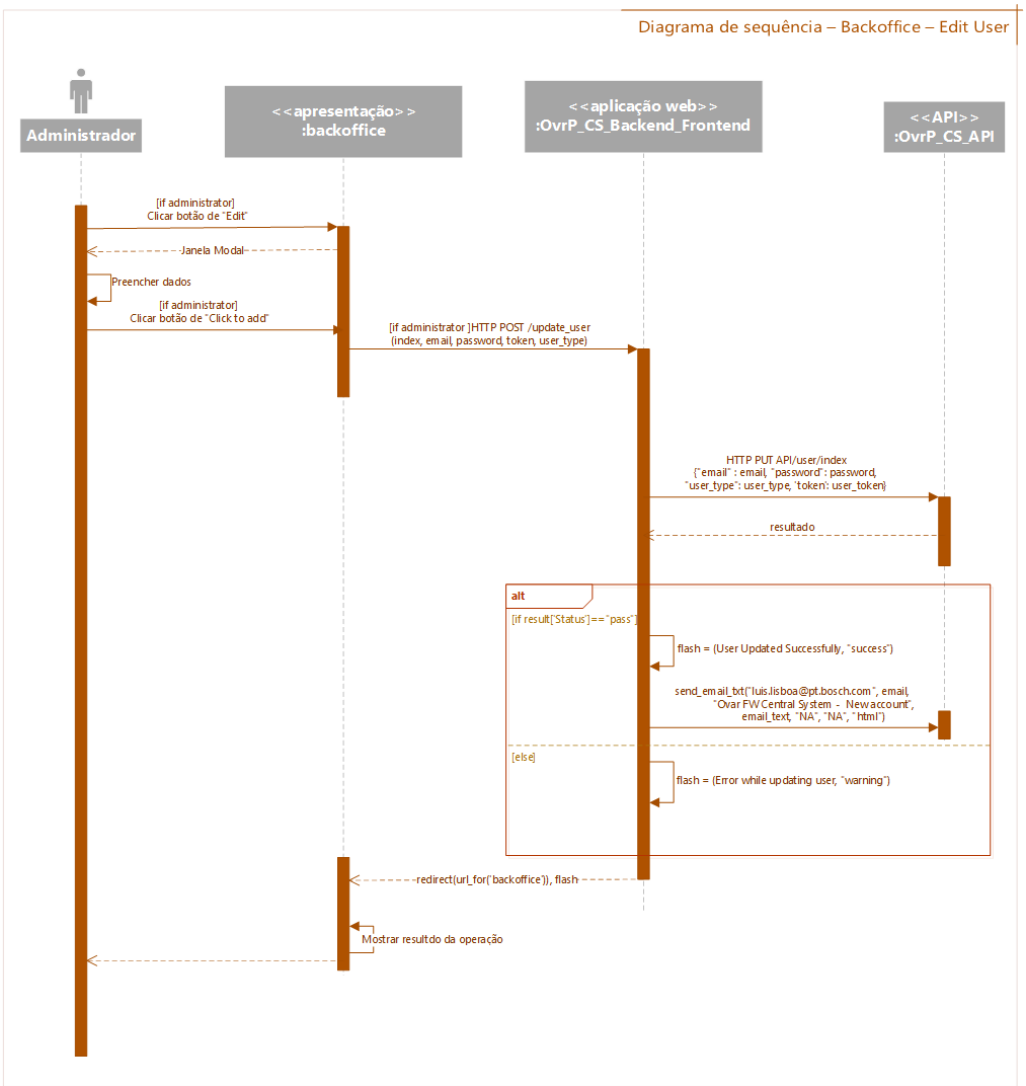


Figura C.10: Diagrama de sequência UML Edit User `Users List`

C.2.6.3 Delete

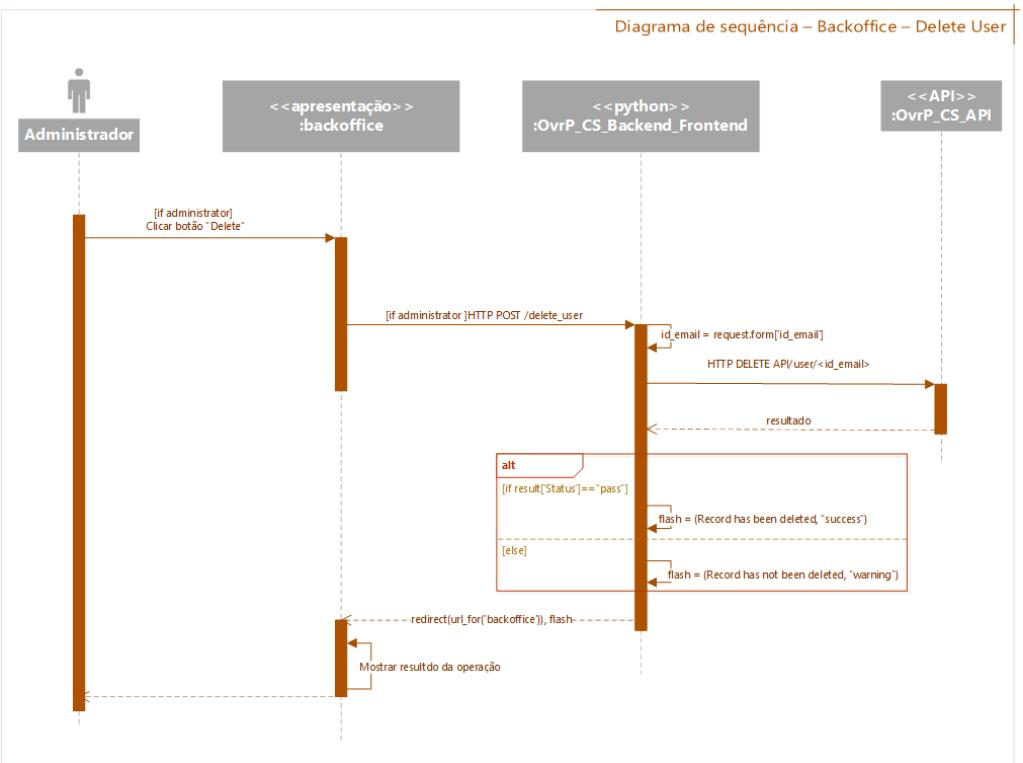


Figura C.11: Diagrama de sequência UML Delete User `Users List`

C.2.7 Menu OvrP ITM Machines

C.2.7.1 Add

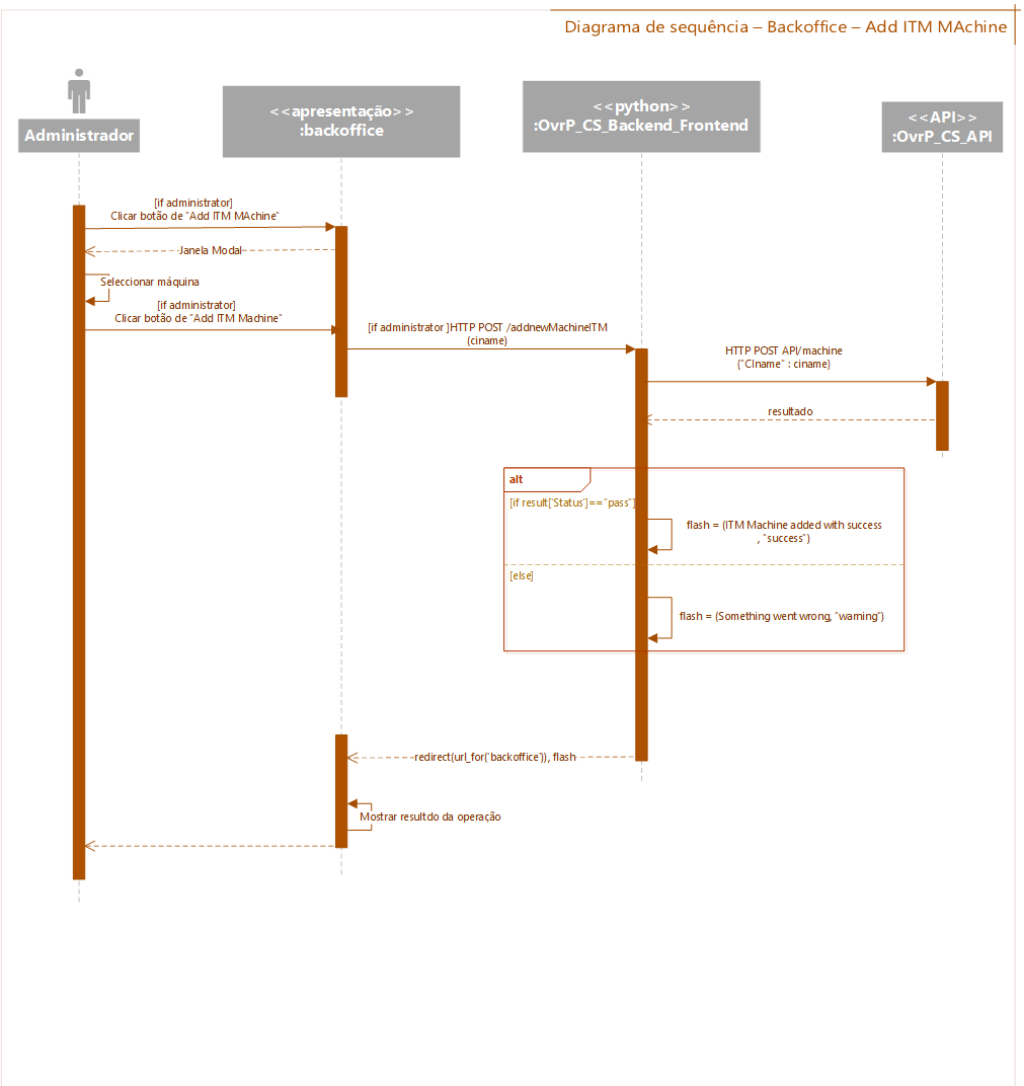


Figura C.12: Diagrama de sequência UML Add ITM Machine `Machine List`

C.2.7.2 Delete

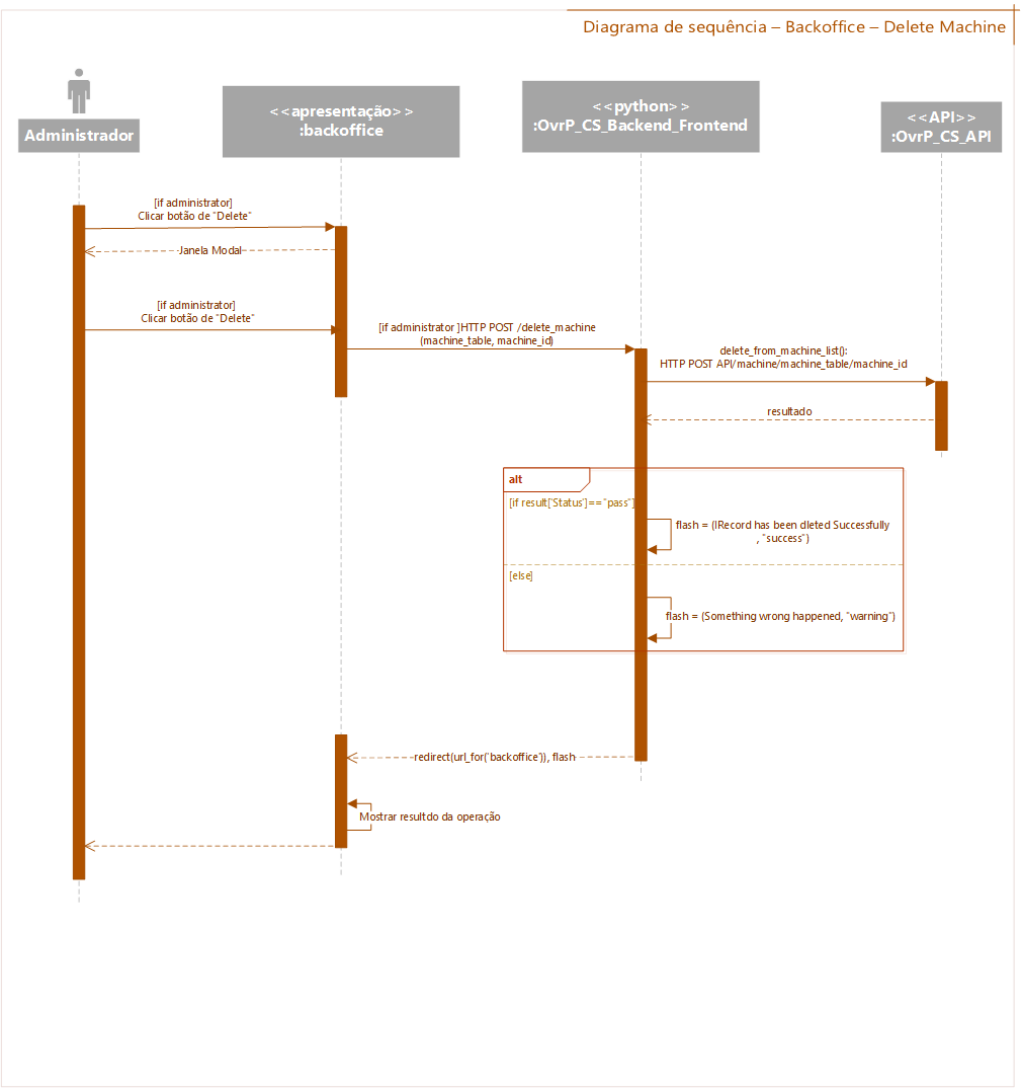


Figura C.13: Diagrama de sequência UML Delete ITM Machine `Machine List`

C.2.8 Menu Non OvrP ITM Machines

C.2.8.1 Add

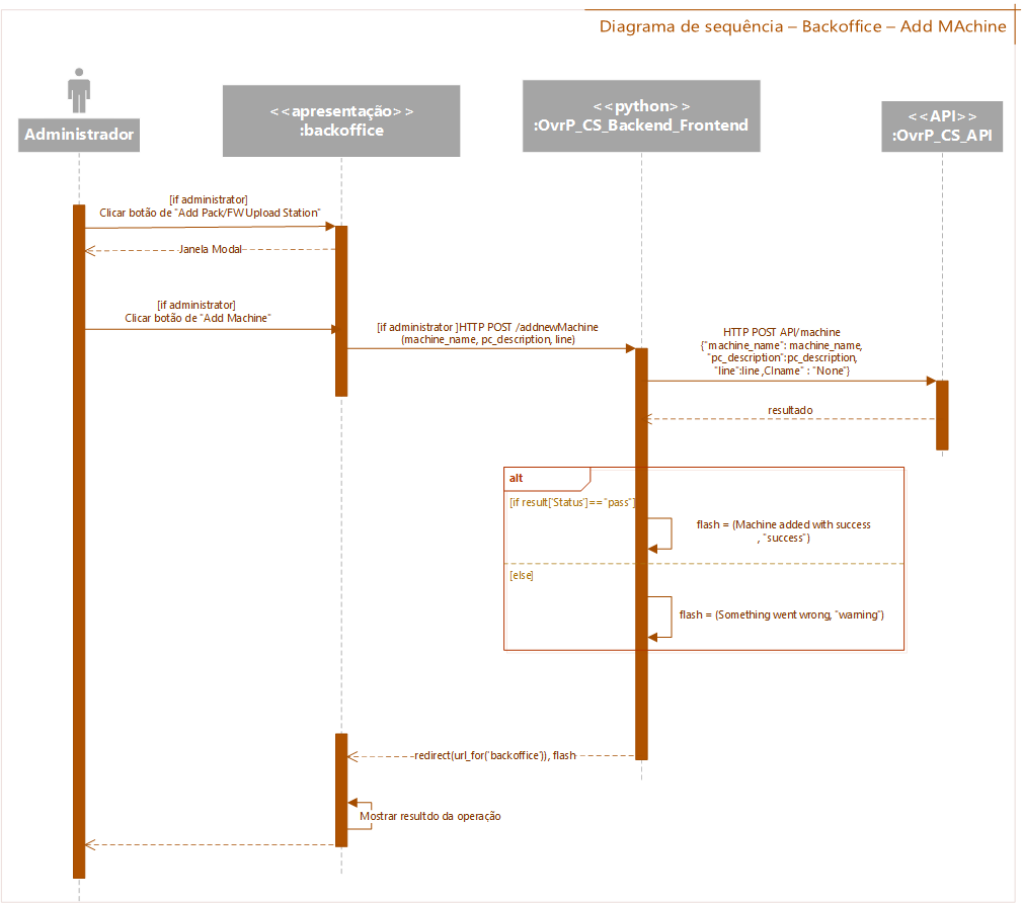


Figura C.14: Diagrama de sequência UML Add Non ITM Machine `Machine List`

C.2.8.2 Edit

C.15.

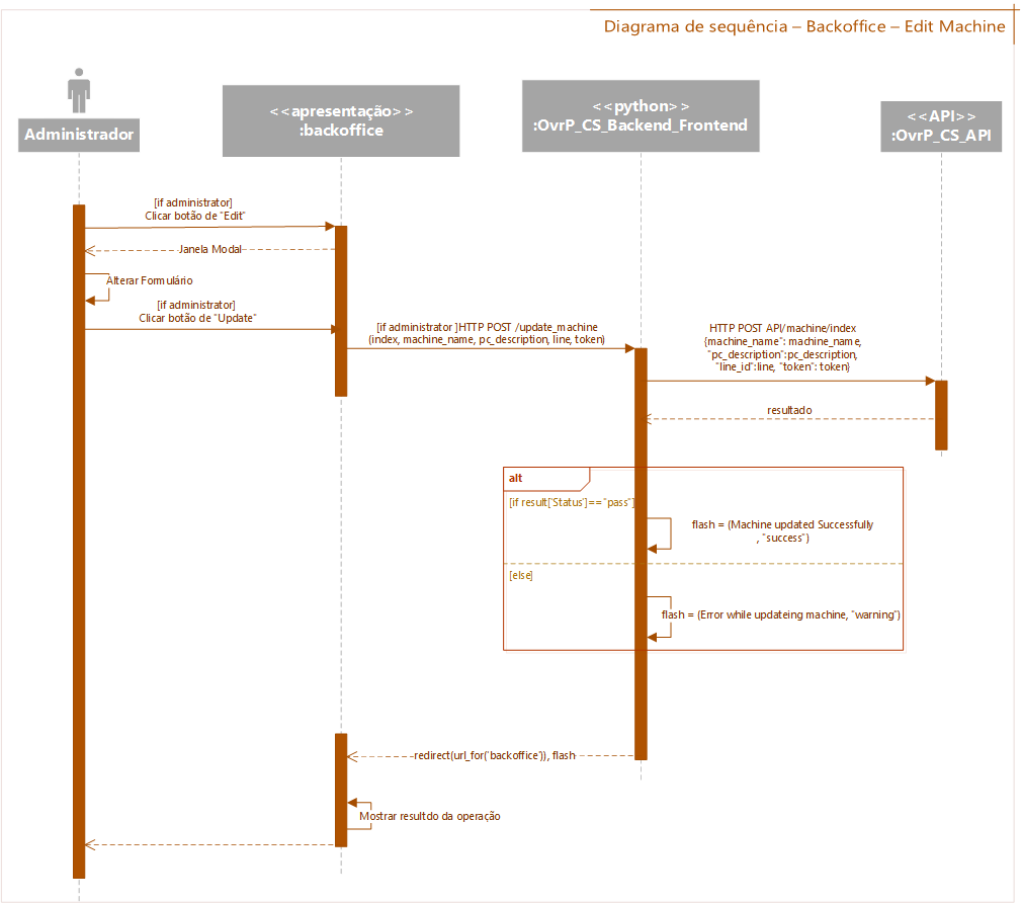


Figura C.15: Diagrama de sequência UML Edit Non ITM Machine `Machine List`

Anexo D

Testes de Desempenho

D.1 API REST

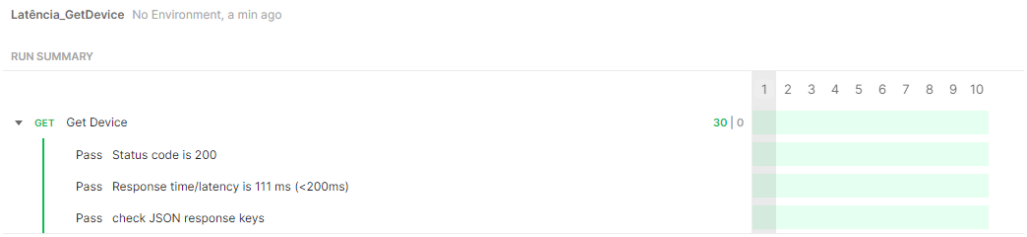


Figura D.1: Teste Latência HTTP GET /device/F.01U.321.597

D.2 Back-office

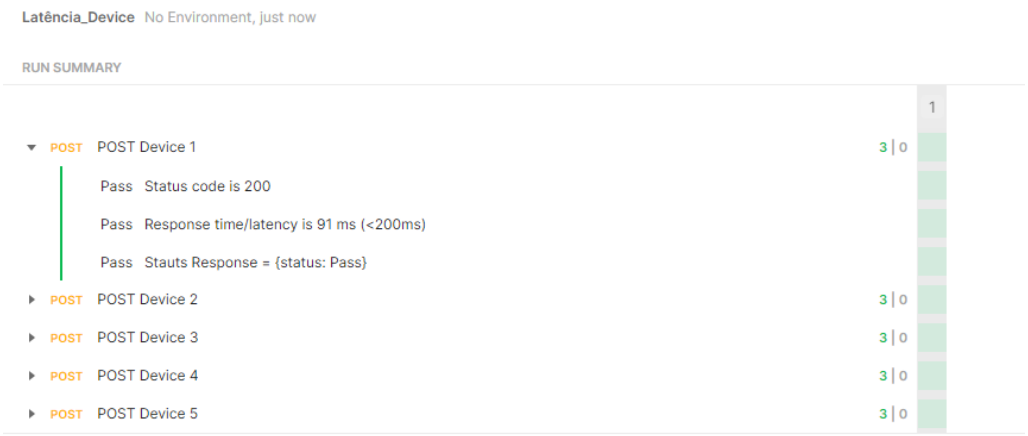


Figura D.2: Teste latência HTTP POST /device

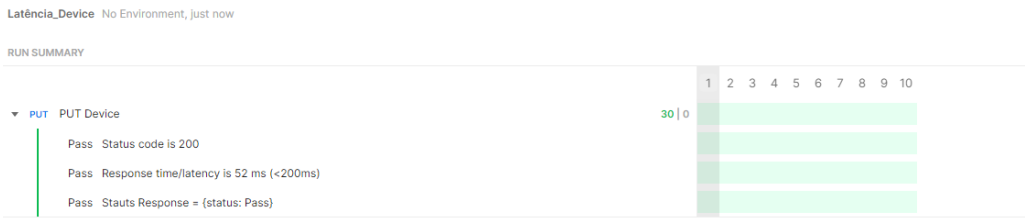


Figura D.3: Teste latência HTTP PUT /device/450

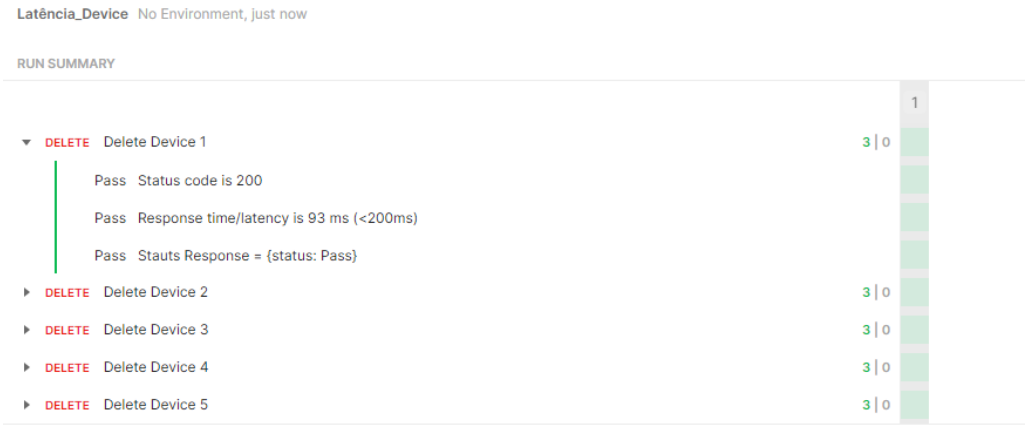


Figura D.4: Teste latência HTTP PUT /device/450...454

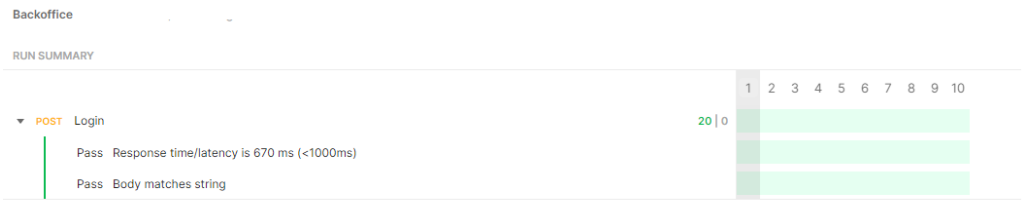


Figura D.5: Teste latência *Back-office* `Sign In`

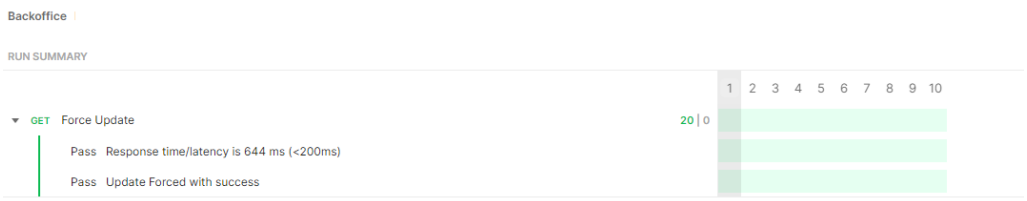


Figura D.6: Teste latência *Back-office* Forçar Atualização

Anexo E

Inquérito de Usabilidade

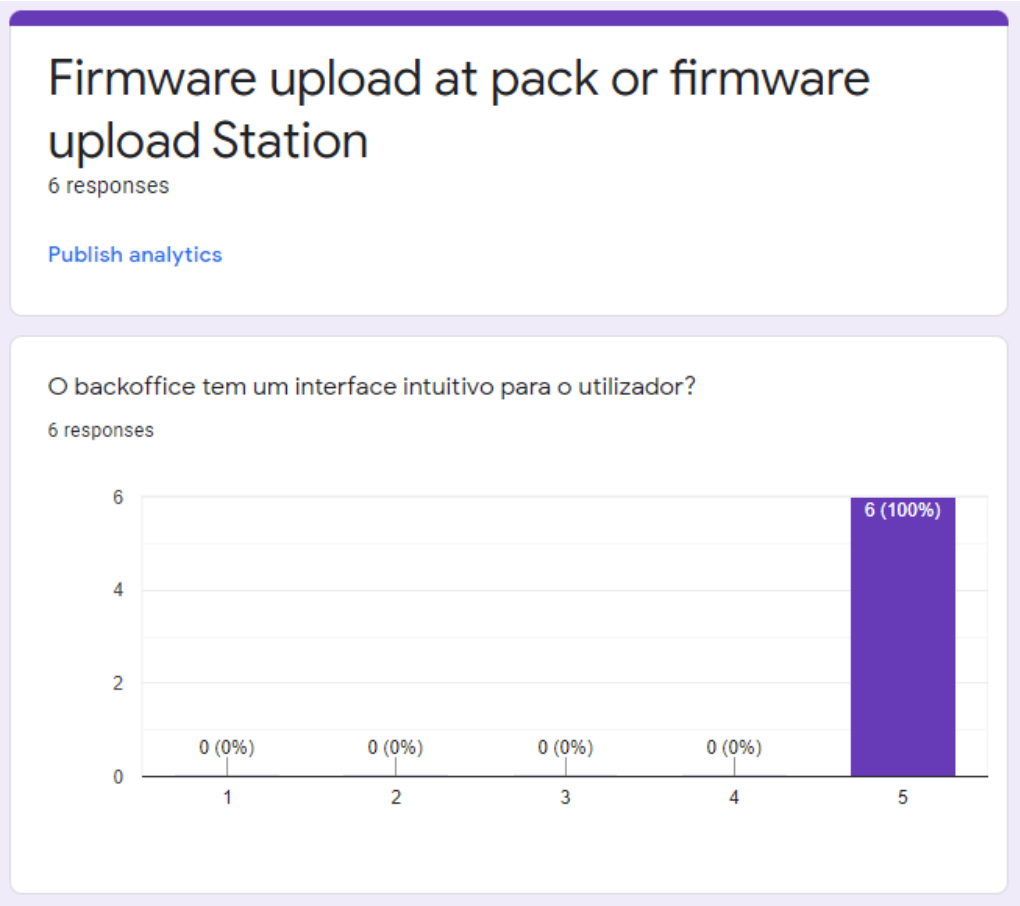


Figura E.1: Pergunta 1

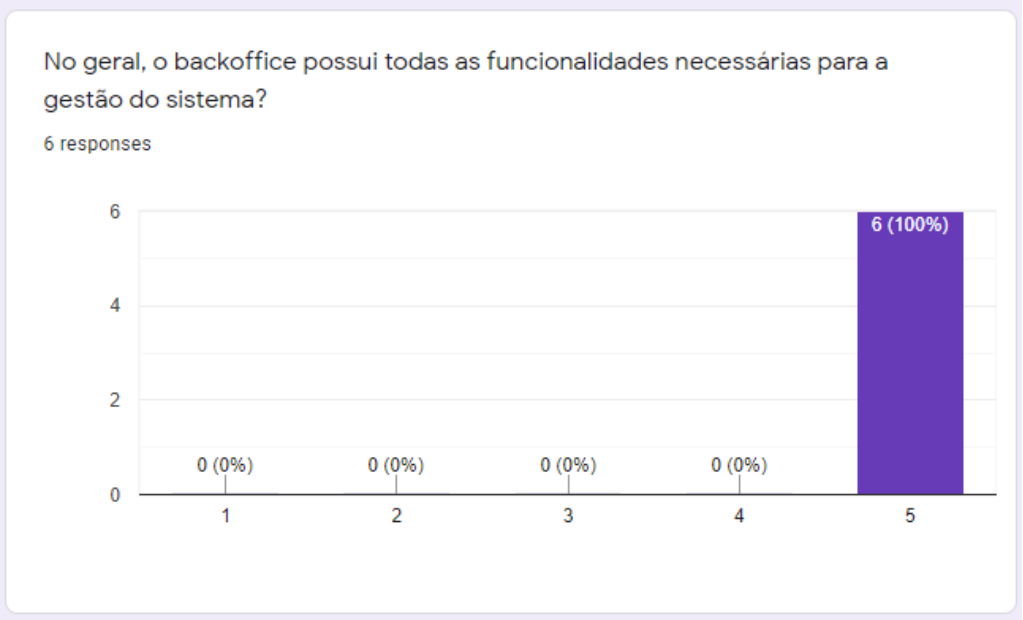


Figura E.2: Pergunta 2

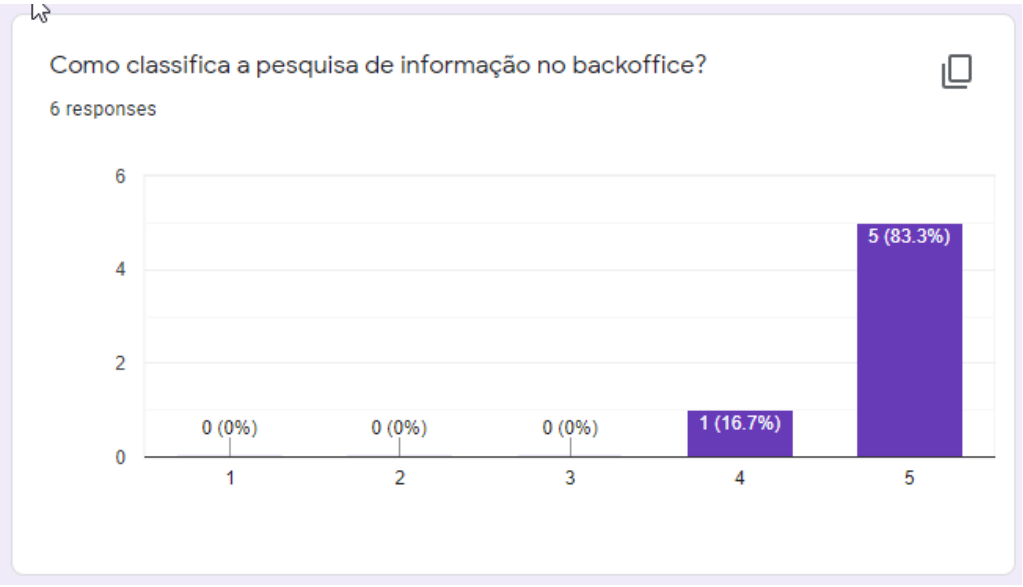


Figura E.3: Pergunta 3

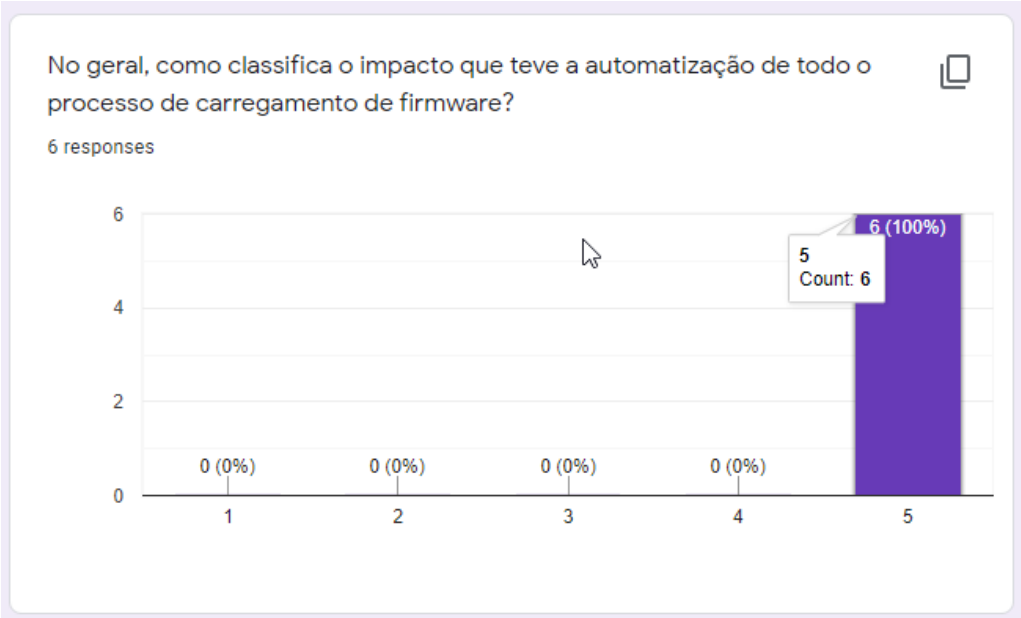


Figura E.4: Pergunta 4

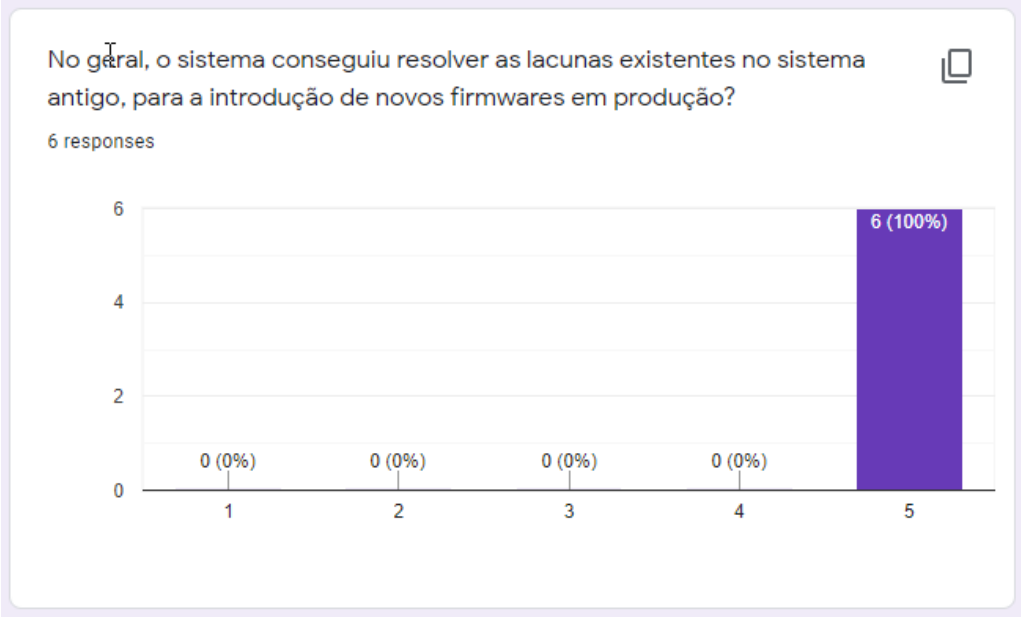


Figura E.5: Pergunta 5

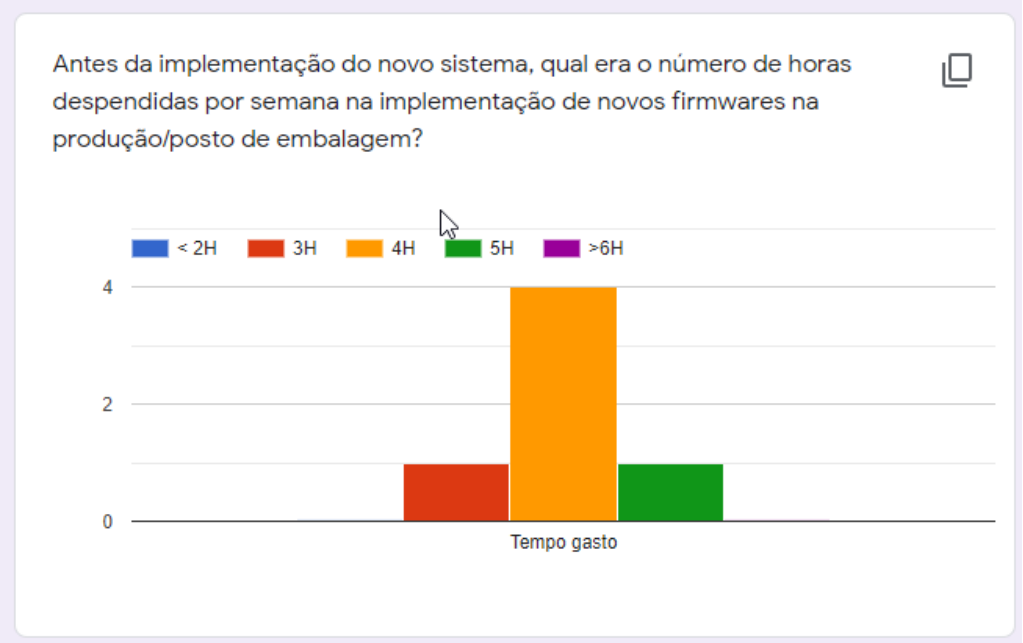


Figura E.6: Pergunta 6

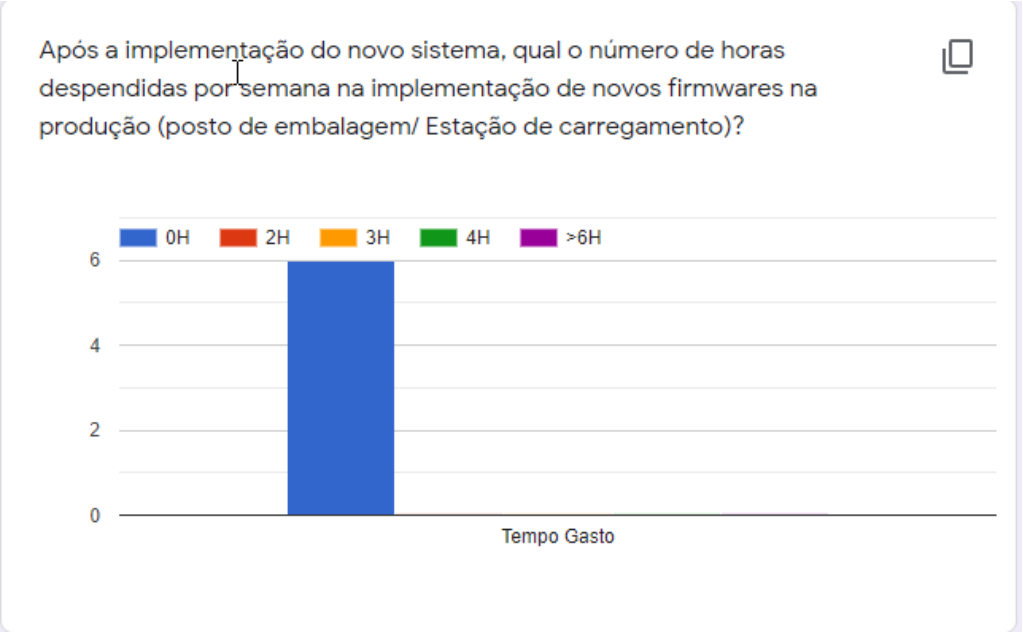


Figura E.7: Pergunta 7