



INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA (ISEL)

DEPARTAMENTO DE ENGENHARIA ELETRÓNICA E DE
TELECOMUNICAÇÕES E COMPUTADORES (DEETC)

LEIM

LICENCIATURA EM ENGENHARIA INFORMÁTICA E MULTIMÉDIA

UNIDADE CURRICULAR DE PROJETO

**osTicket-SINCRO – Implementação de um Sistema de
"Ticketing" para a Rede SINCRO-ANSR**

Rodrigo Costa 48633

Rodrigo Bugalhão 48961

Orientadores

Professor Doutor Carlos Gonçalves

Professor Doutor Tiago M. Dias

julho, 2024

Resumo

O projeto *osTicket-SINCRO – Implementação de um Sistema de Ticketing para a Rede SINCRO* resulta de uma colaboração entre o *Instituto Superior de Engenharia de Lisboa* (ISEL) e a *Autoridade Nacional de Segurança Rodoviária* (ANSR), que visa a contribuir para o combate à sinistralidade rodoviária, favorecendo a segurança pública ao melhorar a eficiência na manutenção dos radares de controle de velocidade, reduzindo assim a ocorrência de acidentes nas estradas.

O *Sistema Nacional de Controlo de Velocidade* (SINCRO), gerido pela ANSR, monitoriza e gera os equipamentos de controle de velocidade em Portugal. A manutenção destes equipamentos é realizada por prestadores de serviços, cada um com seu próprio sistema de gestão. O sistema de monitorização envia informações sobre problemas com os radares para os prestadores, solicitando a abertura de **tickets**.

Um dos principais desafios enfrentados é o trabalho duplicado pelos prestadores de serviços, que têm de gerir **tickets** nos seus próprios sistemas e no sistema da ANSR. Para resolver isso, está a ser desenvolvido, por outro grupo, um projeto relacionado, "osTicket-API", que criará uma *API REST* para automatizar a criação de **tickets** no *osTicket* a partir dos sistemas dos prestadores.

Atualmente, a ANSR não tem maneira de estruturar as informações nos **tickets**, por isso, os problemas nos radares são descritos em texto corrido, o que dificulta a sua interpretação. O nosso projeto foca-se em estruturar os dados dos **tickets** no *osTicket*, garantindo a inserção organizada das informações, seja de forma automática pela *API* ou manualmente, melhorando a análise e a manutenção dos equipamentos de segurança rodoviária. O objetivo será atingido com a criação de um *plugin* para o *osTicket*.

Palavras-chave: Sistema de Ticketing, *osTicket*, SINCRO, ANSR, Segurança Rodoviária, *API REST*, Manutenção de Radares.

Abstract

The project *osTicket-SINCRO – Implementation of a Ticketing System for the SINCRO Network* is a product of the collaboration between the *Instituto Superior de Engenharia de Lisboa* (ISEL) and the *Autoridade Nacional de Segurança Rodoviária* (ANSR), aiming to contribute to road safety by improving the efficiency of speed radar maintenance, thereby reducing accidents.

The *National Speed Control System* (SINCRO), managed by ANSR, monitors and manages speed control equipment in Portugal. Maintenance of this equipment is carried out by service providers, each with their own management system. The monitoring system sends information about radar issues to the providers, requesting the opening of **tickets**.

One of the main challenges faced is the duplicated work by service providers, who have to manage **tickets** in their own systems and in the ANSR system. To address this, a related project, "osTicket-API," is being developed, by another group, to create a *REST API* that automates ticket creation in **osTicket** from the providers' systems.

Currently, ANSR has no way to structure the information in the **tickets**, so radar issues are described in plain text, making interpretation difficult. Our project focuses on structuring the **ticket** data in **osTicket**, ensuring organized information entry, either automatically via the **API** or manually, improving the analysis and maintenance of road safety equipment. This will be achieved by creating a *plugin* for **osTicket**.

Keywords: Ticketing System, osTicket, SINCRO, ANSR, Road Safety, REST API, Radar Maintenance.

Agradecimentos

Gostaríamos de expressar aqui a nossa gratidão, a todas as pessoas que nos apoiaram neste percurso de aprendizagem.

Agradecemos aos docentes, pelo conhecimento valioso que nos transmitiram e pela enorme disponibilidade de nos atender qualquer dúvida.

Estendemos os nossos agradecimentos, com maior destaque, aos orientadores de projeto, Doutor Carlos Gonçalves e Doutor Tiago M. Dias, pelo apoio que nos deram durante a realização deste projeto. Os seus conselhos foram cruciais para o sucesso deste trabalho.

Queremos mencionar também, os amigos e família que nos apoiaram incansavelmente durante esta jornada, e celebrar especialmente todos os colegas que se tornaram amigos.

Gostaríamos de expressar o nosso agradecimento mútuo pelo apoio e colaboração ao longo deste projeto. Trabalhar juntos foi uma experiência enriquecedora, e valorizamos a dedicação e o empenho de cada um para alcançar os nossos objetivos.

Por fim agradecemos ao ISEL, por possibilitar toda esta experiência e nos valorizar.

*Rodrigo Costa
Rodrigo Bugalhão*

Índice

Resumo	i
Abstract	iii
Agradecimentos	v
Índice	vii
Lista de Figuras	ix
Lista de Listagens	xi
Lista de Acrônimos	xiii
1 Introdução	1
1.1 Contributos	3
1.2 Organização do Relatório	4
2 Análise e Estudo do Sistema osTicket	5
2.1 osTicket	5
2.2 Tickets	7
2.2.1 Composição e Estrutura dos Tickets	9
2.3 Plugins no osTicket	10
2.3.1 O que é um <i>Plugin</i>	10
2.3.2 Como Construir um <i>Plugin</i>	11
3 Modelo Proposto	19
3.1 Requisitos Funcionais	19
3.2 Requisitos Não Funcionais	23

3.3	Casos de Utilização	24
3.4	Arquitetura Proposta	27
3.4.1	O que é um <i>Patch</i>	29
4	Implementação do Modelo	31
4.1	Desenvolvimento do Plugin	31
4.1.1	Descrição do Plugin Inicial	32
4.2	Planeamento da Implementação	32
4.2.1	Resumo das Funcionalidades do Plugin	32
4.3	Desenvolvimento do Plugin <i>Radares Details</i>	32
4.3.1	Codificação do Plugin	33
4.3.2	Implementação da página de configuração	33
4.3.3	Implementação do Fluxo de Ativação e Desativação . .	36
4.3.4	Criação dos Campos para Adicionar a um Formulário .	42
4.3.5	Implementação da Funcionalidade de <i>Backup</i>	45
4.3.6	Implementação de Campos de Filtragem através de AJAX	48
5	Demonstrador do Plugin	53
5.1	Ambiente de Desenvolvimento	53
5.2	Instalação	54
5.3	Desinstalação	57
5.4	Demonstração de Utilização Manual	58
5.5	Demonstração de Utilização do Plugin via <i>API</i>	63
6	Conclusões e Trabalho Futuro	67
6.1	Conclusões	67
6.2	Trabalho Futuro	68
Bibliografia		68
A	Guia Completo do Plugin	71

Listas de Figuras

1.1	Arquitetura Simplificada do projeto SINCRO	2
2.1	Modelo EA dos formulários	7
2.2	Estrutura da pasta do plugin	12
2.3	Página de instalação do plugin	14
2.4	Página de configuração do Plugin	15
2.5	Exemplo de entradas na Tabela <code>ost_config</code>	16
2.6	Exemplo de entradas na Tabela <code>ost_plugin</code> na base de dados	18
2.7	Exemplo de entradas na Tabela <code>ost_plugin_instance</code> na base de dados	18
3.1	Casos de Utilização - Criação de Tickets	25
3.2	Casos de Utilização - Gestão do Plugin	26
3.3	Diagrama UML do projeto	28
4.1	Página de configuração do plugin	34
4.2	Página de configuração do plugin	34
4.3	Erro ao não selecionar um formulário	35
4.4	Erro de privilégios insuficientes	35
4.5	Entradas na Tabela <code>ost_config</code> da configuração do plugin . .	36
4.6	Entrada na Tabela <code>ost_list</code> da lista de cabines	39
4.7	Entradas na Tabela <code>ost_list_items</code> com informações das cabines do projeto SINCRO	39
4.8	Página do <code>osTicket</code>	39
4.9	Página de <i>login</i> para agentes <code>osTicket</code>	40
4.10	Página com as informações da empresa	40
4.11	<code>ticket</code> com os novos campos	45
4.12	Entradas na Tabela <code>ost_list_items</code>	47

4.13 Atualização do <code>listId</code> da Tabela de <code>ost_backup_form_field</code> para a Tabela <code>ost_form_field</code>	47
4.14 Exemplo de seleção dos campos de filtragem	49
5.1 página Admin Panel Plugins	54
5.2 página Plugin Installation	54
5.3 página Plugin Installed	55
5.4 página Add New Instance	55
5.5 página Instance	56
5.6 página Instance Configuration	56
5.7 página Enable Plugin	57
5.8 Seleção da cabine	58
5.9 Abrir <code>ticket</code>	59
5.10 <code>Ticket</code> guardado	60
5.11 Tabela <code>ost_form_field</code>	60
5.12 Tabela <code>ost_form_entry_values</code>	61
5.13 Tabela <code>ost_form_entry</code>	61
5.14 <code>ticket</code> depois de desativar o <code>plugin</code>	62
5.15 Tabelas de <code>backup</code>	62
5.16 <code>ticket</code> reposto	63
5.17 Pedido <code>http</code> no <i>Postman</i>	64
5.18 Resposta no <i>Postman</i>	64
5.19 <code>ticket</code> criado através da <i>API</i> do Projeto 15	65

List a de Listagens

1	Exemplo de ficheiro <code>plugin.php</code>	13
2	Exemplo de ficheiro <code>Config.php</code>	14
3	Exemplo de ficheiro <code>PluginExemplo.php</code>	17
4	Métodos para a gestão de estados do plugin	36
5	Método <code>isActive()</code>	37
6	Método <code>handleActivationChange(currentlyActive)</code>	37
7	Método <code>enablePlugin()</code>	38
8	Método <code>disablePlugin()</code>	41
9	Método <code>bootstrap()</code>	41
10	Chamada do método <code>keepPluginInfo()</code>	48
11	Patch no método <code>getFormName()</code>	49
12	Algoritmo para o <i>Patch</i> da classe <code>class.forms.php</code>	50
13	Função AJAX	50
14	Subscrição ao Sinal AJAX	51
15	Registro de Rotas AJAX	51

Lista de Acrônimos

AJAX Asynchronous JavaScript and XML

ANSR Autoridade Nacional de Segurança Rodoviária

API Application Programming Interface

CRM Customer Relationship Management

EA Entidade-Associação

ERP Sistema integrado de gestão empresarial

HTTP Hypertext Transfer Protocol

ISEL Instituto Superior de Engenharia de Lisboa

JSON JavaScript Object Notation

LCV Local de Controlo de Velocidade

LCVI Local de Controlo de Velocidade Instantânea

LCVM Local de Controlo de Velocidade Média

LEIM Licenciatura em Engenharia Informática e Multimédia

PHP Hypertext Preprocessor

SCP Staff Control Painel

SIGET Sistema de Gestão de Eventos de Trânsito

SINCRO Sistema Nacional de Controlo de Velocidade

SQL Structured Query Language

UML Unified Modeling Language

URL Uniform Resource Locator

Capítulo 1

Introdução

Considerando os projetos de Investigação e Desenvolvimento (I&D) entre o Instituto Superior de Engenharia de Lisboa (ISEL) e a Autoridade Nacional de Segurança Rodoviária (ANSR), surgiu a necessidade de desenvolver uma solução para um problema atual do mercado. O projeto da Unidade curricular de Projeto Final do curso de Licenciatura em Engenharia Informática e Multimédia (LEIM) do ISEL, intitulado *osTicket-SINCRO – Implementação de um Sistema de Ticketing para a Rede SINCRO*, resultou desta colaboração. Esta iniciativa pretende não só responder às necessidades imediatas do mercado, mas também contribuir para o interesse público através da redução da sinistralidade rodoviária. A ANSR, enquanto entidade pública do Estado, desempenha um papel crucial na promoção da segurança rodoviária, e este projeto está alinhado com a sua missão principal de garantir a segurança nas estradas.

O Sistema Nacional de Controlo de Velocidade (SINCRO), gerido pela ANSR, é responsável pela gestão e monitorização dos equipamentos de controlo de velocidade em Portugal. Estes equipamentos são monitorizados pelo *Zabbix*, correspondente ao módulo *SIGET-M*. A manutenção dos equipamentos é realizada por prestadores de serviços, cada um com seu próprio sistema de gestão. O sistema de monitorização envia informações para cada prestador, indicando problemas e solicitando a abertura de **tickets**. Paralelamente, a ANSR deseja uma visão integrada do sistema de monitorização e do estado dos **tickets** abertos pelos diferentes fornecedores, necessitando assim de um sistema de **ticketing** integrado para uma visão consolidada dos problemas.

Conforme ilustrado na Figura 1.1, o projeto está estruturado em vários módulos: o módulo *SIGET-SM*, parte do sistema informático *SIGET*, visa suportar a gestão do serviço de manutenção de todos os Locais de Controlo de Velocidade (LCV) do SINCRO, seja de controlo de velocidade instantânea (LCVI) ou de velocidade média (LCVM). Este módulo gera pedidos de intervenção solicitados pela ANSR e pelas entidades responsáveis pelas ações de manutenção preventiva e corretiva. O módulo *SIGET-M* é responsável pela monitorização dos radares, utilizando a plataforma de código aberto Zabbix. Quando um problema é detetado, o *SIGET-M* notifica os prestadores de serviço, solicitando a abertura de um **ticket** no módulo *SIGET-SM* (**osTicket**) e o registo do problema nos seus próprios sistemas de **ticketing**.

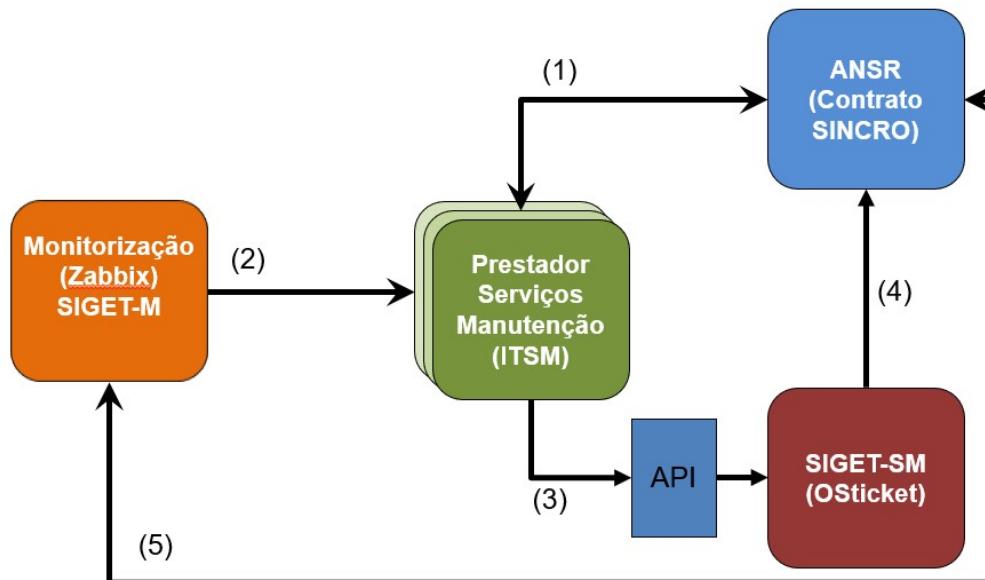


Figura 1.1: Arquitetura Simplificada do projeto SINCRO:

1. Interação com Prestador - pedidos, incidentes, relatórios, pontos de situação do serviço;
2. Alertas Monitorização;
3. Replicação de Abertura e Fecho de Tickets;
4. Relatórios / Vistas On-line dos Serviços;
5. Relatórios / Vistas On-line da Disponibilidade Rede;

Um dos principais desafios identificados foi o trabalho duplicado enfrentado pelos prestadores de serviço. Devido à falta de uma *API* no sistema de **tickets** do *osTicket*, os prestadores precisavam de gerir **tickets** nos seus próprios sistemas e no sistema da ANSR. Para resolver este problema, os colegas do Projeto 15, intitulado *osTicket-API – Desenvolvimento de uma API REST para o software osTicket*, estão a desenvolver uma *API* que permitirá a criação automática de **tickets** no *osTicket* a partir dos sistemas dos prestadores de serviço. Este **plugin** da *API* simplificará o processo, permitindo que os diversos prestadores de serviço repliquem, de forma programática e quase automática, os seus sistemas privados no sistema único da ANSR. Cada prestador poderá comunicar o estado das resoluções dos problemas ao sistema de **ticketing** da ANSR através de uma *API* unificada.

No entanto, atualmente a inserção de dados nos **tickets** é realizada através de texto corrido, de forma não estruturada, o que pode dificultar a análise estatística dos problemas/tickets criados no *osTicket*. O principal objetivo do nosso trabalho é estruturar os dados dos **tickets**, desenvolvendo uma funcionalidade no *osTicket* que garanta a inserção organizada das informações. De um modo geral, esses **tickets** serão abertos e preenchidos automaticamente pela *API* do Projeto 15, mas também poderá haver casos de preenchimento manual. Esta funcionalidade facilitará, futuramente, a análise eficiente dos dados dos **tickets** para identificar padrões de avarias, como quais radares avariam mais e em quais zonas ou distritos, melhorando a manutenção dos equipamentos e a gestão da segurança rodoviária.

O desenvolvimento de uma funcionalidade que permita a estruturação dos dados dos **tickets** é essencial para otimizar a manutenção dos equipamentos de segurança rodoviária e fornecer uma visão unificada e estruturada para a ANSR, melhorando a eficiência operacional e a análise de dados.

1.1 Contributos

Como resultado do projeto, foi desenvolvido o **plugin Radares Details**, que está disponível no repositório [1]. Este **plugin** permite a criação de **tickets** estruturados para o reporte de avarias dos radares. O mesmo oferece uma lista de cabines para a seleção da cabine com problemas, uma caixa de seleção para os componentes do radar que apresentaram falhas e, finalmente, um

campo para a descrição detalhada da avaria. Desta forma, o campo de texto livre é utilizado apenas para fornecer informações adicionais relevantes sobre as avarias.

1.2 Organização do Relatório

Este relatório está organizado em seis capítulos, cada um detalhando uma parte essencial do desenvolvimento e implementação do projeto *osTicket-SINCRO – Implementação de um Sistema de Ticketing para a Rede SINCRO*. O **Capítulo 2 - Análise e Estudo do Sistema osTicket** introduz o leitor ao domínio da gestão de tickets, explicando conceitos e técnicas-chave. O **Capítulo 3 - Modelo Proposto** apresenta a proposta de uma solução de implementação do plugin para o *osTicket*. O **Capítulo 4 - Implementação do Modelo** detalha a implementação do modelo proposto no Capítulo 3. O **Capítulo 5 - Demonstrador do Plugin** demonstra a utilização do plugin *Radares Details*, mostrando como configurá-lo para uma utilização bem-sucedida. O **Capítulo 6 - Conclusão e Trabalhos Futuros** apresenta considerações finais e delineia a nossa visão para o futuro do plugin *Radares Details*.

Capítulo 2

Análise e Estudo do Sistema osTicket

Neste capítulo, apresenta-se uma análise do sistema `osTicket` para compreender a sua estrutura interna e identificar pontos de integração essenciais para o desenvolvimento de `plugins`. Esta fase de estudo é fundamental para garantir uma implementação eficaz e alinhada com os objetivos do projeto. O capítulo começa pela Secção 2.1, onde se aborda o sistema `osTicket`, incluindo a sua arquitetura e interfaces principais. Na Secção 2.2, descreve-se o funcionamento e a estrutura dos `tickets` no `osTicket`. Em seguida, na Secção 2.3, discute-se a definição, importância e funções dos `plugins` no `osTicket`. Finalmente, na Secção 2.3.2, apresentam-se as etapas essenciais para o desenvolvimento de um `plugin` para o `osTicket`.

2.1 osTicket

O `osTicket` é uma plataforma de software de código aberto concebida para facilitar a gestão eficiente e colaborativa de `tickets` de suporte. Este sistema responde à necessidade, presente em diversas organizações, de centralizar a receção, a monitorização e a resolução de solicitações de suporte, sejam elas de utilizadores externos, como clientes, ou de utilizadores internos.

Desenvolvido em PHP e utilizando uma base de dados MySQL, o `osTicket` é uma aplicação web que suporta uma ampla variedade de processos de *Help Desk*. Entre as suas funcionalidades destacam-se a criação, gestão e encerramento de `tickets` através da sua interface web, bem como a capacidade de

personalização para atender às necessidades específicas de cada organização.

O **osTicket** é composto por duas interfaces principais:

- **Interface do Agente:** Esta interface é destinada aos agentes de suporte e permite a criação, edição e encerramento de **tickets**, além de outras funcionalidades avançadas para a gestão dos pedidos de suporte. Adicionalmente, possibilita a personalização da plataforma em diferentes áreas, como formulários e fluxos de trabalho.
- **Interface do Utilizador:** Destinada aos utilizadores finais, esta interface permite a criação de **tickets** e o acompanhamento do seu estado. A simplicidade desta interface facilita a interação do utilizador com o sistema.

A interação com o **osTicket** pode ser realizada de várias formas, incluindo formulários web, telefone, *e-mail* ou através de uma *API* [2], cuja versão atual é limitada, pois apenas permite a criação de tickets, não sendo possível fechar, reabrir ou editar tickets. Esta flexibilidade garante que os utilizadores e agentes possam interagir com o sistema de maneira conveniente e adaptável.

Uma das principais vantagens do **osTicket** em relação a outras soluções reside na sua natureza de código aberto. Esta característica permite uma customização extensiva utilizando novos **plugins**, possibilitando às organizações ajustar o sistema às suas necessidades específicas. Além disso, a plataforma beneficia de uma comunidade ativa de desenvolvedores que contribuem com melhorias contínuas e novos recursos, mantendo o sistema atualizado e relevante.

O **osTicket** destaca-se pela sua capacidade de integração com diversas ferramentas e sistemas, oferecendo adaptabilidade a um ambiente corporativo dinâmico. Embora as *APIs* do **osTicket** permitam algumas integrações, elas não cobrem todas as funcionalidades de gestão necessárias. No entanto, o sistema de **plugins** do **osTicket** permite que os desenvolvedores criem soluções personalizadas que se integram perfeitamente com o núcleo do sistema, garantindo estabilidade e segurança.

2.2 Tickets

Como referido na Secção 2.1, a base do osTicket são, fundamentalmente, os **tickets**. Por isso, é essencial compreender como funcionam no sistema e como são compostos e estruturados.

Um **ticket** no osTicket é essencialmente um conjunto de formulários utilizados para registar e monitorizar solicitações de suporte. Cada **ticket** está normalmente associado a pelo menos um formulário, e cada formulário é vinculado a um *help topic*. Um *help topic* funciona como uma categoria ou filtro que ajuda a organizar e categorizar os **tickets** de acordo com o tipo de suporte solicitado.

Para a criação dos campos personalizados, que foi a base da funcionalidade do nosso **plugin**, tivemos que entender quais Tabelas da base de dados eram necessárias para essa implementação. Esta análise foi crucial para garantir que a adição de novos campos através do **plugin** fosse realizada de maneira estruturada e eficiente.

Embora todas as alterações, como a adição de novos campos e *help topics*, possam ser feitas através da interface do osTicket, focámo-nos no *backend* do sistema para realizar estas modificações via **plugin**, dado que o objetivo do projeto é instalar e aplicar o nosso **plugin** num sistema osTicket diferente e já existente sem comprometer o mesmo.

A Figura 2.1 apresenta o modelo Entidade-Associação (EA) das Tabelas envolvidas na gestão dos **tickets**.

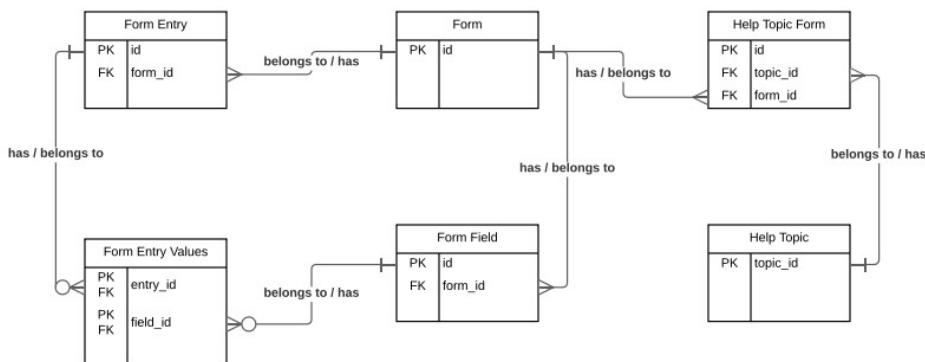


Figura 2.1: Modelo EA dos formulários

Como ilustra a Figura 2.1, é representado o modelo EA dos formulários, composto pelas seguintes entidades: Form Entry, Form, Form Field, Form Entry Values, Help Topic Form e Help Topic. Cada uma destas entidades está inter-relacionada, detalhando como os formulários e campos são utilizados para capturar e organizar as informações dos **tickets**. Este modelo cobre aspectos essenciais, como a associação de formulários a tópicos de ajuda (*help topics*) e a estrutura necessária para adicionar e gerir campos personalizados através do **plugin**.

As entidades apresentadas anteriormente e suas respectivas funções são:

- **ost_form**: Armazena as informações básicas sobre cada formulário, como o seu ID, tipo, título, instruções, notas e datas de criação e atualização. Esta Tabela é essencial para identificar e gerir os formulários existentes no sistema, sendo também crucial para a criação de novos formulários.
- **ost_form_entry**: Regista as entradas de formulários, ligando-as aos objetos correspondentes, no nosso caso, apenas a **tickets**. Contém campos como **form_id**, **object_id** e **object_type**. O campo **form_id** identifica a qual formulário a entrada pertence. O campo **object_id** refere-se ao identificador do objeto associado, como um ticket. O campo **object_type** indica o tipo de objeto, sendo 'T' para **tickets**.
- **ost_form_entry_values**: Armazena os valores específicos inseridos em cada campo de um formulário. Contém **entry_id**, **field_id**, **value** e **value_id**, permitindo localizar os dados submetidos em cada campo do formulário de cada ticket.
- **ost_form_field**: Define os campos individuais dentro de cada formulário, incluindo o ID do formulário, tipo de campo, **label**, nome, configuração e outras propriedades, contendo toda a informação relevante sobre o campo.
- **ost_help_topic**: Armazena os tópicos de ajuda, que são usados para categorizar e organizar os **tickets**. Cada tópico de ajuda pode ser associado a um ou mais formulários, facilitando a gestão dos **tickets**.

- `ost_help_topic_form`: Liga os tópicos de ajuda aos formulários, permitindo que múltiplos formulários sejam associados a um tópico de ajuda específico. Esta Tabela contém informações sobre quais formulários estão vinculados a quais tópicos de ajuda.

2.2.1 Composição e Estrutura dos Tickets

Nesta Secção, aborda-se a composição e estrutura dos **tickets**, referindo os principais componentes que os constituem.

Os formulários são componentes essenciais dos **tickets** e contêm os campos necessários para capturar as informações relevantes sobre uma solicitação de suporte. Estes campos podem variar desde campos de texto simples até listas de seleção, caixas de verificação e áreas de texto. A flexibilidade dos formulários permite que as organizações personalizem os campos de acordo com as suas necessidades específicas, garantindo que todas as informações relevantes sejam adequadamente registradas e monitorizadas.

Os *help topics* são utilizados para categorizar os **tickets** e facilitar a sua gestão. Cada *help topic* pode estar associado a um ou mais formulários, permitindo uma configuração flexível e adaptável. Por exemplo, um *help topic* denominado "Problemas de Rede" pode estar associado a um formulário que solicita informações específicas sobre a rede, enquanto um *help topic* "Problemas de Software" pode ter um formulário diferente que solicita detalhes sobre o *software* em questão. Esta estrutura facilita a organização e a categorização dos **tickets** de acordo com o tipo de suporte solicitado, tornando o processo de gestão mais eficiente.

O **osTicket** permite a criação de campos personalizados nos formulários dos **tickets**. Esta funcionalidade é crucial para garantir que todas as informações necessárias sejam capturadas de maneira estruturada e eficiente. Campos personalizados podem incluir, por exemplo, números de série de equipamentos, endereços de *IP*, versões de *software*, entre outros. A criação desses campos personalizados foi a base para o desenvolvimento do nosso **plugin**, exigindo uma análise detalhada das tabelas da base de dados envolvidas.

Embora seja possível adicionar campos personalizados diretamente através das funcionalidades do **osTicket**, o desenvolvimento de um **plugin** oferece vantagens significativas. Um **plugin** permite uma maior flexibilidade e ex-

tensibilidade, facilitando a manutenção e a atualização das funcionalidades sem necessidade de fazer essas adições manualmente, ou seja, de forma programática. Além disso, um **plugin** pode ser facilmente instalado ou removido, minimizando o impacto nas operações do sistema e permitindo personalizações específicas para diferentes necessidades sem comprometer a integridade do sistema principal.

2.3 *Plugins* no osTicket

Nesta Secção, aborda-se a definição e a importância dos **plugins** no contexto do **osTicket**, conforme apresentado na Secção 2.3.1, mas também os passos necessários para construir e integrar um **plugin** nesta plataforma, conforme descrito na Secção 2.3.2.

2.3.1 O que é um *Plugin*

No contexto do **osTicket**, um **plugin** é um componente de software adicional que expande ou melhora as funcionalidades do sistema principal. Os **plugins** são módulos independentes que podem ser instalados e configurados separadamente, permitindo que as organizações personalizem o **osTicket** para melhor atender às suas necessidades específicas sem alterar o núcleo do sistema.

Os **plugins** no **osTicket** podem desempenhar diversas funções, incluindo a integração com outros sistemas de software. Por exemplo, a integração com sistemas de *Customer Relationship Management* (CRM) como o Salesforce permite a sincronização de informações de clientes e **tickets**, facilitando um atendimento mais personalizado e eficiente. Integrações com plataformas de *e-commerce* como Shopify permitem que pedidos e problemas relatados pelos clientes sejam automaticamente convertidos em **tickets**, assegurando um acompanhamento eficaz. Além disso, sistemas de gestão empresarial (*ERP*) como o SAP podem ser integrados para sincronizar dados financeiros e logísticos, proporcionando uma visão coordenada entre os setores de suporte e operações.

Esta integração automatiza a troca de informações entre sistemas, reduzindo a necessidade de entrada manual de dados e melhorando a eficiência dos processos. Adicionalmente, os **plugins** podem ser utilizados para automati-

zar tarefas repetitivas dentro do `osTicket`, como a atribuição de `tickets` a agentes específicos com base em critérios predefinidos, o envio de notificações automáticas ou a atualização de estados de `tickets`.

Outra funcionalidade dos `plugins` é a personalização das funcionalidades do `osTicket`. Eles permitem que as organizações adicionem ou modifiquem funcionalidades para atender a requisitos específicos. Por exemplo, um `plugin` pode adicionar novos campos a um formulário de `ticket` ou criar relatórios personalizados. Adicionalmente, os `plugins` podem aumentar a segurança do `osTicket` através da implementação de medidas adicionais, como autenticação de dois fatores, verificações de segurança aprimoradas e monitorização de atividades suspeitas.

A utilização de `plugins` no `osTicket` oferece várias vantagens. A principal vantagem é a flexibilidade e personalização, permitindo que as organizações adaptem o sistema às suas necessidades específicas sem modificar o seu núcleo. Esta característica facilita a manutenção e atualização do sistema, garantindo que as personalizações permaneçam intactas após as atualizações do software principal. Embora a quantidade de `plugins` disponíveis não seja vasta e a contribuição de novos `plugins` pela comunidade não seja frequente, utilizar `plugins` já existentes ou desenvolver `plugins` personalizados que se integram perfeitamente ao sistema pode economizar tempo e recursos significativos para as organizações.

2.3.2 Como Construir um *Plugin*

Desenvolver um `plugin` para o `osTicket` envolve várias etapas essenciais. Primeiramente, é necessário identificar a necessidade específica que o `plugin` irá atender, com base nos requisitos da organização ou em funcionalidades adicionais desejadas. Em seguida, na fase de planeamento e design, definem-se as funcionalidades do `plugin` e a sua integração com o núcleo do `osTicket`. A implementação implica a codificação das funcionalidades planeadas, utilizando a arquitetura e as API do `osTicket` para garantir uma integração perfeita.

Antes de ser implementado em produção, o `plugin` deve ser rigorosamente testado para garantir o seu funcionamento correto e a não interferência com outras partes do sistema. Este processo inclui a verificação de compatibilidade, testes de desempenho e validação da segurança. Após os testes, o

plugin pode ser implementado no ambiente de produção, sendo necessária uma manutenção contínua para assegurar a compatibilidade com as atualizações do osTicket e a satisfação contínua das necessidades da organização.

Durante a fase de estudo, criámos um documento mais completo sobre a construção de plugins para o osTicket, que abrange funcionalidades avançadas e exemplos práticos. Este guia completo pode ser encontrado no Anexo A. No entanto, para o presente relatório, optámos por apresentar um guia mais simples, focado nas noções básicas e na criação de um plugin básico, de modo a facilitar a compreensão do processo. As instruções que serão apresentadas foram testadas e validadas para a versão 1.18 do osTicket.

O desenvolvimento de plugins para o osTicket apresenta desafios, sobretudo devido à escassez de documentação oficial. Esta Secção baseia-se em informações obtidas a partir de um *blog* não oficial [3], que descreve como criar um plugin no osTicket.

Para que um plugin seja reconhecido pelo osTicket, ele deve estar numa pasta contendo um ficheiro com as informações do plugin (`plugin.php`), um ficheiro de configuração (`Config.php`) e um ficheiro principal com a lógica do plugin (`PluginExemplo.php`). A pasta deve ser inserida na diretoria `[INSTALL_ROOT]/include/plugins` do osTicket, sendo `INSTALL_ROOT` a diretoria de instalação do osTicket. A estrutura da pasta do plugin deve seguir o formato apresentado na Figura 2.2:

```
[INSTALL_ROOT]
└── include
    └── plugins
        └── nome_do_plugin
            ├── plugin.php
            ├── Config.php
            └── PluginExemplo.php
```

Figura 2.2: Estrutura da pasta do plugin

Nos próximos parágrafos será explicado com detalhe o papel do ficheiro `plugin.php`, `Config.php` e do `PluginExemplo.php`.

Passo 1: Criar o Ficheiro ‘plugin.php’

O primeiro passo será criar um ficheiro `plugin.php` que contém a descrição e os dados essenciais do `plugin`. Na Listagem 1, apresenta-se um exemplo prático de como este ficheiro deve ser estruturado:

Listagem 1: Exemplo de ficheiro `plugin.php`

```
1 <?php
2 return array(
3     'id' => 'rodrigo:PluginExemplo',
4     'version' => '1.0',
5     'name' => 'Plugin Exemplo',
6     'author' => 'Rodrigo',
7     'description' => 'Plugin exemplo que faz alguma coisa.',
8     'url' => 'https://github.com/poctob/OSTEquipmentPlugin/
9                 wiki/Plugin-Development-Introduction',
10    'plugin' => 'PluginExemplo.php:PluginExemplo'
11 );
12 ?>
```

Este ficheiro define os elementos essenciais do `plugin`. O `id` é um identificador único para o `plugin`, como 'rodrigo:PluginExemplo'. A `version` indica a versão do `plugin`, por exemplo, '1.0'. O `name` é o nome do `plugin` que será exibido na interface, como 'Plugin Exemplo'. O `author` indica o nome do autor ou dos autores, por exemplo, 'Rodrigo'. A `description` fornece uma breve descrição do `plugin`. O `url` é o Uniform Resource Locator (URL) para o que o criador desejar, como por exemplo para mais documentação sobre o `plugin`. Finalmente, `plugin` refere-se ao ficheiro com a lógica do `plugin` e à classe a ser instanciada, como `PluginExemplo.php:PluginExemplo`.

A Figura 2.3 ilustra o resultado visual na interface do `osTicket` da configuração do ficheiro `plugin.php` apresentado na Listagem 1.

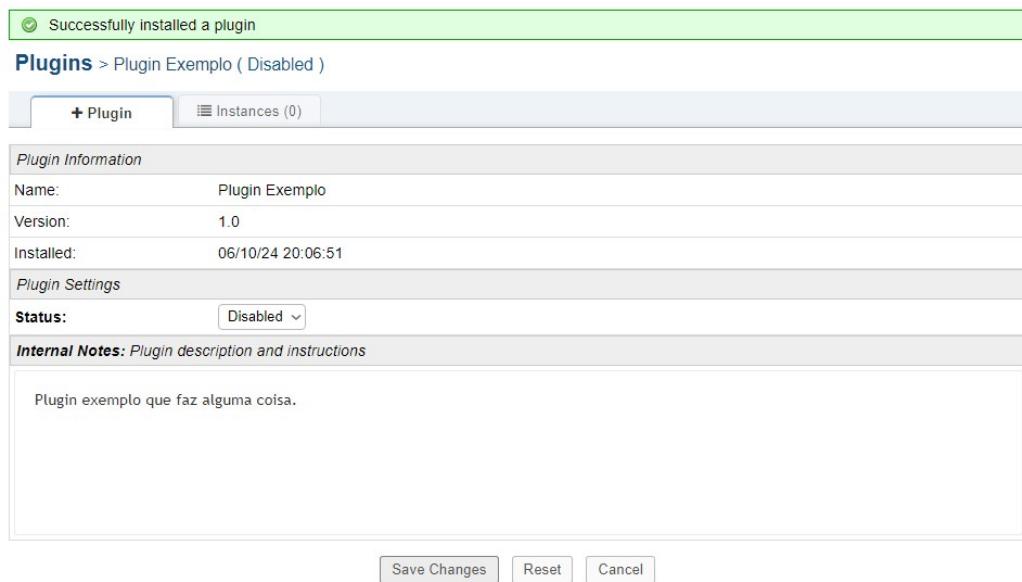


Figura 2.3: Página de instalação do plugin

Passo 2. Página de Configuração do Plugin

Como mencionado anteriormente, o plugin necessita de uma classe de configuração para ser reconhecido pelo osTicket. Na Listagem 2, apresenta-se um exemplo prático da estrutura que esta classe deve adotar.

Listagem 2: Exemplo de ficheiro Config.php

```

1 class PluginExemploConfig extends PluginConfig {
2     function getOptions() {
3         return array(
4             'plugin_exemplo_backup' => new BooleanField([
5                 'id' => 'plugin_exemplo_backup',
6                 'label' => 'Fazer Backup',
7                 'configuration' => array(
8                     'desc' => 'Ativar ou desativar backups automáticos.'
9                 )
10            ],
11        );
12    }
13
14    function pre_save(&$config, &$errors) {
15        global $msg;
16        // Guardar a configuração antes de salvar
17        ...
18        $this->set('plugin_exemplo_backup', $config['plugin_exemplo_backup']);
19        ...
20    }
21    ...

```

22 }

A classe `PluginExemploConfig` deriva de `PluginConfig`, que se encontra localizada em `[INSTALL_ROOT]/include/class.plugin.php`. A função `getOptions()` é projetada para retornar um *array* com as opções de configuração do plugin.

A opção de configuração `plugin_exemplo_backup` é representada por uma `select box` com o ID `plugin_exemplo_backup` e o *label* "Fazer Backup". Esta `select box` inclui uma descrição que especifica a sua funcionalidade: "Ativar ou desativar backups automáticos".

O resultado visual da página de configuração com a opção de 'fazer backup' relativo à Listagem 2 pode ser visto na Figura 2.4.

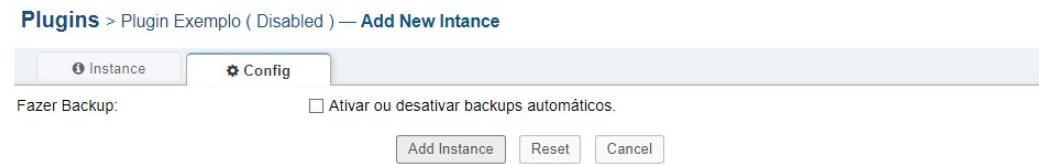


Figura 2.4: Página de configuração do Plugin

Os dados configurados são armazenados na Tabela `ost_config` da base de dados. Esta Tabela é utilizada para armazenar as configurações gerais do sistema e dos plugins, garantindo que cada configuração seja facilmente acessível e gerenciável.

Cada nova instalação de um plugin incrementa um número específico, criando *namespaces* como `plugin.6.instance.6` ou `plugin.7.instance.7`. Este sistema permite a instalação e gestão independente de múltiplos plugins, com cada um acedendo às suas configurações específicas. Os dados são armazenados de forma escalar ou em formato JavaScript Object Notation (JSON) no campo de valor da Tabela.

A Figura 2.5 apresenta um exemplo de entradas na Tabela `ost_config`, ilustrando como as configurações dos plugins são armazenadas.

id	namespace	key	value	updated
110	core	staff_backdrop_id	0	2024-06-28 17:28:18
111	core	previousActiveStatePlugin	0	2024-06-28 17:28:18
121	plugin.6.instance.6	last_selected_form_id	2	2024-06-15 17:36:54
122	plugin.6.instance.6	selected_form_id	{"2": "Ticket Details"}	2024-06-15 17:36:54
123	plugin.6.instance.6	save_on_deactivate	1	2024-06-15 17:41:10

Figura 2.5: Exemplo de entradas na Tabela `ost_config`

Tipos de Campos Disponíveis

Os campos disponíveis para configuração, definidos na classe `[INSTALL_ROOT]/include/class.forms.php`, incluem:

- `TextField` - Caixa de texto simples.
- `TextareaField` - Área de texto para conteúdos mais extensos.
- `ThreadEntryField` - Área de texto, normalmente utilizada para discussões em fóruns.
- `DatetimeField` - Select de datas baseado em JQuery.
- `PhoneField` - Caixa de texto otimizada para a inserção de números de telefone.
- `BooleanField` - Select box para opções binárias de verdadeiro/falso.
- `ChoiceField` - Campo dropdown para seleção entre várias opções.
- `SectionBreakField` - Quebra de Secção horizontal, que auxilia na organização visual das configurações.

Função `pre_save(&$config, &$errors)`

Conforme apresentado na Listagem 2, além da função `getOptions()`, existe a função `pre_save(&$config, &$errors)`. Esta função é ativada quando as configurações são submetidas pelo utilizador. A função `pre_save()` realiza a validação do lado do servidor e implementa a lógica necessária para o processamento das configurações.

Passo 3: Classe Principal do *Plugin*

Para que um **plugin** seja válido no **osTicket**, a classe do **plugin** deve obrigatoriamente incluir o método `bootstrap()`. Este método é responsável por inicializar o **plugin** e integrá-lo ao sistema. A função `bootstrap()` é chamada pelo **osTicket** durante a inicialização do **plugin** e sempre que um utilizador acede ao **osTicket**. Os demais métodos nesta classe são opcionais e destinam-se a ampliar a funcionalidade do **plugin**.

Tendo isso em mente, o próximo passo é criar um ficheiro denominado `PluginExemplo.php` para definir a classe principal do **plugin**, que deve conter toda a lógica necessária. Esta classe deve estender `Plugin` e incluir os métodos de inicialização e configuração do **plugin**. Na Listagem 3, apresentamos uma versão simples de um **plugin**.

Listagem 3: Exemplo de ficheiro `PluginExemplo.php`

```

1 <?php
2 require_once(INCLUDE_DIR . 'class.plugin.php');
3 require_once(INCLUDE_DIR . 'class.signal.php');
4 require_once(INCLUDE_DIR . 'class.osticket.php');
5 require_once(__DIR__ . '/config.php');
6
7 define('PLUGIN_DIR', __DIR__ . '/');
8 define('EXAMPLE_TABLE', TABLE_PREFIX . 'example_table');
9
10 class PluginExemplo extends Plugin {
11     public const PLUGIN_NAME = 'PluginExemplo';
12     var $config_class = 'PluginExemploConfig';
13
14     /**
15      * Inicializa o plugin.
16      *
17      * @return void
18      */
19     function bootstrap() {
20         $config = $this->getConfig();
21         error_log('A opção da configuração para a funcionalidade de backup
22         esta ' . $config->get('plugin_exemplo_backup'));
23     }
24 }
25 ?>

```

A classe `PluginExemplo` utiliza a classe `PluginExemploConfig`, de configuração, permitindo que o **plugin** utilize as opções de configuração definidas nesta classe. Durante a inicialização, através do método `bootstrap`, aceude-se à configuração do **plugin** e regista-se um *log* com a informação sobre o estado da seleção da configuração, conforme pode ser observado nas linhas 20 e 21 da Listagem 3.

Com as três classes prontas e colocadas na diretoria correta, basta instalar o plugin através da interface do osTicket e ativá-lo.

No osTicket, quando um plugin é instalado, ele é representado pelas Tabelas `ost_plugin` e `ost_plugin_instance`. A Tabela `ost_plugin` armazena os plugins instalados, incluindo informações como o nome, a versão e a data de instalação de cada plugin. A Tabela `ost_plugin_instance` armazena as instâncias dos plugins, permitindo a configuração e a gestão de múltiplas instâncias de um mesmo plugin. Cada instância possui um `plugin_id` que corresponde ao ID do plugin na Tabela `ost_plugin`, garantindo a associação correta entre plugins e suas instâncias.

Estas Tabelas permitem a consulta de informações sobre os plugins e suas respetivas instâncias. A Figura 2.6 apresenta um exemplo de entradas na Tabela `ost_plugin` na base de dados, enquanto a Figura 2.7 apresenta um exemplo de entradas na Tabela `ost_plugin_instance`.

<code>id</code>	<code>name</code>	<code>install_path</code>	<code>isphar</code>	<code>isactive</code>	<code>version</code>	<code>notes</code>	<code>installed</code>
6	Radar Details	plugins/PRJ_LEIM_SINCRO	0	0	1.0	NULL	2024-06-15 17:36:46
7	OSTicket API Extension	plugins/OSTicket-API-Extension	0	1	0.1	<p>Adds extra functionality to OSTicket API.</p>	2024-06-24 17:58:54

Figura 2.6: Exemplo de entradas na Tabela `ost_plugin` na base de dados

<code>id</code>	<code>plugin_id</code>	<code>flags</code>	<code>name</code>	<code>notes</code>	<code>created</code>	<code>updated</code>
6	6	1	plugintest	NULL	2024-06-15 17:36:54	2024-06-15 17:41:10
7	7	1	plugintest2	NULL	2024-06-24 17:59:17	2024-06-24 18:13:53

Figura 2.7: Exemplo de entradas na Tabela `ost_plugin_instance` na base de dados

Capítulo 3

Modelo Proposto

Este capítulo aborda a proposta de uma solução de implementação do **plugin** para o **osTicket**. Assim, nas Secções 3.1 e 3.2, elencam-se, respetivamente, os requisitos funcionais e não funcionais associados a este modelo. Já na Secção 3.3, apresentam-se os casos de utilização. Depois, na Secção 3.4, apresenta-se uma arquitetura de referência para o **plugin** que cumpra esses requisitos.

3.1 Requisitos Funcionais

Os requisitos funcionais de um sistema correspondem ao conjunto de especificações que descrevem as funcionalidades, capacidades e comportamentos esperados do sistema. Essas especificações devem descrever detalhadamente as ações que o sistema terá de ser capaz de realizar para atender às necessidades dos seus utilizadores e aos objetivos do projeto.

Com base neste estudo, foram definidos os seguintes requisitos funcionais mínimos para o **plugin**:

RF1 Inicialização, Configuração e Desativação do Plugin

RF1.1 O **plugin** deve permitir a inicialização automática ao ser carregado e ativado no sistema.

- **Explicação:** Facilita a ativação rápida e automática do **plugin**, garantindo que ele esteja pronto para uso imediatamente.

RF1.2 O **plugin** deve criar novas Tabelas para armazenar informações do projeto SINCRO.

- **Explicação:** As Tabelas com as informações do projeto SIN-CRO contêm dados importantes que serão utilizados no plugin.

RF1.3 O plugin deve criar novos campos para o reporte de avarias de radares num formulário específico.

- **Explicação:** Permite criar tickets com a informação estruturada relativamente às avarias das cabines.

RF1.4 Na configuração da instância do plugin, deve perguntar ao utilizador qual formulário pretende utilizar para inserir os novos campos, sendo esta seleção obrigatória.

- **Explicação:** Garante que os novos campos sejam inseridos no formulário correto conforme a necessidade da organização.

RF1.5 Na configuração da instância do plugin, deve perguntar ao utilizador se pretende ativar a funcionalidade de *backup* das entradas dos novos campos.

- **Explicação:** Oferece uma opção para *backup*, guardando as entradas dos campos do nosso plugin.

RF1.6 O plugin deve criar Tabelas para o *backup* das entradas dos novos campos do plugin.

- **Explicação:** Para podermos guardar as informações das entradas e dos campos na base de dados.

RF1.7 Na configuração da instância do plugin, deve verificar se o utilizador possui permissões para escrever no sistema de ficheiros; caso contrário, deve notificar o utilizador na interface.

- **Explicação:** Assegura que as operações de escrita no sistema de ficheiros sejam permitidas, prevenindo problemas no plugin e garantindo que ele consiga executar a sua função. É importante também para notificar o utilizador sobre os problemas.

RF1.8 Na configuração da instância do plugin, uma vez guardada, o formulário selecionado já não poderá ser alterado.

- **Explicação:** Evita inconsistências e garante a integridade dos dados após a configuração inicial.

RF1.9 O **plugin** deve realizar *patches* (*hot patching*) ao código core do **osTicket** para adicionar a funcionalidade de campos dinâmicos através de **AJAX**.

- **Explicação:** Permite a atualização do sistema em tempo real, sem necessidade de reiniciar o sistema, para adicionar as novas funcionalidades introduzidas pelo **plugin**.

RF1.10 Na primeira execução, o **plugin** deve verificar se existem Tabelas de *backup* e tentar recuperar os valores das entradas dos campos em questão.

- **Explicação:** Realiza a recuperação dos dados das entradas dos campos do **plugin** em todos os formulários que os contêm.

RF1.11 O **plugin** deve atualizar os logótipos padrão do **osTicket** para os da ANSR.

- **Explicação:** Alinha a interface do utilizador com a identidade visual da ANSR.

RF1.12 O **plugin** deve atualizar os *backdrops* padrão do **osTicket** para os da ANSR.

- **Explicação:** Alinha a interface do utilizador com a identidade visual da ANSR.

RF1.13 O **plugin** deve atualizar as informações da empresa para as da ANSR, incluindo:

RF1.1.1 Nome

RF1.1.2 URL do site

RF1.1.3 Número de telefone

RF1.1.4 Morada

- **Explicação:** Garante que todas as informações de contacto sejam atualizadas no sistema de acordo com a ANSR.

RF1.14 A desativação do **plugin** deve limpar todas as alterações feitas pelo **plugin** e pelas suas instâncias, com exceção das Tabelas de *backup*, que devem permanecer.

- **Explicação:** O plugin deve ser o menos invasivo possível para não prejudicar o sistema em nenhum aspecto.

RF2 Campos Adicionais

RF2.1 O plugin deve adicionar uma lista *dropdown* para selecionar a cabine com avaria.

- **Explicação:** Facilita a seleção da cabine correta para o registo de avarias.

RF2.2 O plugin deve adicionar campos do tipo `select box` para cada elemento da cabine:

RF2.1.1 Cabine

RF2.1.2 Router

RF2.1.3 Cinemómetro

RF2.1.4 UPS

RF2.1.5 Caixa

RF2.1.6 Outro

- **Explicação:** Permite ao utilizador selecionar o componente da cabine com a avaria.

RF2.3 O plugin deve adicionar uma caixa de texto para a descrição da avaria.

- **Explicação:** Permite uma descrição detalhada da avaria, ajudando na resolução de problemas.

RF3 AJAX

RF3.1 O plugin deve adicionar campos de filtragem para a seleção da cabine de acordo com:

RF3.1.1 Distrito

RF3.1.2 Estrada

RF3.1.3 Direção

- **Explicação:** Melhora a precisão e rapidez na seleção da cabine correta através de filtros específicos.

RF3.2 Através da filtragem, no *dropdown* das cabines, devem aparecer apenas as cabines que estão de acordo com os critérios de filtragem.

- **Explicação:** Reduz a complexidade da seleção exibindo apenas opções relevantes, melhorando a eficiência do utilizador.

3.2 Requisitos Não Funcionais

Os requisitos não funcionais de um sistema reportam às características ou qualidades que esse sistema deve possuir para atender às expectativas dos seus utilizadores. No caso do `plugin`, o nosso estudo levou à definição dos seguintes requisitos não funcionais mínimos:

RNF1 Desempenho

RNF1.1 O `plugin` deve ser responsivo, garantindo tempos de resposta rápidos ao adicionar novos campos e realizar backups.

- **Explicação:** Assegura que o `plugin` opere de maneira eficiente, sem atrasos significativos, melhorando a experiência do utilizador.

RNF1.2 O `plugin` deve ser capaz de lidar com um aumento no número de `tickets` sem causar degradação significativa no desempenho do sistema.

- **Explicação:** Garante que o `plugin` possa escalar adequadamente e suportar cargas crescentes sem impactar negativamente o sistema.

RNF1.3 O `plugin` deve utilizar eficientemente os recursos disponíveis, minimizando o impacto no desempenho geral do sistema `osTicket`.

- **Explicação:** Reduz o consumo de recursos do sistema, garantindo que o `osTicket` continue a funcionar de maneira otimizada.

RNF2 Usabilidade

RNF2.1 O `plugin` deve garantir que as suas operações de *backup* não interferem com as operações normais do sistema `osTicket`.

- **Explicação:** As operações de backup não devem comprometer de alguma forma o `osTicket`.

RNF2.2 O `plugin` deve funcionar consistentemente em diferentes navegadores e dispositivos computacionais, assegurando compatibilidade total com a interface do `osTicket`.

- **Explicação:** Garante uma experiência de utilizador uniforme e sem problemas, independentemente da plataforma ou dispositivo utilizado.

RNF3 Manutenção

RNF3.1 Deve existir documentação sobre o `plugin`, incluindo instruções de instalação e configuração.

- **Explicação:** Facilita a instalação, configuração e manutenção do `plugin` por parte dos utilizadores e administradores.

RNF4 Integração e Compatibilidade

RNF4.1 O `plugin` não deve comprometer a performance do sistema.

- **Explicação:** Garante que a adição do `plugin` não degrada o desempenho geral do `osTicket`.

RNF4.2 O `plugin` deve ser totalmente compatível com a versão do `osTicket` em que será instalado, assegurando que não causa conflitos ou instabilidade no sistema.

- **Explicação:** Previne problemas de compatibilidade, assegurando um bom funcionamento.

RNF4.3 O `plugin` deve ser projetado para facilitar a integração com outros `plugins` e personalizações existentes no sistema `osTicket`.

- **Explicação:** Permite uma fácil coexistência e integração com outras ferramentas e personalizações, aumentando a flexibilidade do sistema.

3.3 Casos de Utilização

Os casos de utilização (*use cases*) descrevem as interações entre os utilizadores e o sistema para alcançar um objetivo específico. Para o projeto

do plugin *osTicket-SINCRO*, foram identificados os seguintes casos de utilização principais, ilustrados na Figura 3.1.

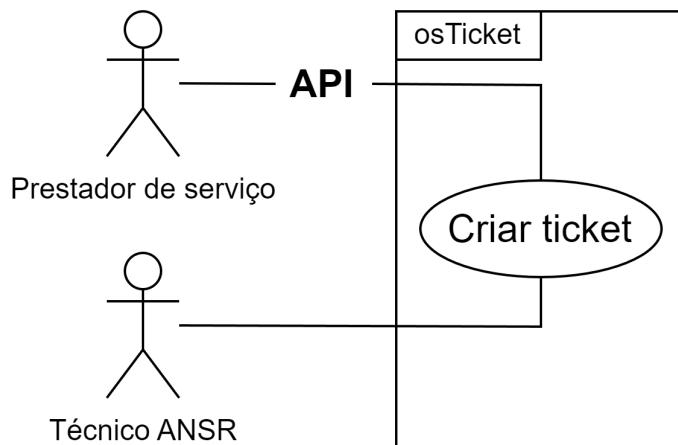


Figura 3.1: Casos de Utilização - Criação de Tickets

Na Figura 3.1, existem dois atores principais, um prestador (**Prestador de serviço**) e um técnico (**Técnico ANSR**). Ambos criam um ticket, mas de maneiras diferentes. O prestador de serviço utiliza a *API*, enquanto o técnico da ANSR cria o *ticket* manualmente. Em ambas as situações, é utilizado o nosso formulário estruturado.

Estes casos de utilização são fundamentais para garantir que as necessidades dos diferentes tipos de utilizadores sejam atendidas de maneira eficiente, eficaz e de forma consistente. A integração com a *API* permite automação e rapidez na criação de *tickets* por parte dos prestadores de serviço, enquanto a criação manual do *ticket* garante que os técnicos possam inserir informações detalhadas conforme necessário, utilizando a lista de seleção de cabines e as caixas de seleção para a filtragem dessas cabines.

A Figura 3.2 apresenta os restantes casos de utilização identificados.

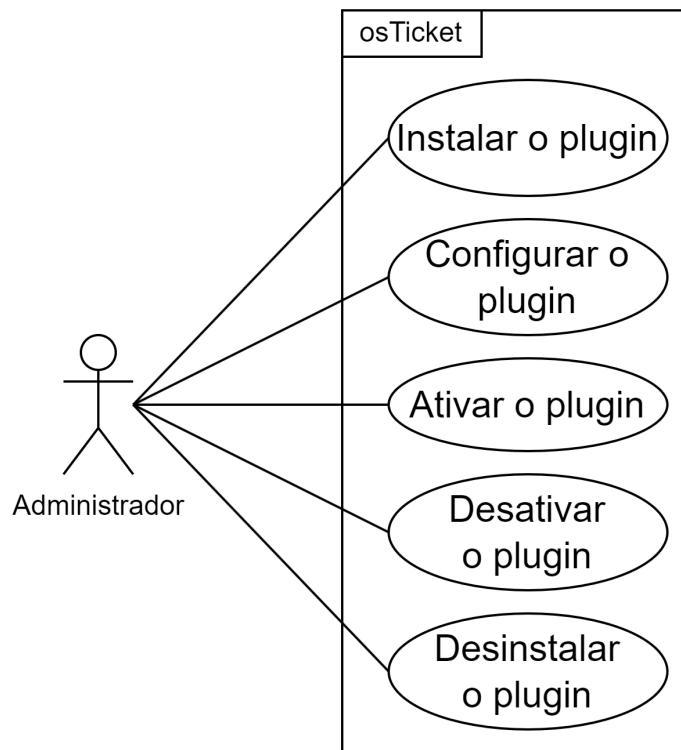


Figura 3.2: Casos de Utilização - Gestão do Plugin

Na Figura 3.2, o administrador (**Administrador ANSR**) é o ator principal, que desempenha um papel crucial na instalação e configuração do nosso plugin no sistema da ANSR. O administrador será responsável por configurar o plugin, decidindo a quais formulários serão adicionados os novos campos estruturados.

Durante a configuração, o administrador poderá selecionar a opção que define se o plugin realizará o *backup* dos campos e das entradas dos mesmos na base de dados, garantindo que, ao ser desativado, a informação permaneça guardada para possível recuperação futura.

O administrador também terá a capacidade de ativar o plugin, que inicialmente estará desativado, bem como desativar e desinstalar o plugin conforme necessário.

Estas funcionalidades são essenciais para assegurar que o plugin possa ser gerido de forma eficiente, atendendo às necessidades de configuração e manutenção do sistema.

3.4 Arquitetura Proposta

O diagrama de classes do nosso projeto de plugin foi elaborado utilizando o *MagicDraw*, conforme apresentado na Figura 3.3.

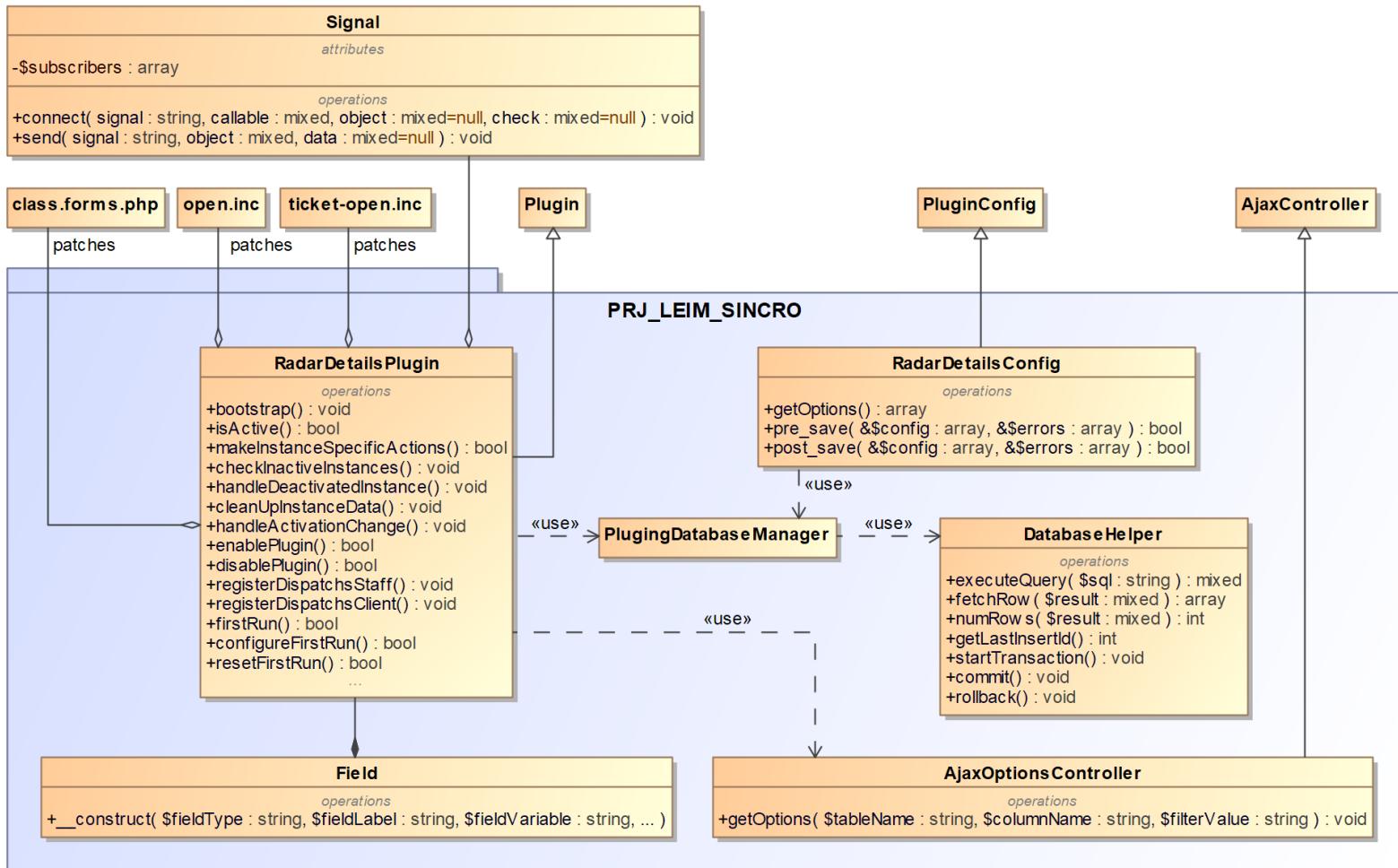


Figura 3.3: Diagrama UML do projeto

O diagrama na Figura 3.3 inclui as classes do nosso plugin, localizadas dentro do *package* PRJ_LEIM_SINCRO, enquanto as classes existentes no osTicket, que utilizamos e alteramos, estão representadas fora do *package*.

Analizando o nosso *package* PRJ_LEIM_SINCRO, podemos observar que é constituído por diversas classes. A classe RadarDetailsPlugin, que contém toda a lógica do plugin, estende a classe Plugin. Esta classe utiliza a classe Field, estabelecendo uma relação de agregação. Portanto, quando criamos novos campos, utilizamos objetos do tipo Field. Além disso, a classe RadarDetailsPlugin conecta-se aos sinais necessários para registrar as rotas essenciais ao funcionamento do AJAX. Para isso, utiliza a classe Signal e a classe AjaxOptionsController, que estende AjaxController e contém a lógica para as rotas AJAX. Todos os métodos que precisam realizar operações na base de dados utilizam a classe PluginDataBaseManager, que estende DataBaseHelper e contém todas as operações relacionadas com a base de dados do osTicket.

Além disso, a classe RadarDetailsPlugin realiza *patches* em três classes: open.inc.php, class.forms.php e ticket-open.inc.php. Estes *patches* permitem a adição dos campos de filtragem de cabines para auxiliar na seleção da cabine adequada, uma funcionalidade que ainda não estava implementada no osTicket e que foi necessário desenvolver através de *patches*. Esses campos de filtragem utilizam o AJAX mencionado anteriormente.

A classe RadarDetailsConfig, que estende PluginConfig, é responsável pela configuração do nosso plugin. Especificamente, gera as opções que aparecem na página de configuração do osTicket durante a configuração do plugin. Esta classe também faz uso da PluginDataBaseManager para realizar operações na base de dados necessárias à configuração, onde o administrador escolhe a quais formulários os novos campos devem ser adicionados, sendo necessário ter acesso a todos os formulários disponíveis no osTicket.

3.4.1 O que é um *Patch*

Um *patch* é uma modificação no código-fonte de um software, destinada a corrigir *bugs*, aprimorar funcionalidades existentes ou adicionar novos recursos, sem alterar significativamente a estrutura original do programa.

Capítulo 4

Implementação do Modelo

Neste capítulo, descreve-se o processo de implementação do modelo proposto, desde o início, onde utilizámos o `plugin` base apresentado na Secção 2.3, até ao desenvolvimento atual do `plugin Radares Details`. Partindo do `plugin` base, adicionámos novas funcionalidades e conceitos para atender às necessidades identificadas. Este capítulo aborda as etapas seguidas, as decisões tomadas e os desafios enfrentados ao longo do desenvolvimento. Todo o código do `Plugin` está presente no repositório *GitHub* [1].

Inicialmente, é feita uma descrição do `plugin` na Secção 4.1, seguida do planeamento da implementação na Secção 4.2, onde são resumidas as suas funcionalidades principais. Por fim, apresenta-se o desenvolvimento do `plugin Radares Details` na Secção 4.3, abordando as etapas de codificação do `plugin`, a implementação da página de configuração, o fluxo de ativação e desativação, a criação de campos adicionais para formulários, a funcionalidade de backup, e a implementação de campos de filtragem utilizando *AJAX*.

4.1 Desenvolvimento do Plugin

Para começar a implementar o modelo, partimos de um `plugin` simples, como demonstrado na Secção 2.3. A escolha de um `plugin` simples permitiu-nos compreender a estrutura básica e as funcionalidades essenciais necessárias para a criação de um `plugin` para o `osTicket`.

4.1.1 Descrição do Plugin Inicial

O plugin inicial escolhido serviu como base para entender a estrutura do `osTicket` e iniciar a customização. Este plugin básico permitiu-nos compreender a utilização da página de configuração do plugin e a integração desta configuração na classe que contém a lógica do nosso plugin. Além disso, aprendemos como o método `bootstrap` é chamado e como utilizá-lo para adicionar novas funcionalidades.

4.2 Planeamento da Implementação

Para assegurar uma implementação eficaz, realizámos um planeamento que incluiu, conforme referido nas Secções anteriores:

- Definição das funcionalidades do *Radares Details* através da análise de requisitos funcionais, presente na Secção 3.1, e pelos casos de utilização, presentes na Secção 3.3.
- Análise dos requisitos não funcionais, presente na Secção 3.2.
- Arquitetura do plugin, presente na Secção 3.4.

4.2.1 Resumo das Funcionalidades do Plugin

Em resumo, as funcionalidades essenciais que o plugin deve suportar incluem a criação de `tickets` estruturados através da adição de novos campos num determinado formulário, sendo este escolhido na configuração do plugin. Além disso, o plugin deve permitir o backup das entradas dos novos campos introduzidos, sendo esta uma opção presente na página de configuração do plugin. O principal objetivo desta funcionalidade é assegurar que, quando o plugin for desativado e desinstalado, se for novamente instalado e ativado, seja possível recuperar toda a informação desses mesmos campos.

4.3 Desenvolvimento do Plugin *Radares Details*

Esta Secção apresenta as etapas de codificação do plugin na Secção 4.3.1, a implementação da página de configuração na Secção 4.3.2, o fluxo de

ativação e desativação na Secção 4.3.3, a criação de campos adicionais para formulários na Secção 4.3.4, a funcionalidade de backup na Secção 4.3.5 e a implementação de campos de filtragem utilizando **AJAX** na Secção 4.3.6.

4.3.1 Codificação do Plugin

Durante a fase de codificação, estas foram as etapas de desenvolvimento do **plugin**:

- Implementação de uma página de configuração que contém as configurações necessárias para o funcionamento do **plugin**.
- Implementação do fluxo de ativação e desativação do **plugin** e das suas instâncias.
- Implementação da funcionalidade de adição de novos campos a um determinado formulário escolhido na configuração.
- Implementação do **backup** dos campos introduzidos pelo **plugin**.
- Implementação de campos de filtragem através de **AJAX** para ajudar na seleção da cabine correta nos **tickets** que contêm o formulário estruturado introduzido pelo **plugin**.

4.3.2 Implementação da página de configuração

Para que o nosso **plugin** funcione corretamente, é necessário criar pelo menos uma instância, onde serão realizadas as configurações do **plugin**. As configurações essenciais para este **plugin** incluem a escolha do formulário para a inserção dos novos campos e a seleção da opção de ativação de **backup** dos nossos campos e das suas respetivas entradas. A página de configuração pode ser observada na Figura 4.1.

Plugins > Radar Details (Disabled) — Update Instance

Select Form: **Ticket Details**

Save Data on Deactivation: Enable this to save form data to a file when the plugin is deactivated.

Save Changes **Reset** **Cancel**

Figura 4.1: Página de configuração do plugin

Neste caso, o formulário escolhido foi o formulário `ticket Details`, que aparece por padrão em todos os *tickets* de todos os *help topics*, sendo portanto adequado para averiguar o funcionamento do plugin. A `select box` para a escolha do formulário apresenta todos os formulários disponíveis no momento no `osTicket`. A `select box` com os formulários disponíveis pode ser observada na Figura 4.2.

Plugins > Radar Details (Disabled) — Add New Instance

Select Form:

- None
- Contact Information
- Ticket Details**
- Company Information
- Organization Information
- Task Details
- Ticket Status Properties
- Test Details

data to a file when the plugin is deactivated.

Reset **Cancel**

Figura 4.2: Página de configuração do plugin

Estas funcionalidades acima descritas, foram implementadas na classe `RadarDetailsConfig`, que estende de `PluginConfig`, e seguimos portanto o a estrutura apresentada na Secção 2.3.2, em concreto na Listagem 2.

A opção de selecionar um formulário é obrigatória, pois é fundamental termos um formulário selecionado para o funcionamento do nosso plugin. Sem um formulário selecionado, e caso o utilizador tente gravar a configuração, será apresentada uma mensagem de erro a informar sobre o problema. A Figura 4.3 ilustra um exemplo de quando não se seleciona um formulário.

You must select a form.

Plugins > Radar Details (Disabled) — **Update Instance**

Instance Config

Select Form:

Save Data on Deactivation: Enable this to save form data to a file when the plugin is deactivated.

Figura 4.3: Erro ao não selecionar um formulário

A página de configuração tem ainda outra função. Quando estamos a configurar, o utilizador será sempre um administrador do sistema, e portanto, é importante que tenha os acessos no sistema de ficheiros disponíveis. Como será explicado mais à frente, iremos realizar alterações em alguns ficheiros e, portanto, precisamos de ter as permissões necessárias, sendo crucial para o funcionamento do nosso plugin. Em caso de não existirem permissões, será exibido um alerta visual que impedirá o avanço da configuração. A Figura 4.4 ilustra a página com a mensagem de erro de privilégios.

Write permissions are not available in the include directory. Please ensure the web server has write access to the directory.

Plugins > Radar Details (Disabled) — **Update Instance**

Instance Config

Select Form:

Save Data on Deactivation: Enable this to save form data to a file when the plugin is deactivated.

Figura 4.4: Erro de privilégios insuficientes

Esta configuração foi possível através da criação de um ficheiro *dummy* no qual realizamos operações de criação, escrita, leitura e exclusão. Se nenhuma dessas operações gerar erro, significa que temos os privilégios suficientes para continuar com a configuração. Caso contrário, será apresentada a mensagem de erro ilustrada na Figura 4.4.

Em caso de a configuração ter sido bem-sucedida, o valor de cada configuração será introduzido na base de dados na Tabela *ost_config*, que irá armazenar o ID do formulário selecionado e o valor da *select box* do *backup*. A Figura 4.5 mostra as duas entradas na Tabela *ost_config* da nossa

configuração.

122 plugin.6.instance.6 selected_form_id	{"2":"Ticket Details"}	2024-06-15 17:36:54
123 plugin.6.instance.6 save_on_deactivate	1	2024-06-15 17:41:10

Figura 4.5: Entradas na Tabela ost_config da configuração do plugin

Quando a lógica do plugin necessitar de saber qual o ID do formulário foi selecionado e se a função de *backup* está ativa ou não, será a esta Tabela que irá obter esses valores.

4.3.3 Implementação do Fluxo de Ativação e Desativação

Para termos um plugin o mais eficaz possível, criámos um fluxo de ativação e desativação que permite gerir os estados do plugin, bem como das suas instâncias. Desta forma, podemos controlar as ações a serem realizadas em cada estado, assegurando uma gestão eficiente e adaptada às necessidades específicas de cada situação.

Métodos principais para a gestão do estado do plugin

Na Listagem 4 estão apresentados os métodos principais utilizados para gerir o estado do plugin.

Listagem 4: Métodos para a gestão de estados do plugin

```

1 function isActive()
2 {
3     $currentlyActive = parent::isActive();
4     $previouslyActive = (int)$this->getCurrentActiveStatePlugin();
5     $currentlyActive = (int)$currentlyActive;
6
7     if ($previouslyActive !== $currentlyActive) {
8         $this->handleActivationChange($currentlyActive);
9         $this->setCurrentActiveStatePlugin($currentlyActive);
10    }
11
12    return $currentlyActive;
13 }
14
15 private function handleActivationChange($currentlyActive)
16 {
17     if ($currentlyActive) {
18         $this->enablePlugin();
19     } else {
20         $this->disablePlugin();
21     }

```

22 }

Os métodos listados na Listagem 4 realizam as seguintes funções:

isActive Verifica se o plugin está ativo. Este método, originalmente da classe `Plugin`, foi sobreescrito para adicionar mais funcionalidades. Originalmente, o método apenas devolvia o estado do plugin. O método é chamado várias vezes, por exemplo, sempre que atualizamos uma página ou mudamos de página. Para evitar que o método execute a lógica repetidamente a cada chamada, implementámos um sistema para que apenas execute alguma ação se o estado anterior for diferente do estado atual, conforme pode ser analisado na Listagem 5.

Listagem 5: Método `isActive()`

```

1 function isActive()
2 {
3     $currentlyActive = parent::isActive();
4     $previouslyActive = (int)$this->getCurrentActiveStatePlugin();
5     $currentlyActive = (int)$currentlyActive;
6
7     if ($previouslyActive !== $currentlyActive) {
8         $this->handleActivationChange($currentlyActive);
9         $this->setCurrentActiveStatePlugin($currentlyActive);
10    }
11
12    return $currentlyActive;
13 }
```

Desta forma, conseguimos detetar se realmente o estado do plugin mudou e executar as ações apropriadas para cada estado, gerindo a mudança e atualizando o estado atual, chamando `handleActivationChange`, sendo o estado passado como argumento ao método. O estado atual do plugin é ainda atualizado através da função `setCurrentActiveStatePlugin`.

handleActivationChange Gere a mudança no estado de ativação, chamando `enablePlugin` se o plugin estiver a ser ativado, ou `disablePlugin` se estiver a ser desativado, como demonstrado na Listagem 6.

Listagem 6: Método `handleActivationChange(currentlyActive)`

```

1 private function handleActivationChange($currentlyActive)
2 {
3     if ($currentlyActive) {
4         $this->enablePlugin();
5     } else {
6         $this->disablePlugin();
7     }
}
```

```
8 }
```

enablePlugin Este método executa as ações necessárias para ativar o plugin, como listado na Listagem 7.

Listagem 7: Método enablePlugin()

```
1 function enablePlugin()
2 {
3     if (parent::isActive()) {
4         PluginDataBaseManager::executeSqlFile(__DIR__ . '/scripts/01-
CreateSchema.sql');
5         PluginDataBaseManager::executeSqlFile(__DIR__ . '/scripts/02-
Populate.sql');
6         PluginDataBaseManager::criarEPopularListaCabines();
7         $this->injectCodeIntoFormClass();
8         $this->injectCodeIntoFiles();
9         $this->uploadCustomFiles();
10        $this->updateCompanyInformation();
11    }
12    return true;
13 }
```

O método `enablePlugin` executa várias ações essenciais para ativar o plugin. Primeiramente, o mesmo cria as Tabelas na base de dados relacionadas com o projeto SINCRO e popula a lista de cabines. A seguir, realiza *patches* nas classes (`class.forms.php`, `ticket-open.php` e `open.php`), e atualiza os logótipos e o *backdrop* do site para os da ANSR.

As Tabelas são criadas pela classe auxiliar `PluginDataBaseManager`, que contém todos os métodos necessários para aceder à base de dados do `osTicket`. Um dos métodos dessa classe executa ficheiros Structured Query Language (SQL), passando como argumento o caminho do ficheiro. Após a criação das Tabelas, é criada e populada a lista de cabines, chamando o método `criarEPopularListaCabines` da mesma classe auxiliar.

O método `criarEPopularListaCabines` cria uma nova lista na Tabela `ost_list`, que contém todas as listas do `osTicket`. Esta lista pode posteriormente ser associada a um formulário específico. Em seguida, o método acede à Tabela `sincro_cabinet`, que contém todas as informações das cabines existentes, e popula a lista recém-criada com esses dados, inserindo-os na Tabela `ost_list_items` do `osTicket`. A Figura 4.6 mostra as entradas na Tabela `ost_list`, enquanto a Figura 4.7 ilustra as entradas na Tabela `ost_list_items`, contendo as informações das cabines do projeto SINCRO.

74	Lista de Cabines	NULL	Alpha	0	NULL	2024-06-27 19:17:36	2024-06-27 19:17:36
----	------------------	------	-------	---	------	---------------------	---------------------

Figura 4.6: Entrada na Tabela `ost_list` da lista de cabines

<code>id</code>	<code>list_id</code>	<code>status</code>	<code>value</code>	<code>extra</code>	<code>sort</code>	<code>properties</code>
367	74	1	1;100;1;1;M-Box;Micotec;SN-B-SIN-1;A1;Porto;km 292...	NULL	0	
368	74	1	2;100;2;2;M-Box;Micotec;SN-B-SIN-2;EN252;Setubal;k...	NULL	0	
369	74	1	3;100;3;3;M-Box;Micotec;SN-B-SIN-3;IC1;Setubal;km ...	NULL	0	
370	74	1	4;100;4;4;YT-Box;Yunex Traffic;SN-C-SIN-1;A44;Vila...	NULL	0	
371	74	1	5;100;5;5;YT-Box;Yunex Traffic;SN-C-SIN-2;EN103;Ba...	NULL	0	
372	74	1	6;100;6;6;YT-Box;Yunex Traffic;SN-C-SIN-3;A2;Faro;...	NULL	0	

Figura 4.7: Entradas na Tabela `ost_list_items` com informações das cabines do projeto SINCRO

O método `enablePlugin()` realiza ainda, *patches* nas classes do *core* do `osTicket class.forms.php`, `ticket-open.php` e `open.php`, cujos detalhes serão descritos na Secção 4.3.6. Finalmente, o método atualiza os logótipos e *backdrop* do site para os da ANSR utilizando o método `uploadCustomFiles()`, cujo resultado pode ser visto na Figura 4.8 e 4.9.

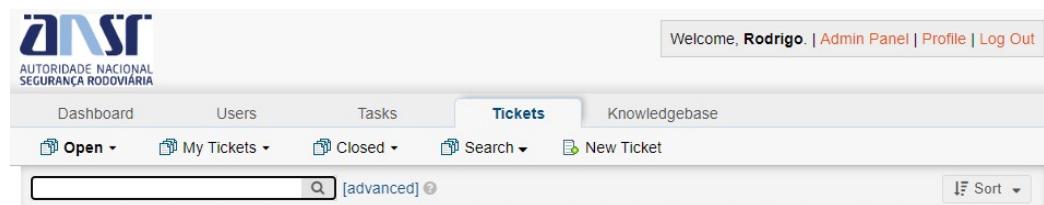
Figura 4.8: Página do `osTicket`



Figura 4.9: Página de *login* para agentes osTicket

Além disso, o método `enablePlugin` realiza a alteração dos dados da empresa para os da ANSR, resultando na visualização mostrada na Figura 4.10.

A screenshot of a web-based application showing the 'Company Profile' section. The top navigation bar includes tabs for 'Basic Information', 'Site Pages', 'Logos', and 'Login Backdrop'. The 'Basic Information' tab is selected. Below the tabs, there is a section titled 'Company Information' with a note: 'Details available in email templates'. It contains four form fields: 'Company Name' (ANSR), 'Website' (http://www.ansr.pt/), 'Phone Number' (21 423 6800), and 'Address' (Av. do Casal de Cabanas 1, 2734-507). At the bottom right are 'Save Changes' and 'Reset Changes' buttons.

Figura 4.10: Página com as informações da empresa

disablePlugin Este método executa as ações necessárias para desativar o plugin. Reverte as alterações feitas durante a ativação, removendo as Tabelas criadas, limpando os dados das instâncias (neste caso, removendo as alterações que as mesmas fizeram, como será descrito na Secção 4.3.3), removendo o código que fez *patch* às classes necessárias, removendo os arquivos personalizados carregados (como o logótipo e o *backdrop*), limpando as informações da empresa e redefinindo o estado de primeira execução, conforme listado na Listagem 8.

Listagem 8: Método `disablePlugin()`

```

1 function disablePlugin()
2 {
3     $instances = $this->getInstances();
4     foreach ($instances as $instance) {
5         $this->cleanUpInstanceData($instance);
6     }
7     PluginDataBaseManager::executeSqlFile(__DIR__ . '/scripts/03-DropSchema.
8     sql');
9     PluginDataBaseManager::removerListaCabines();
10    $this->removeInjectedCodeFromFormClass();
11    $this->removeInjectedCodeFromFiles();
12    $this->removeUploadedFiles();
13    $this->clearCompanyInformation();
14    $this->resetFirstRun();
15    PluginDataBaseManager::deleteFieldListaCabinesProperties();
16    return true;
17 }
```

Fluxo de Ativação e Desativação das Instâncias

A gestão de estados das instâncias do plugin envolve a chamada ao método `bootstrap`, que é executado pelo sistema ao longo da execução do plugin. Esse método pode ser visto na Listagem 9.

Listagem 9: Método `bootstrap()`

```

1 function bootstrap()
2 {
3     ...
4     $this->makeInstanceSpecificActions();
5     $this->checkInactiveInstances();
6     ...
7 }
```

O método `bootstrap` chama o método `makeInstanceSpecificActions`, que, caso a instância esteja ativa, executa a lógica necessária para ativar a instância.

O método `makeInstanceSpecificActions` é responsável por realizar ações específicas de instância quando o `plugin` está ativo. Ele restaura as informações do `plugin` a partir de backups, configura a primeira execução e adiciona os campos necessários ao formulário selecionado.

Além disso, o método `bootstrap`, para verificar se há alguma instância inativa, chama o método `checkInactiveInstances`. Se existir, é chamado o método `cleanUpInstanceData` para realizar a limpeza dos dados.

O método `cleanUpInstanceData` limpa os dados das instâncias desativadas, removendo os campos adicionados e preservando as informações se a opção de `backup` estiver ativada.

Esta abordagem garante que o `plugin` funcione de forma robusta, com uma gestão eficiente de ativação, desativação e manipulação de instâncias, atendendo às necessidades específicas do projeto SINCRO e garantindo a integridade dos dados.

4.3.4 Criação dos Campos para Adicionar a um Formulário

A funcionalidade de criação de campos para adicionar a um formulário é a principal e mais importante do nosso `plugin`. Através destes campos, será possível criar um `ticket` estruturado, garantindo a organização e a eficácia na gestão das informações.

Para a criação dos campos, é necessário seguir uma série de passos, desde a definição dos campos até à sua integração no formulário selecionado. Abaixo, detalhamos cada um destes passos:

Definição dos Campos

Os campos necessários para o nosso formulário incluem:

- Campo de seleção para a lista de cabines.
- Campos booleanos para os componentes da cabine (e.g., Cabine, Router, Cinemómetro, UPS, Caixa, Outro).
- Campo de texto para a descrição da avaria.

Criação do Objeto de Campos

A criação do objeto que irá armazenar os campos é feita através da função `createCamposObj()`. Este objeto será posteriormente utilizado para adicionar ou remover os campos necessários no formulário selecionado. Um objeto do tipo `Campo` deve ser composto pelos seguintes atributos para satisfazer a Tabela `form_field` do `osTicket`:

- `fieldType`: O tipo do campo.
- `fieldLabel`: O *label* do campo.
- `fieldVariable`: O nome da variável do campo.
- `fieldDefaultValue`: O valor padrão do campo.
- `fieldFlags`: As *flags* do campo.
- `fieldConfiguration`: A configuração do campo.
- `fieldHint`: A *hint* do campo.

Adição dos Campos ao Formulário

A adição dos campos ao formulário selecionado é realizada através da função `adicionarCamposNecessarios()`, que faz parte da classe auxiliar `PluginDataBaseManager`. Esta função é responsável por inserir os campos definidos no objeto de campos na Tabela `form_field` do `osTicket`. Para cada campo a ser inserido, a função `adicionarCampoPersonalizado()` é chamada, verificando se o campo já existe no formulário e, se não existir, adicionando-o à base de dados.

Remoção dos Campos do Formulário

A remoção dos campos do formulário é realizada quando o `plugin` é desativado. Esta ação garante que o formulário volte ao seu estado original, sem os campos adicionais que foram inseridos pelo `plugin`, e que todas as entradas dos nossos campos sejam removidas. A função `removerCamposNecessarios()` garante que todos os campos personalizados adicionados pelo `plugin` sejam removidos corretamente das Tabelas `form_field` e `form_entry_values`.

Resumo das Operações nas Tabelas do osTicket

- Tabela `form_field`: Esta Tabela armazena as definições dos campos personalizados adicionados pelo `plugin`, incluindo o tipo, `label`, nome da variável, valor padrão, `flags`, configuração e `hint`.
- Tabela `form_entry_values`: Esta Tabela armazena os valores das entradas dos formulários, associando cada campo personalizado aos `tickets` correspondentes.

Para ilustrar a funcionalidade, o método `createCamposObj` cria um *array* de campos com os detalhes necessários e retorna um objeto que será usado para adicionar ou remover esses campos nos formulários. A função `adicionarCamposNecessarios` percorre cada campo e chama a função `adicionarCampoPersonalizado` para inseri-los na Tabela `form_field`. Similarmente, a função `removerCamposNecessarios` garante que todos os campos e suas entradas sejam removidos corretamente das Tabelas `form_field` e `form_entry_values`.

Esta abordagem modular permite uma gestão eficiente e segura dos campos personalizados adicionados pelo `plugin`, garantindo a integridade e a consistência dos dados no sistema `osTicket`.

Demonstração de um ticket com os Novos Campos

Para demonstrar o funcionamento, a Figura 4.14 ilustra o aspetto de um `ticket` com os novos campos adicionados.

The screenshot shows a 'ticket' form with the following sections:

- Test Details:** A text area containing "This is a test!"
- Lista de Cabines:** A dropdown menu showing "Setubal" selected, with other options like "EN252" and "AF/AP". Below it is a list of items: "2;100;2;2;M-Box;Micotec;SN-B-SIN-2;EN252;Setubal;k1". A placeholder "Selecione uma Cabine" is present.
- Selecionar dispositivos com avarias:** A section with checkboxes for different device types:
 - Cabine: Cabine (hint)
 - Router: Router (hint)
 - Cinemómetro: Cinemómetro (hint)
 - UPS: UPS (hint)
 - Caixa: Caixa (hint)
 - Outro: Outro (hint)
- Descrição:** A text area with placeholder "Descreva a avaria".

Figura 4.11: ticket com os novos campos

4.3.5 Implementação da Funcionalidade de Backup

Para garantir a preservação dos dados dos campos introduzidos pelo plugin, implementámos uma funcionalidade de backup. Esta funcionalidade assegura que, mesmo que o plugin seja desativado ou desinstalado, os dados dos campos adicionados não se percam e possam ser restaurados posteriormente. Abaixo, detalhamos como esta funcionalidade foi desenvolvida utilizando a classe `PluginDataBaseManager`.

Criação do Backup

Para criar o backup dos campos personalizados e dos valores das entradas dos formulários, o plugin segue um procedimento automatizado que envolve:

1. **Criação das Tabelas de Backup:** Verificação e criação das Tabelas de backup necessárias.
2. **Limpeza dos Dados Antigos:** Remoção de dados antigos nas Tabelas de backup para evitar duplicações.

3. Adição dos Novos Dados:

Inserção dos dados atuais dos campos personalizados e entradas dos formulários nas Tabelas de backup.

Este procedimento é invocado pelo método `keepPluginInfo`, que é chamado sempre que o plugin é desativado, desde que a opção de backup esteja ativada. Portanto, se a opção `save_on_deactivate` estiver ativada, o método irá realizar o backup dos nossos campos e entradas, assegurando a integridade e a continuidade dos dados.

Para armazenar os dados de backup, foram criadas três Tabelas na base de dados:

- `ost_backup_form_field`: Armazena a estrutura dos campos personalizados, incluindo o tipo, rótulo, nome da variável, valor padrão, *flags*, configuração e dica do campo.
- `ost_backup_form_entry_values`: Armazena os valores das entradas dos formulários para cada campo personalizado, permitindo a restauração dos dados inseridos pelos utilizadores.
- `ost_backup_form_entry`: Armazena as entradas dos formulários, garantindo que todas as entradas dos `tickets` sejam preservadas.

Estas Tabelas são criadas apenas se ainda não existirem na base de dados, garantindo que o processo de backup não afeta outras operações.

Adição de Campos e Valores ao Backup

Para mover as informações do plugin, das Tabelas principais do osTicket, para as Tabelas de backup, apenas fazemos um `SELECT` dos elementos adicionados pelo plugin às Tabelas principais, e um `INSERT` nas Tabelas de backup.

Estas operações envolvem a leitura das Tabelas originais `ost_form_field`, `ost_form_entry_values` e `ost_form_entry`, e a inserção dos dados nas Tabelas de backup correspondentes.

Restauração dos Dados do Backup

A restauração dos dados de backup é crucial para garantir que, caso o plugin seja reinstalado, todos os campos e dados anteriores sejam recuperados corretamente.

O processo de restauração é semelhante ao processo de backup, no sentido em que apenas fazemos um **SELECT** de todos os elementos das Tabelas de backup, e um **INSERT** nas Tabelas correspondentes do **osTicket**.

Uma particularidade que o processo de restauro tem sobre o processo de backup, é o facto do identificador da lista de cabines ter de ser atualizado antes de ser reinserido na Tabela `ost_form_field`.

Quando o plugin é ativado, são criadas e populadas as Tabelas que contêm todas as cabines disponíveis. Estas cabines são depois listadas na Tabela `ost_list_items` (como mostrado na Figura 4.12), para que possam aparecer na interface gráfica, para o utilizador as selecionar.

		Editar		Copiar		Apagar	id	list_id	status	value	extra	sort	properties
<input type="checkbox"/>		Editar		Copiar		Apagar	205	24	1	1;100;1;1:M-Box;Micotec;SN-B-SIN-1;A1;Porto;km 292...	NULL	0	
<input type="checkbox"/>		Editar		Copiar		Apagar	206	24	1	2;100;2;2:M-Box;Micotec;SN-B-SIN-2;EN252;Setubal;k...	NULL	0	
<input type="checkbox"/>		Editar		Copiar		Apagar	207	24	1	3;100;3;3:M-Box;Micotec;SN-B-SIN-3;IC1;Setubal;km ...	NULL	0	
<input type="checkbox"/>		Editar		Copiar		Apagar	208	24	1	4;100;4;4:YT-Box;Yunex Traffic;SN-C-SIN-1;A44;Vila...	NULL	0	
<input type="checkbox"/>		Editar		Copiar		Apagar	209	24	1	5;100;5;5:YT-Box;Yunex Traffic;SN-C-SIN-2;EN103;Ba...	NULL	0	
<input type="checkbox"/>		Editar		Copiar		Apagar	210	24	1	6;100;6;6:YT-Box;Yunex Traffic;SN-C-SIN-3;A2;Faro;...	NULL	0	

Figura 4.12: Entradas na Tabela ost_list_items

Cada lista tem um identificador, que incrementa sempre que uma lista é inserida na Tabela, logo, quando o `plugin` é reativado, o `id` da lista incrementa, mas o da `lista_cabines`, na Tabela de `backup`, não estando atualizado, tem de ser corrigido, como mostra a Figura 4.13.

	Id	form_id	flags	type	label	name	configuration	sort	hint	created	updated
Editar	Copiar	Apagar	1505	2	13057	list-23	Lista de Cabines	lista_cabines		0	Seleciona uma Cabine
Editar	Copiar	Apagar	1505	2	13057	cabin	{“text”: “Cabin”}	0		2024-06-26 18:15:02	2024-06-26 18:15:52
Editar	Copiar	Apagar	1505	2	13057	router	{“text”: “Router”}	0		2024-06-26 18:15:02	2024-06-26 18:15:52
Editar	Copiar	Apagar	1505	2	13057	circurometer	{“text”: “Circurometer”}	0		2024-06-26 18:15:02	2024-06-26 18:15:52
Editar	Copiar	Apagar	1505	2	13057	ups	{“text”: “UPS”}	0		2024-06-26 18:15:02	2024-06-26 18:15:52
Editar	Copiar	Apagar	1505	2	13057	caixa	{“text”: “Caixa”}	0		2024-06-26 18:15:02	2024-06-26 18:15:52
Editar	Copiar	Apagar	1505	2	13057	outro	{“text”: “Outro”}	0		2024-06-26 18:15:02	2024-06-26 18:15:52
Editar	Copiar	Apagar	1505	2	13057	descricao	{“text”: “Descrição”}	0	Desebre a avana	2024-06-26 18:15:02	2024-06-26 18:15:52
 											
	Id	form_id	flags	type	label	name	configuration	sort	hint	created	updated
Editar	Copiar	Apagar	1505	2	13057	list-24	Lista de Cabines	lista_cabines		0	Seleciona uma Cabine
Editar	Copiar	Apagar	1505	2	13057	cabin	{“text”: “Cabin”}	0		2024-06-26 18:15:02	2024-06-26 18:15:52
Editar	Copiar	Apagar	1505	2	13057	outro	{“text”: “Outro”}	0		2024-06-26 18:15:02	2024-06-26 18:15:52
Editar	Copiar	Apagar	1505	2	13057	caixa	{“text”: “Caixa”}	0		2024-06-26 18:15:02	2024-06-26 18:15:52
Editar	Copiar	Apagar	1505	2	13057	ups	{“text”: “UPS”}	0		2024-06-26 18:15:02	2024-06-26 18:15:52
Editar	Copiar	Apagar	1505	2	13057	circurometer	{“text”: “Circurometer”}	0		2024-06-26 18:15:02	2024-06-26 18:15:52
Editar	Copiar	Apagar	1505	2	13057	router	{“text”: “Router”}	0		2024-06-26 18:15:02	2024-06-26 18:15:52
Editar	Copiar	Apagar	1505	2	13057	cabin	{“text”: “Cabin”}	0		2024-06-26 18:15:02	2024-06-26 18:15:52
Editar	Copiar	Apagar	1505	2	13057	descricao	{“text”: “Descrição”}	0	Desebre a avana	2024-06-26 18:15:02	2024-06-26 18:15:52

Figura 4.13: Atualização do `listId` da Tabela de `ost_backup_form_field` para a Tabela `ost_form_field`

Este processo é desencadeado quando o plugin é reativado, assegurando

que todas as informações são restauradas e o plugin pode continuar a funcionar sem perda de dados.

Exemplo de Utilização

Um exemplo da utilização deste procedimento pode ser observado na classe principal do plugin, onde o método `cleanUpInstanceData` é responsável por chamar o método `keepPluginInfo` para realizar o backup quando necessário:

Listagem 10: Chamada do método `keepPluginInfo()`

```

1 private function cleanUpInstanceData($instance) {
2     $form_id = $this->getSelectedFormIdByInstance($instance);
3     if ($instance->getConfig()->get('save_on_deactivate')) {
4         PluginDataBaseManager::keepPluginInfo($this->createCamposObj());
5     }
6     $result = PluginDataBaseManager::removerCamposNecessarios($form_id,
7     $this->createCamposObj());
8     if ($result) {
9         // Campos removidos com sucesso
10    } else {
11        // Falha ao remover campos
12    }
13 }
```

Portanto, se a opção `save_on_deactivate` estiver ativada, o método `keepPluginInfo` será invocado para realizar o backup dos nossos campos, garantindo que todos os dados são preservados.

4.3.6 Implementação de Campos de Filtragem através de AJAX

Como mencionado anteriormente, será disponibilizado um campo para seleção da cabine para reporte de avarias. No entanto, em ambiente de produção do projeto SINCRO, existem mais de 100 cabines, o que pode tornar difícil a seleção da cabine correta. Para resolver este problema, decidimos implementar campos de filtragem de acordo com o distrito, estrada e sentido da cabine, reduzindo assim a lista de cabines a selecionar.

Contudo, surgiu outro problema, dado que o `osTicket` não suporta campos dependentes entre si. A única solução foi desenvolver esta funcionalidade modificando o core do `osTicket`. Realizámos *patches* nos ficheiros `class.forms.php`, `ticket-open.inc.php` e `open.inc.php` para suportar essa funcionalidade.

Os *patches* para os campos de filtragem foram implementados nos ficheiros `ticket-open.inc.php` e `open.inc.php`, que são os arquivos *frontend* da *interface* para abertura de `tickets` para o agente e para o cliente, respetivamente. Estes arquivos foram modificados para suportar as novas funcionalidades utilizando *JavaScript* com *AJAX* e de código PHP para obter informações internas, como dados da base de dados ou prefixos das Tabelas. O *JavaScript* será responsável por gerir as filtragens dos campos e modificar a lista de cabines com base nas seleções dos campos de filtragem.

A Figura 4.14 mostra um exemplo de seleção dos campos de filtragem.

Ticket Details
Please Describe Your Issue

Lista de Cabines
Selecione uma Cabine

Setubal ▾ | IC1 ▾ | AF/AP ▾
3:100,3,3;M-Box,Micotec,SN-B-SIN-3;IC1,Setubal;km 548.590;D;AF/AP;38.5122;-8.61138;10.10.1.31

Figura 4.14: Exemplo de seleção dos campos de filtragem

No `class.forms.php`, foi adicionado um *patch* de uma linha para permitir que o nosso *JavaScript* aceda ao componente *frontend* da lista de cabines. O `osTicket` codifica os *names* e *ids* do *frontend* por questões de segurança. Adicionámos uma linha de código para que, no caso do nosso campo da lista de cabines, essa codificação não ocorra, permitindo o acesso através do *JavaScript*. A Listagem 11 demonstra o método responsável pela codificação dos *names* e *ids*, com o *patch* aplicado entre os comentários *custom code start* e *custom code end*.

Listagem 11: Patch no método `getFormName()`

```

1 function getFormName() {
2     $default = $this->get('name') ?: $this->get('id');
3     // Custom code start
4     if ($default === 'lista_cabines') return $default;
5     // Custom code end
6     if ($this->_form && is_numeric($fid = $this->_form->getFormId()))
7         return substr(md5(
8             session_id() . "-form-field-id-$fid-$default-" . SECRET_SALT),
9             -14);
10    elseif (is_numeric($this->get('id')))
11        return substr(md5(
12            session_id() . '-field-id-' . $this->get('id') . '-' . SECRET_SALT
13        ), -16);
14    return $default;
}

```

Este *patch* foi realizado pela classe `RadarDetailsPlugin` (classe principal do `plugin`), utilizando o algoritmo presente na Listagem 12.

Listagem 12: Algoritmo para o *Patch* da classe `class.forms.php`

```

1 $filepath = INCLUDE_DIR . 'class.forms.php';
2 $tag_start = CUSTOM_CODE_START;
3 $tag_end = CUSTOM_CODE_END;
4 $custom_code = "if (\$default === '') . NAME_LISTA_CABINES . '') return \
    \$default;";
5
6 if (!file_exists($filepath)) {
7     $this->logger(MSG_FILE_NOT_FOUND . $filepath);
8     return;
9 }
10
11 $contents = file_get_contents($filepath);
12 $pattern = '/(function getFormName\(\)\s*\{[^}\]*\$default\s*=\s*\$this->get
    \(\\'name\'\)\s*\?\:\s*\$this->get\(\\'id\'\)\s*\;)/';
13
14 if (strpos($contents, $tag_start) !== false) {
15     return;
16 }
17
18 $replacement = "$1\n        $tag_start\n        $custom_code\n
        $tag_end";
19 $contents = preg_replace($pattern, $replacement, $contents);
20
21 file_put_contents($filepath, $contents);

```

Este algoritmo aplica o *patch* procurando o local adequado no código, através de uma expressão regular onde queremos introduzir a modificação e, se ainda não existir, insere o código entre as *tags start* e *end*.

Para os ficheiros `open.inc.php` e `ticket-open.inc.php`, o algoritmo é semelhante, mas como não há um local específico para a modificação, inserimos o código no fim do ficheiro, garantindo assim que o código existente não seja prejudicado.

Para remover os *patches* quando o `plugin` é desativado, utilizamos um algoritmo semelhante. Este algoritmo procura pelos *patches* nos ficheiros, identificados pelas *tags* de início e fim de código personalizado (*custom code start* e *custom code end*), e remove-os de forma segura.

Em relação ao AJAX no JavaScript, a Listagem 13 exemplifica parte do código que realiza a chamada AJAX.

Listagem 13: Função AJAX

```

1 function fetchAndUpdateOptions(selectId, tableName, columnName, filterValue)
{
2     if (isFetching) return;
3     isFetching = true;
4

```

```

5     var url = 'ajax.php/ajax-options/getOptions/' + tablePrefix + tableName
6     + '/' + columnName + '/' + filterValue;
7
8     $.ajax({
9         url: url,
10        type: 'GET',
11        dataType: 'json',
12        success: function(data) {
13            const select = document.getElementById(selectId);
14            const placeholder = 'Selecione uma opção';
15            updateOptions(select, data, placeholder);
16            isFetching = false;
17        },
18        error: function(xhr, status, error) {
19            console.error("Plugin AJAX Error: " + status, error);
20            isFetching = false;
21        }
22    });

```

Esta função é chamada para realizar a filtragem e atualização da lista de opções, utilizando o URL para a chamada **AJAX**.

Para que o URL funcione, ele deve ser registrado no **dispatcher** do **osTicket**. Para isso, o **osTicket** disponibiliza um sistema de sinais (*signals*) que podemos conectar ou enviar. A Listagem 14 apresenta a subscrição ao sinal para registar o **AJAX** tanto para o cliente quanto para o agente (**ajax.client** e **ajax.scp**, respetivamente), sendo chamado no **bootstrap** do **plugin**.

Listagem 14: Subscrição ao Sinal **AJAX**

```

1 function bootstrap() {
2     Signal::connect('ajax.scp', [$this, 'registerDispatchsStaff']);
3     Signal::connect('ajax.client', [$this, 'registerDispatchsClient']);
4     ...
5 }

```

Ao conectarmos ao sinal e o mesmo ser chamado, o mesmo invoca o método **registerDispatchsStaff** para o **staff** e **registerDispatchsClient** para o cliente. A Listagem 15 apresenta o método **registerDispatchsStaff**, sendo que para o cliente é exatamente igual.

Listagem 15: Registro de Rotas **AJAX**

```

1 function registerDispatchsStaff($dispatcher, $data) {
2     $optionsRoute = url('/ajax-options/', patterns(
3         'plugins/PRJ_LEIM_SINCRO/AjaxOptionsController.php:
4             AjaxOptionsController',
5             url_get('^getOptions/(?P<tableName>[^/]+)/(?P<columnName>[^/]+)/(?P<
6             filterValue>[^/]+)$', 'getOptions')
7     ));
8     $dispatcher->append($optionsRoute);
9 }

```

Este método segue a forma como as restantes *URL* do AJAX são registadas na classe `ajax.php` no diretório `install_root/upload/scp`. Para registar a rota, definimos um padrão de *URL* e uma classe que irá tratar os pedidos, neste caso, a classe `AjaxOptionsController`. Esta classe estende a `AjaxController` do `osTicket` e é responsável por tratar as solicitações AJAX, verificando se a solicitação está correta e enviando a resposta adequada de acordo com o pedido.

Apesar de ser bem-sucedida, esta implementação apresenta algumas limitações, principalmente o fato de ficar limitada e comprometida com a versão do `osTicket` instalada na máquina. Mesmo assim, as vantagens são superiores às desvantagens, pois melhoraram significativamente a experiência do utilizador.

Capítulo 5

Demonstrador do Plugin

Neste capítulo, demonstramos a utilização do nosso **plugin**, explicando como configurá-lo para uma utilização bem-sucedida na Secção 5.2 e fornecendo um exemplo de criação de **ticket** na Secção 5.4. Adicionalmente, dedicamos uma secção para explicar as versões das tecnologias utilizadas no desenvolvimento deste **plugin** na Secção 5.1. Este capítulo inclui exemplos de utilização, explora o ambiente de execução, detalha as versões das tecnologias empregadas e descreve as etapas de instalação e desinstalação do **plugin**. Além disso, apresentamos uma demonstração de como integrar o nosso **plugin** com o **plugin** osTicket API do Projeto 15 na Secção 5.5.

5.1 Ambiente de Desenvolvimento

O desenvolvimento do **plugin** foi realizado utilizando as linguagens de programação PHP e *JavaScript*, com o objetivo de garantir uma integração eficiente e eficaz com o sistema do **osTicket**. Durante o processo de desenvolvimento, foram empregadas as seguintes versões de software:

- Versão 5.2.1 do phpMyAdmin
- Versão 8.2.12 do PHP
- Versão 1.18.1 do **osTicket**

O progresso do desenvolvimento do nosso projeto, foi mantido num repositório *Git* na plataforma *GitHub* [1].

5.2 Instalação

Esta Secção indica os passos a tomar para instalar o plugin, de forma correta.

Preparação Fazer *download* da pasta *.zip* do plugin e extrair os seus conteúdos.

Relocar a Pasta Mover a pasta (já com os conteúdos extraídos) do plugin para a diretoria de plugins do osTicket (<osTicket>/include/plugins).

Acesso a Admin Panel Dentro do osTicket, aceder a Admin Panel > Manage > Plugins.



Figura 5.1: página Admin Panel Plugins

Instalar Selecionar o plugin **Radar Details**, da lista de plugins disponíveis, e clicar no botão **Install**.

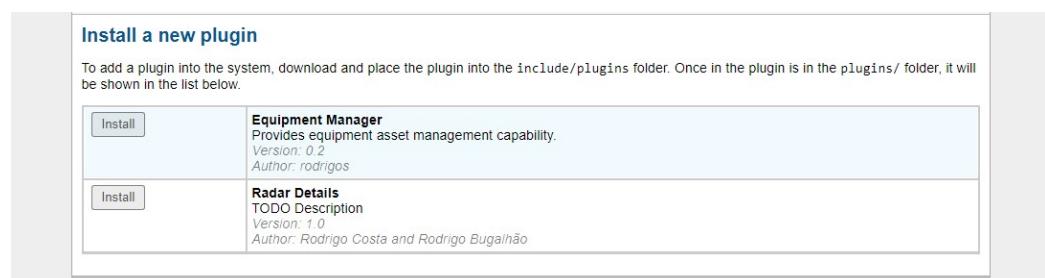


Figura 5.2: página Plugin Installation

Nota Sobre Ativação do Plugin O plugin deve ser ativado, apenas depois da configuração do mesmo, para garantir o seu correto funcionamento. Se o plugin estiver ativo, é necessário desativa-lo para mudar a sua configuração.

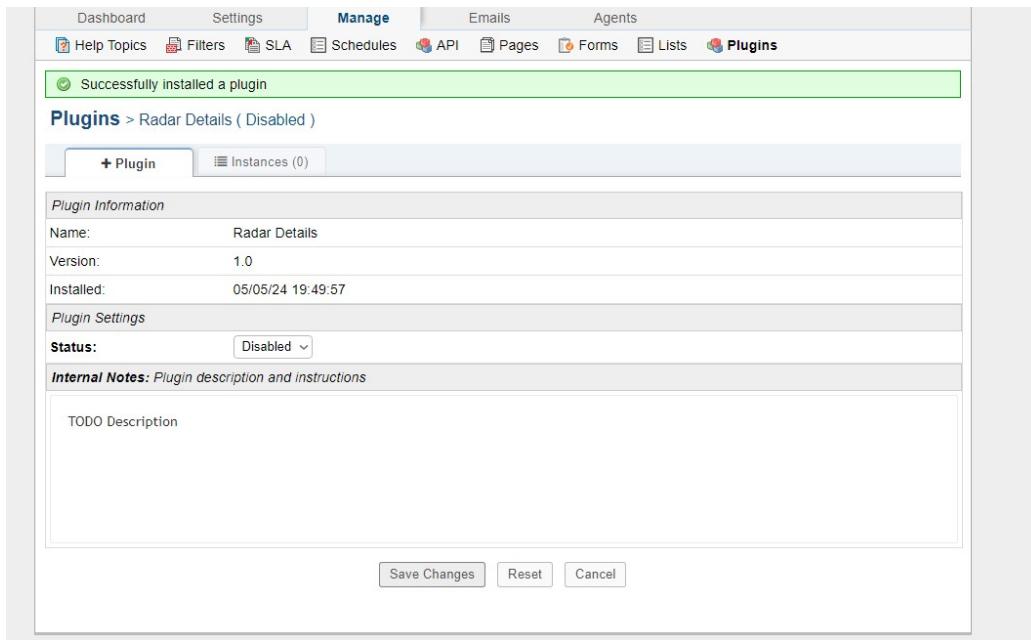


Figura 5.3: página Plugin Installed

Instâncias do Plugin No menu Instances, adicionar e configurar novas instâncias conforme necessário.



Figura 5.4: página Add New Instance

Mudar o *status* da(s) nova(s) instância(s) para **Enabled**.

Plugins > Radar Details (Disabled) — Add New Instance

Instance Name and Status

Name:

Status:

Internal Notes: Instance description and notes

Add Instance **Reset** **Cancel**

Figura 5.5: página Instance

Definições da Configuração Ajustar as definições conforme a necessidade, incluindo a seleção do formulário à qual adicionar os novos campos do plugin, e a indicação do uso da funcionalidade de backup.

Help Topics **Filters** **SLA** **Schedules** **API** **Pages** **Forms** **Lists** **Plugins**

Plugins > Radar Details (Disabled) — Update Instance

Instance **Config**

Select Form:

Save Data on Deactivation: Enable this to save form data to a file when the plugin is deactivated.

Save Changes **Reset** **Cancel**

Copyright © 2006-2024 Uma Empresa LDA. All Rights Reserved.

Figura 5.6: página Instance Configuration

Aplicar a Configuração Depois de definir a configuração da instância, salvar a mesma clicando no botão **Save Changes**.

Ativação do Plugin Depois das configurações necessárias, ativar o plugin, para garantir que as mudanças tomem efeito.

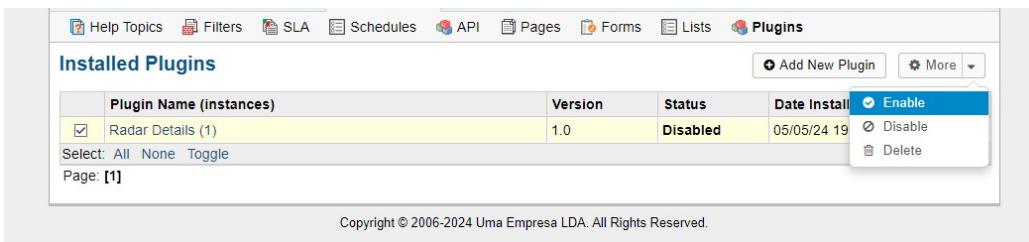


Figura 5.7: página Enable Plugin

Modificações Futuras Alterações à configuração de uma instância requerem a desativação e remoção da mesma, e a criação de uma instância nova.

5.3 Desinstalação

Esta Secção indica os passos a tomar para desinstalar o plugin, de forma correta.

Desinstalação do Plugin Para desinstalar o plugin:

1. Se o plugin estiver ativo, é preciso desativa-lo na página Admin Panel > Manage > Plugins.
2. Depois da desativação, remover o plugin através da opção Delete.

Desinstalação de uma Instância Para remover uma instância:

1. Se o plugin estiver ativo, é necessário desativa-lo.
2. Navegar para a página Instances.
3. Selecionar a instância a remover, e usar a opção remove.

Ao seguir estes passos, é possível garantir que o plugin ou as suas instâncias são removidas corretamente, sem deixar dados ou configurações residuais.

5.4 Demonstração de Utilização Manual

Nesta Secção, ilustramos um exemplo de uso manual do plugin.

Primeiro, depois da configuração inicial realizada (com funcionalidade de backup ativa), abrimos um novo **ticket**, que utilize o formulário afetado pelo plugin.

A seleção da cabine, é facilitada através de uma filtragem por distrito, estrada e sentido. Quando um elemento de filtragem é selecionado, os seguintes elementos também são atualizados. A Figura 5.8 mostra o processo de filtragem do total de cabines até só sobrar a cabine desejada.

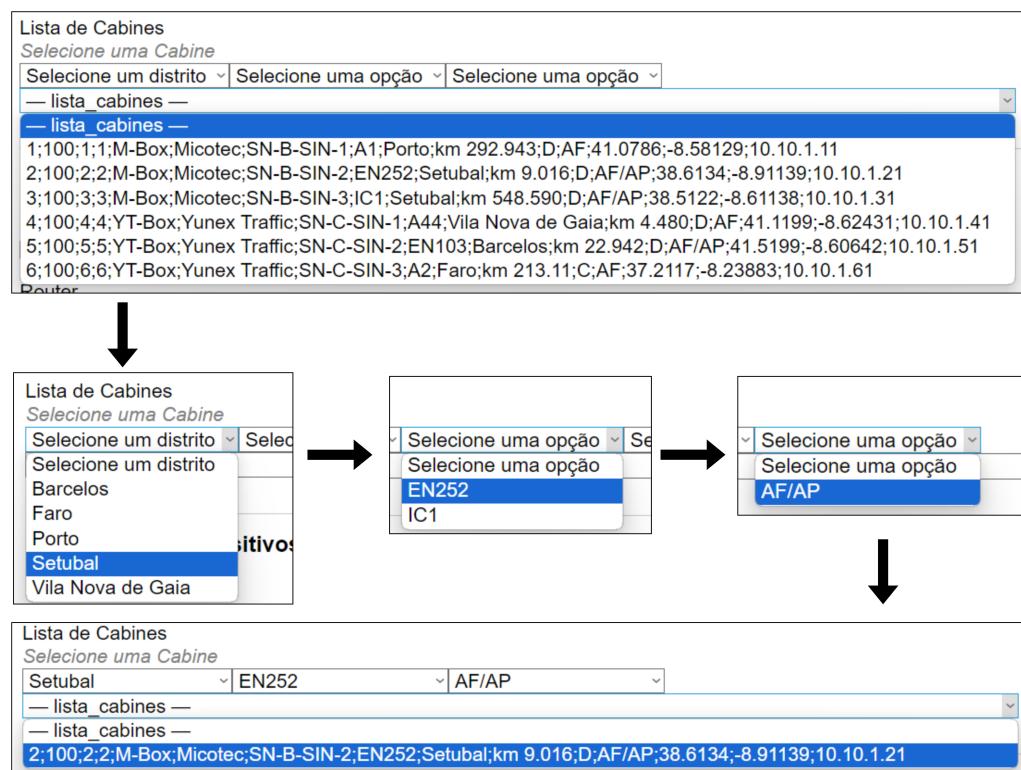


Figura 5.8: Seleção da cabine

Depois basta selecionar, através de elementos *check-box*, os dispositivos com avaria. Podemos também adicionar um texto para explicar a avaria ou transmitir mais informações que não sejam possíveis transmitir de forma estruturada.

The screenshot shows the ANSR Support Center Home page with the following details:

- Header:** ANSR - AUTORIDADE NACIONAL DE SEGURANÇA RODOVIÁRIA. Guest User | Sign In.
- Main Navigation:** Support Center Home, Open a New Ticket, Check Ticket Status.
- Section: Open a New Ticket**
 - Contact Information:**
 - Email Address *: umEmail@gmail.com
 - Full Name *: Nome Completo
 - Phone Number: 987654321 Ext: [empty]
 - Help Topic:** Personal Computer Issues *
- Ticket Details:**
 - Please Describe Your Issue:
Lista de Cabines
Selecionar uma Cabine
Setubal EN252 AF/AP 2:100.2;2;M-Box;Micotec;SN-B-SIN-2;EN252;Setubal;km 9.016;D AF/AP;38.6134;-8.91139;10.10.1.21
 - Selecionar os dispositivos com avarias:**
 - Cabine: Cabine
 - Router: Router
 - Cinemômetro: Cinemômetro
 - UPS: UPS
 - Caixa: Caixa
 - Outro: Outro
 - Descrição:** Descreve a avaria
- Issue Summary:** um probleminha
- Text Area:** Temos problemas no router e no ups
- File Upload:** all changes saved
Drop files here or choose them
- Action Buttons:** Create Ticket, Reset, Cancel
- Page Footer:** Copyright © 2024 ANSR - All rights reserved.
powered by

Figura 5.9: Abrir ticket

No final de preencher os campos do ticket basta clicar em **Create Ticket**, que este será adicionado à base de dados.

Na Figura 5.10 podemos ver que o ticket foi efetivamente registado na base de dados.

AUTORIDADE NACIONAL
SEGURANÇA RODOVIÁRIA
Welcome, Rodrigo | Admin Panel | Profile | Log Out

[Dashboard](#)
[Users](#)
[Tasks](#)
[Tickets](#)
[Knowledgebase](#)

[Open](#)
[My Tickets](#)
[Closed](#)
[Search](#)
[New Ticket](#)

Ticket #119431

um probleminha

Status: Open

Priority: Normal

Department: Support

Create Date: 06/26/24 18:17:55

User: Nome COMpleto (1)

Email: umEmail@gmail.com

Source: Web ((1))

Assigned To: — Unassigned —

SLA Plan: Default SLA

Due Date: 06/28/24 17:00:00

Help Topic: Personal Computer Issues

Last Message: 06/26/24 18:17:56

Last Response:

Ticket Details

Lista de Cabines:	2;100;2;2;M-Box,Micotec,SN-B-SIN-2;EN252;Setubal;km 9.016;D;AF/AP;38.6134;-8.91139;10.10.1.21
Cabine:	No
Router:	Yes
Cinemómetro:	No
UPS:	Yes
Caixa:	No
Outro:	No
Descrição:	—Empty—

Figura 5.10: Ticket guardado

Nas Figuras 5.11, 5.12 e 5.13 podemos confirmar que as informações do ticket foram adicionadas à base de dados.

<input type="checkbox"/>						1503	2	13057 memo	Descrição	descricao ("desc" "Descrição")	0	Descreve a avaria	2024-06-26 18:15:52	2024-06-26 18:15:52
<input type="checkbox"/>						1502	2	13057 bool	Outro	outro ("desc" "Outro")	0		2024-06-26 18:15:52	2024-06-26 18:15:52
<input type="checkbox"/>						1511	2	13057 bool	Caixa	caixa ("desc" "Caixa")	0		2024-06-26 18:15:52	2024-06-26 18:15:52
<input type="checkbox"/>						1510	2	13057 bool	UPS	ups ("desc" "UPS")	0		2024-06-26 18:15:52	2024-06-26 18:15:52
<input type="checkbox"/>						1509	2	13057 bool	Cinémodômetro	cinemometro ("desc" "Cinémodômetro")	0		2024-06-26 18:15:52	2024-06-26 18:15:52
<input type="checkbox"/>						1508	2	13057 bool	Router	router ("desc" "Router")	0		2024-06-26 18:15:52	2024-06-26 18:15:52
<input type="checkbox"/>						1507	2	13057 bool	Cabine	cabine ("desc" "Cabine")	0		2024-06-26 18:15:52	2024-06-26 18:15:52
<input type="checkbox"/>						1506	2	13057 break	Selecionar os dispositivos com avarias	break_avarias []	0		2024-06-26 18:15:52	2024-06-26 18:15:52
<input type="checkbox"/>						1505	2	13057 list-20	Lista de Cabines	lista_cabines []	0	Seleciona uma Cabine	2024-06-26 18:15:52	2024-06-26 18:15:52
						1504	2	13057 memo	Informações da Avaria	avaria ("desc" "Informações da Avaria")	0		2024-06-26 18:15:52	2024-06-26 18:15:52
						1503	2	487605 issue	Issue	issue ("desc" "Issue")	1	NULL	2024-06-26 19:20:23	2024-06-26 19:20:23
						1502	2	414930 memo	Internal Notes	description ("desc" "Description")	2	Details on the reason(s) for creating the ticket.	2024-06-26 19:20:23	2024-06-26 19:20:23
						1501	2	4 487601 ticket	Ticket	tsk ("desc" "length:50")	1	NULL	2024-06-26 19:20:23	2024-06-26 19:20:23
						1500	2	12289 memo	Internal Notes	notes ("desc" "length:40")	5	NULL	2024-06-26 19:20:23	2024-06-26 19:20:23
						1499	2	4 10367 ticket	Website	website ("desc" "length:50")	4	NULL	2024-06-26 19:20:23	2024-06-26 19:20:23
						1498	2	4 13507 phone	Phone	phone ("desc" "length:50")	5	NULL	2024-06-26 19:20:23	2024-06-26 19:20:23
						1497	2	4 13507 memo	Address	address ("desc" "length:100; 'name' value")	2	NULL	2024-06-26 19:20:23	2024-06-26 19:20:23
						1496	2	4 13507 memo	Address	address ("desc" "length:100; 'name' value")	1	NULL	2024-06-26 19:20:23	2024-06-26 19:20:23
						1495	2	4 12545 memo	Address	address ("desc" "length:64")	4	NULL	2024-06-26 19:20:23	2024-06-26 19:20:23
						1494	2	4 237405 phone	Phone Number	phone ("desc" "length:64")	3	NULL	2024-06-26 19:20:23	2024-06-26 19:20:23
						1493	2	4 274705 ticket	Website	website ("desc" "length:64")	2	NULL	2024-06-26 19:20:23	2024-06-26 19:20:23
						1492	2	3 291249 ticket	Company Name	name ("desc" "length:54")	1	NULL	2024-06-26 19:20:23	2024-06-26 19:20:23
						1491	2	2 274609 priority	Priority Level	priority ("desc" "length:54")	3	NULL	2024-06-26 19:20:23	2024-06-26 19:20:23
						1490	2	2 485047 ticket	Internal Details	message ("desc" "length:100")	2	Details on the reason(s) for opening the ticket.	2024-06-26 19:20:23	2024-06-26 19:20:23
						1489	2	2 485025 issue	Issue Summary	subject ("desc" "length:50")	1	NULL	2024-06-26 19:20:23	2024-06-26 19:20:23

Figura 5.11: Tabela `ost_form_field`

		entry_id	1	field_id	value	value_id
<input type="checkbox"/>	Editar Copiar Apagar	121	1513		NULL	
<input type="checkbox"/>	Editar Copiar Apagar	121	1512	0	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	121	1511	0	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	121	1510	1	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	121	1509	0	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	121	1508	1	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	121	1507	0	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	121	1505	{"182": "2;100;2;2;M-Box;Micotec;SN-B-SIN-2;EN252;S...}	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	121	22	Normal	2	
<input type="checkbox"/>	Editar Copiar Apagar	121	20	um probleminha	NULL	
<hr/>						
<input type="checkbox"/>	Editar Copiar Apagar	120	4	NULL	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	120	3	987654321	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	117	4	NULL	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	117	3	987654321	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	34	897	edw	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	34	896	192.168.1.1	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	34	894	edw	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	34	890	192.168.1.1	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	29	4	NULL	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	29	3	927226841	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	16	4	NULL	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	16	3	NULL	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	15	4	NULL	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	15	3	NULL	NULL	
<input type="checkbox"/>	Editar Copiar Apagar	12	4	NULL	NULL	

Figura 5.12: Tabela ost_form_entry_values

		id	1	form_id	object_id	object_type	sort	extra	created	updated
<input type="checkbox"/>	Editar Copiar Apagar	122	7	53	T		1	{"disable":[]}]	2024-06-26 18:17:55	2024-06-26 18:17:55
<input type="checkbox"/>	Editar Copiar Apagar	121	2	53	T		0	{"disable":[]}]	2024-06-26 18:17:55	2024-06-26 18:17:55
<input type="checkbox"/>	Editar Copiar Apagar	120	1	8	U		1	NULL	2024-06-26 18:17:55	2024-06-26 18:17:55
<input type="checkbox"/>	Editar Copiar Apagar	117	1	7	U		1	NULL	2024-06-26 18:09:19	2024-06-26 18:09:19
<input type="checkbox"/>	Editar Copiar Apagar	29	1	6	U		1	NULL	2024-04-23 15:50:31	2024-04-23 15:50:31
<input type="checkbox"/>	Editar Copiar Apagar	16	1	5	U		1	NULL	2024-04-16 18:06:59	2024-04-16 18:06:59
<input type="checkbox"/>	Editar Copiar Apagar	15	1	4	U		1	NULL	2024-03-27 16:25:14	2024-03-27 16:25:14
<input type="checkbox"/>	Editar Copiar Apagar	12	1	3	U		1	NULL	2024-03-27 16:11:34	2024-03-27 16:11:34
<input type="checkbox"/>	Editar Copiar Apagar	7	4	2	O		1	NULL	2024-03-27 10:31:32	2024-03-27 10:31:32
<input type="checkbox"/>	Editar Copiar Apagar	6	2	2	T		0	{"disable":[]}]	2024-03-26 19:31:55	2024-03-26 19:31:55
<input type="checkbox"/>	Editar Copiar Apagar	5	1	2	U		1	NULL	2024-03-26 19:28:37	2024-03-26 19:28:37
<input type="checkbox"/>	Editar Copiar Apagar	4	2	1	T		0	{"disable":[]}]	2024-03-26 19:20:24	2024-03-26 19:20:24
<input type="checkbox"/>	Editar Copiar Apagar	3	1	1	U		1	NULL	2024-03-26 19:20:24	2024-03-26 19:20:24
<input type="checkbox"/>	Editar Copiar Apagar	2	3	NULL	C		1	NULL	2024-03-26 19:20:24	2024-03-26 19:20:24
<input type="checkbox"/>	Editar Copiar Apagar	1	4	1	O		1	NULL	2024-03-26 19:20:23	2024-03-26 19:20:23

Figura 5.13: Tabela ost_form_entry

Após o **ticket** ser criado, desativamos o **plugin** e mostramos que os campos e as entradas do formulário, afetado pelo **plugin**, foram eliminados das Tabelas principais da base de dados. Como a funcionalidade de **backup** está ativa, as informações do **ticket** foram adicionadas às Tabelas de **backup**, como podemos confirmar na Figura 5.15. A Figura 5.14 mostra que as in-

formações do **ticket**, que dependiam do **plugin**, foram eliminadas.

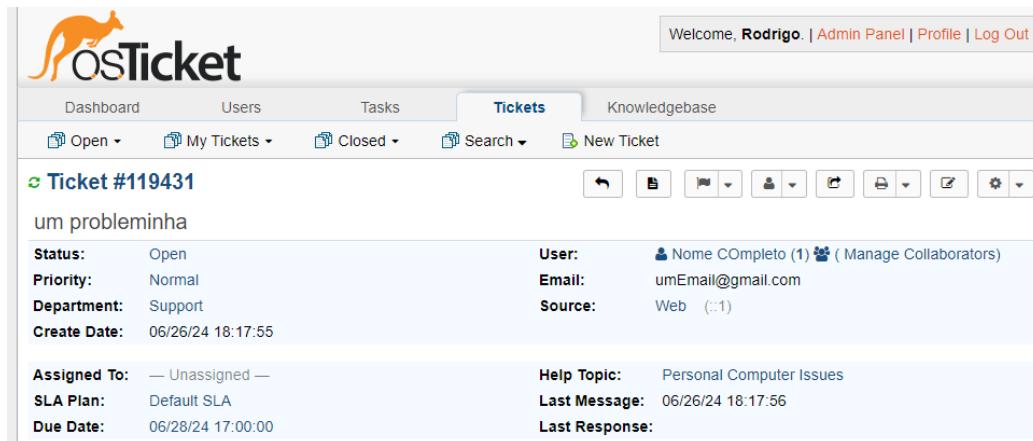


Figura 5.14: ticket depois de desativar o plugin

tabela ost_backup_form_field														
	←	T	→	id	form_id	flags	type	label	name	configuration	sort	hint	created	updated
<input type="checkbox"/>	Editar	Copiar	Apagar	1505	2	13057	list-19	Lista de Cabines	lista_cabines	[]	0	Selecione uma Cabine	2024-06-26 18:15:52	2024-06-26 18:15:52
<input type="checkbox"/>	Editar	Copiar	Apagar	1506	2	13057	break	Seleciona os dispositivos com avarias	break_avarias	[]	0		2024-06-26 18:15:52	2024-06-26 18:15:52
<input type="checkbox"/>	Editar	Copiar	Apagar	1507	2	13057	bool	Cabine	cabine	[{"desc": "Cabine"}]	0		2024-06-26 18:15:52	2024-06-26 18:15:52
<input type="checkbox"/>	Editar	Copiar	Apagar	1508	2	13057	bool	Router	router	[{"desc": "Router"}]	0		2024-06-26 18:15:52	2024-06-26 18:15:52
<input type="checkbox"/>	Editar	Copiar	Apagar	1509	2	13057	bool	Cinemómetro	cinemometro	[{"desc": "Cinemómetro"}]	0		2024-06-26 18:15:52	2024-06-26 18:15:52
<input type="checkbox"/>	Editar	Copiar	Apagar	1510	2	13057	bool	UPS	ups	[{"desc": "UPS"}]	0		2024-06-26 18:15:52	2024-06-26 18:15:52
<input type="checkbox"/>	Editar	Copiar	Apagar	1511	2	13057	bool	Caixa	caixa	[{"desc": "Caixa"}]	0		2024-06-26 18:15:52	2024-06-26 18:15:52
<input type="checkbox"/>	Editar	Copiar	Apagar	1512	2	13057	bool	Outro	outro	[{"desc": "Outro"}]	0		2024-06-26 18:15:52	2024-06-26 18:15:52
<input type="checkbox"/>	Editar	Copiar	Apagar	1513	2	13057	memo	Descrição	descricao	[{"desc": "Descrição"}]	0	Descreve a avaria	2024-06-26 18:15:52	2024-06-26 18:15:52

tabela ost_backup_form_entry_values							
	←	T	→	entry_id	field_id	value	value_id
<input type="checkbox"/>	Editar	Copiar	Apagar	121	1505	(“176”,”2;100;2;2;M-Box,Micotec,SN-B-SIN-2;EN252;S...)	NULL
<input type="checkbox"/>	Editar	Copiar	Apagar	121	1507	0	NULL
<input type="checkbox"/>	Editar	Copiar	Apagar	121	1508	1	NULL
<input type="checkbox"/>	Editar	Copiar	Apagar	121	1509	0	NULL
<input type="checkbox"/>	Editar	Copiar	Apagar	121	1510	1	NULL
<input type="checkbox"/>	Editar	Copiar	Apagar	121	1511	0	NULL
<input type="checkbox"/>	Editar	Copiar	Apagar	121	1512	0	NULL
<input type="checkbox"/>	Editar	Copiar	Apagar	121	1513		NULL

tabela ost_backup_form_entry											
	←	T	→	id	form_id	object_id	object_type	sort	extra	created	updated
<input type="checkbox"/>	Editar	Copiar	Apagar	121	2	53	T	0	{"disable": []}	2024-06-26 18:17:55	2024-06-26 18:17:55

Figura 5.15: Tabelas de backup

Ao voltar a ativar o **plugin**, o conteúdo das Tabelas de **backup** é reintegrado nas Tabelas principais da base de dados. Na Figura 5.16 verificamos que as informações do **ticket** foram repostas.

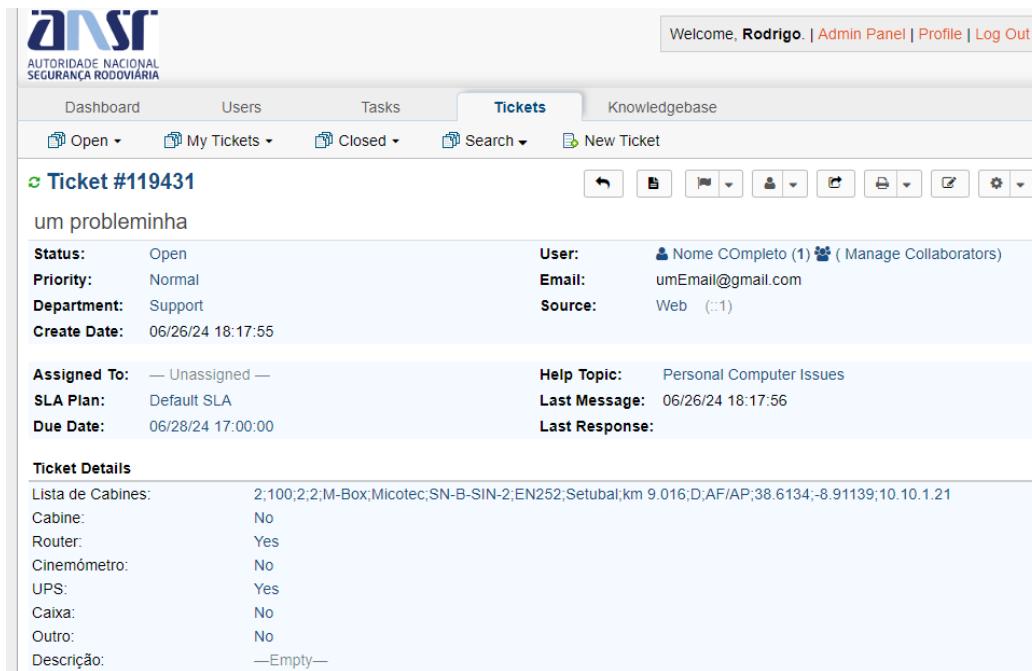


Figura 5.16: ticket reposto

5.5 Demonstração de Utilização do Plugin via *API*

Nesta Secção, apresentamos um exemplo de utilização do plugin *Radares Details* através da *API* desenvolvida pelo grupo do Projeto 15, mencionado no Capítulo 1. Esta demonstração ilustra como o plugin pode ser integrado e utilizado em conjunto com a *API* para criar *tickets* de forma automatizada.

Para obter mais informações detalhadas sobre a utilização do plugin da *API*, consulte a documentação do Projeto 15 disponível online [4].

Para criar um *ticket*, via *API*, é necessário realizar um pedido *http* via a plataforma *Postman*. A Figura 5.17 apresenta um exemplo para criar um *ticket* automaticamente utilizando portanto a *API* do Projeto 15.

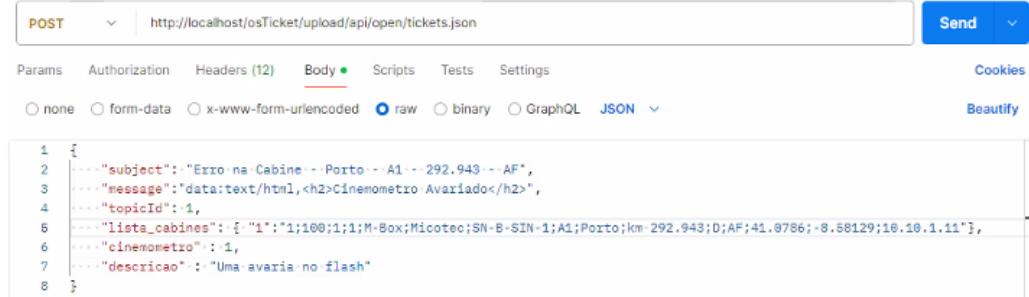


Figura 5.17: Pedido http no *Postman*

Analizando a Figura 5.17, o pedido tem a seguinte estrutura:

- **subject** : o título do ticket.
- **message** : a descrição do problema.
- **topicId** : identificador do tópico.
- **lista_cabines** : a cabine desejada a reportar.
- **cabine, router, cinemometro, ups, caixa, outro** : os vários componentes da cabine. Só é necessário incluir os componentes com avaria, passando-lhes o valor 1.
- **descricao** : uma mensagem opcional para descrever o problema.

Após enviar o pedido é recebida a confirmação que o **ticket** foi criado, como mostrado na Figura 5.18.



Figura 5.18: Resposta no *Postman*

Na Figura 5.19, é possível comprovar que o **ticket** foi criado com sucesso e que o resultado não difere daquele obtido através da utilização manual do plugin.

_ticket #451950

Erro na Cabine - Porto - A1 - 292.943 - AF

Status:	Open	User:	cgoncalves (5) (Manage Collaborators)
Priority:	Normal	Email:	carlos.goncalves@isel.pt
Department:	Support	Source:	API
Create Date:	6/28/24 10:21 AM		
Assigned To:	— Unassigned —	Help Topic:	General Inquiry
SLA Plan:	Default SLA	Last Message:	6/28/24 10:21 AM
Due Date:	7/2/24 10:21 AM	Last Response:	

Ticket Details

Lista de Cabines:	1;100;1;1;M-Box;Micotec;SN-B-SIN-1;A1;Porto;km 292.943;D;AF;41.0786;-8.58129;10.10.1.11
Cabine:	No
Rouler:	No
Cinemómetro:	Yes
UPS:	No
Caixa:	No
Outro:	No
Descrição:	Uma avaria no flash

Figura 5.19: ticket criado através da API do Projeto 15

Capítulo 6

Conclusões e Trabalho Futuro

Neste último capítulo apresentamos as conclusões tiradas, do desenvolvimento deste projeto, e refletimos sobre o produto final. Apresentamos também, o possível trabalho futuro, que pode melhorar e expandir sobre a nossa implementação.

6.1 Conclusões

Este relatório começa com uma introdução à situação de **ticketing** existente entre a ANSR e os prestadores de serviço de manutenção de radares, e os desafios que enfrentam na gestão, criação e interpretação dos **tickets**.

A dificuldade de utilizar **tickets**, em que a informação está exposta em texto livre, ou seja, de forma não estruturada, motivou a criação deste projeto. O trabalho desenvolvido, trata-se de um **plugin** para o **osTicket** que permite a criação de **tickets** para reportar problemas em radares, em que a informação se encontra estruturada e fácil de interpretar.

Ao longo deste documento, apresentamos o desenvolvimento do **plugin** e como se integra com o **osTicket**, bem como atende às necessidades da ANSR e dos prestadores de serviço.

Para atingir o objetivo desejado, começámos por estudar a arquitetura do **osTicket** e como seria possível criar um **plugin**, programaticamente. De seguida começámos a implementar a primeira versão do **plugin**.

A criação de um **plugin** foi realizada em PHP e segue uma estrutura específica, sobre a qual se pode construir.

Para adicionar os campos necessários a um formulário, adicionaram-se

à base de dados, os campos desejados, com a respetiva configuração. Com recurso ao formulário atualizado, o utilizador pode abrir um **ticket**, que estruture a informação necessária sobre uma avaria num radar.

Ao configurar o **plugin**, o utilizador pode ativar a funcionalidade de **backup**, para guardar toda a informação relacionada com o **plugin** adicionada ao **osTicket**. Para guardar essa informação, foram adicionadas tabelas à base de dados do **osTicket**. No futuro, quando o **plugin** voltar a ser ativado, as informações de **backup** podem ser repostas.

Foi também, possível melhorar a experiência de utilização do **plugin**, adicionando uma funcionalidade de filtragem da lista de radares, para uma seleção mais fácil. Esta funcionalidade foi conseguida através da injeção de código *JavaScript* nos ficheiros *core* do **osTicket**.

Com a conclusão deste relatório, observa-se que os objetivos foram cumpridos e, que disponibilizamos um **plugin** funcional, implementado segundo as normas do **osTicket**, que permite a criação de **tickets** estruturados de suporte a avarias em radares, e que pode ser usado manualmente, ou via a *API* criada pelo grupo do projeto 15, pronto a ser usado pela ANSR e prestadores de serviço.

6.2 Trabalho Futuro

Com o sucesso na implementação do **plugin**, possibilitamos que outras funcionalidades sejam criadas com compatibilidade, ou base no mesmo.

A principal opção para trabalho futuro seria a disponibilização de *dashboards* com dados e estatísticas relevantes sobre as avarias registadas. Esta adição seria uma ferramenta poderosa de análise, que permitiria à ANSR observar os radares com mais problemas e tirar conclusões sobre essas avarias, cruzando os dados para identificar as causas mais frequentes de problemas.

Bibliografia

- [1] Rodrigo Costa and Rodrigo Bugalhão. osticket-sincro-project. <https://github.com/RodrigoFMC/osticket-sincro-project>, 2024.
- [2] osTicket. osticket documentation. <https://docs.osticket.com/en/latest/>, June 2022.
- [3] poctob. ostequipmentplugin. <https://github.com/poctob/OSTEquipmentPlugin/wiki/Plugin-Development-Introduction>, 2014.
- [4] Tomás Gomes. Documentação do projeto 15. <https://tomasgomesisel.gitbook.io/projeto-ansr-isel>, 2024.

Apêndice A

Guia Completo do Plugin

O guia completo do plugin, que inclui funcionalidades adicionais, está disponível na página seguinte.

Como Criar um Plugin para o osTicket

Introdução ao Desenvolvimento de Plugins

O desenvolvimento de plugins para o osTicket representa um desafio significativo, principalmente devido à falta de documentação oficial. Este guia baseia-se em informações obtidas a partir de um [blog não oficial](#), que descreve como criar um plugin no osTicket.

Neste guia, focamos na construção de um plugin compatível com a versão 1.18 do osTicket, enquanto o guia original foi elaborado para a versão 1.9.3. Abordaremos os desafios resultantes dessas diferenças de versão e forneceremos soluções adequadas ao longo do processo.

Para que um plugin seja reconhecido pelo osTicket, ele deve estar numa pasta que contenha um ficheiro `plugin.php`, um ficheiro de configuração `Config.php` e um ficheiro principal com a lógica do plugin. Esta pasta deve ser colocada na diretoria `[INSTALL_ROOT]/include/plugins` do OsTicket.

```
[INSTALL_ROOT]
└── include
    └── plugins
        └── nome_do_plugin
            ├── plugin.php
            ├── Config.php
            └── PluginExemplo.php
```

Passo 1: Criar o Ficheiro `plugin.php`

O primeiro passo será criar um ficheiro `plugin.php` que contém a descrição e os dados essenciais do plugin. Abaixo, fornecemos um exemplo prático de como este ficheiro deve ser estruturado:

```
<?php
return array(
    'id' => 'rodrigo:PluginExemplo',
    'version' => '1.0',
    'name' => 'Plugin Exemplo',
    'author' => 'Rodrigo',
    'description' => 'Plugin exemplo que faz alguma coisa.',
    'url' => 'https://github.com/poctob/OSTEquipmentPlugin/wiki/Plugin-
Development-Introduction',
    'plugin' => 'PluginExemplo.php:PluginExemplo'
);
```

Elementos do Array

- **id**: Identificador único para o plugin, ex. 'rodrigo:PluginExemplo'.

- **version**: Versão do plugin, ex. '1.0'.
- **name**: Nome do plugin exibido na interface, ex. 'Plugin Exemplo'.
- **author**: Nome do(s) autor(es), ex. 'Rodrigo'.
- **description**: Breve descrição do plugin.
- **url**: URL para o helpdesk se aplicável.
- **plugin**: Ficheiro principal com a lógica do plugin e a classe a ser instanciada, ex. 'PluginExemplo.php:PluginExemplo'.

Passo 2. Página de Configuração do Plugin

Como mencionado anteriormente, o plugin necessita ter uma classe de configuração para ser reconhecido pelo osTicket. A seguir, apresentamos um exemplo prático da estrutura que esta classe deve adotar:

Exemplo de Ficheiro de Configuração `Config.php`

```
class PluginExemploConfig extends PluginConfig {
    function getOptions() {
        return array(
            'plugin_exemplo_backup' => new BooleanField([
                'id' => 'plugin_exemplo_backup',
                'label' => 'Fazer Backup',
                'configuration' => array(
                    'desc' => 'Ativar ou desativar backups automáticos.'
                )
            ]),
        );
    }

    function pre_save(&$config, &$errors) {
        global $msg;

        // Guardar a configuração antes de salvar
        if (!empty($this->get('plugin_exemplo_backup', ''))) {
            $this->set('plugin_exemplo_backup', $config['plugin_exemplo_backup']);
        }

        if (!$errors)
            $msg = 'Configuration updated successfully';

        return true;
    }

    function post_save(&$config, &$errors) {
        global $msg;

        // Lógica post_save, se necessário
        if (isset($config['plugin_exemplo_backup'])) {
            $msg = 'Configuração post_save realizada com sucesso';
        }

        return true;
    }
}
```

```
    }  
}
```

Detalhes da Configuração

A classe `PluginExemploConfig` deriva de `PluginConfig`, que se encontra localizada em `[INSTALL_ROOT]/include/class.plugin.php`. A função `getOptions()` é projetada para retornar um array com as opções de configuração do plugin.

A opção de configuração `plugin_exemplo_backup` é representada por uma select box com o ID `plugin_exemplo_backup` e o label "Fazer Backup". Esta select box inclui uma descrição que refere a sua funcionalidade "Ativar ou desativar backups automáticos". Os dados configurados são armazenados na tabela `ost_config` na base de dados, designada pelo número do plugin. Cada nova instalação de um plugin aumenta este número, criando namespaces como `plugin.3` ou `plugin.44`. Este sistema permite a instalação e gestão independente de múltiplos plugins, cada um acessando as suas configurações específicas através do método `$this->getConfig()->get('setting');`, garantindo a unicidade de cada configuração. Os dados são armazenados de forma escalar ou em formato JSON no campo de valor da tabela.

Tipos de Campos Disponíveis

Os campos disponíveis para configuração, definidos em `[INSTALL_ROOT]/include/class.forms.php`, incluem:

- `TextboxField` - Caixa de texto simples.
- `TextareaField` - Área de texto para conteúdos mais extensos.
- `ThreadEntryField` - Área de texto, normalmente utilizada para discussões em fóruns.
- `DatetimeField` - Select de datas baseado em JQuery.
- `PhoneField` - Caixa de texto otimizada para a inserção de números de telefone.
- `BooleanField` - select box para opções binárias de verdadeiro/falso.
- `ChoiceField` - Campo dropdown para seleção entre várias opções.
- `SectionBreakField` - Quebra de secção horizontal, que auxilia na organização visual das configurações.

Função `pre_save(&$config, &$errors)`

Esta função é ativada quando as configurações são submetidas pelo utilizador. Realiza a validação do lado do servidor e implementa a lógica necessária para o processamento das configurações. Para indicar uma falha durante as verificações, um erro é adicionado ao array de erros com `$errors['err'] = "mensagem";`, e a função retorna `false` em `pre_save`, impedindo a atualização das configurações.

Função `post_save(&$config, &$errors)`

Esta função é chamada após as configurações terem sido guardadas com sucesso. Aqui podemos adicionar qualquer lógica que precise de ser executada após a atualização das configurações. Por exemplo, enviar notificações ou atualizar caches.

Passo 3. Classe Principal do Plugin

Estrutura do Plugin

Para que um plugin seja válido no osTicket, a classe do plugin deve incluir apenas um método obrigatório: o método `bootstrap()`. Os demais métodos são opcionais e servem para ampliar a funcionalidade do plugin.

Tendo isso em mente, o próximo passo é criar um ficheiro denominado `PluginExemplo.php` para definir a classe principal do plugin, que deve conter toda a lógica necessária. Esta classe deve estender `Plugin` e incluir os métodos de inicialização e configuração do plugin. No exemplo abaixo, alguns métodos não estão completamente implementados, pois o objetivo principal é demonstrar a estrutura do plugin e não a lógica completa de funcionalidades específicas.

```
<?php
require_once(INCLUDE_DIR . 'class.plugin.php');
require_once(INCLUDE_DIR . 'class.signal.php');
require_once(INCLUDE_DIR . 'class.osticket.php');
require_once(__DIR__ . '/config.php');

define('PLUGIN_DIR', __DIR__ . '/');
define('EXAMPLE_TABLE', TABLE_PREFIX . 'example_table');
define('SESSION_PREVIOUS_STATE_PLUGIN_PREFIX', 'previousActiveStatePlugin_');

class PluginExemplo extends Plugin {
    public const PLUGIN_NAME = 'PluginExemplo';
    var $config_class = 'PluginExemploConfig';

    /**
     * Inicializa o plugin.
     *
     * @return void
     */
    function bootstrap() {
        // Conectar ao sinal de criação de tickets
        Signal::connect('model.created', array($this, 'onTicketCreated'),
        'Ticket');

        if ($this->firstRun() && !$this->configureFirstRun()) {
            error_log('Falha ao configurar a primeira execução do plugin.');
        }

        if ($this->needUpgrade()) {
            $this->configureUpgrade();
        }

        // Realiza as ações específicas da instância
        $this->makeInstanceSpecificActions();
    }

    /**
     * Realiza as ações específicas da instância.
     *
     * @return void
     */
    private function makeInstanceSpecificActions() {
        $config = $this->getConfig();
```

```
    if ($config->get('plugin_exemplo_backup') && parent::isActive()) {
        $this->realizarBackup();
    }

}

/**
 * Função chamada quando um ticket é criado.
 *
 * @param Ticket $ticket O ticket recém-criado.
 * @return void
 */
function onTicketCreated($ticket) {
    error_log('Novo ticket criado: ID ' . $ticket->getId());
    // Adicionar lógica adicional a ser executada quando um ticket é criado
}

/**
 * Verifica se o plugin está ativo globalmente.
 *
 * @return bool True se ativo, false caso contrário.
 */
function isActive() {
    $currentlyActive = parent::isActive();
    $previouslyActive = $this->getCurrentActiveStatePlugin();

    if ($previouslyActive !== $currentlyActive) {
        $this->handleActivationChange($currentlyActive);
        $this->setCurrentActiveStatePlugin($currentlyActive);
    }

    return $currentlyActive;
}

/**
 * Manipula a mudança de ativação do plugin.
 *
 * @param bool $currentlyActive O estado atual do plugin.
 * @return void
 */
private function handleActivationChange($currentlyActive) {
    error_log("Plugin está a ser " . ($currentlyActive ? "ativado" :
"desativado") . ".");
    if ($currentlyActive) {
        $this->enablePlugin();
    } else {
        $this->disablePlugin();
    }
}

/**
 * Lógica a ser executada quando o plugin é ativado.
 *
 * @return void
*/
```

```
function enablePlugin() {
    if ($this->createDBTables()) {
        error_log("Plugin Exemplo ativado com sucesso!");
    } else {
        error_log("Erro ao ativar o Plugin Exemplo.");
    }
}

/**
 * Lógica a ser executada quando o plugin é desativado.
 *
 * @return void
 */
function disablePlugin() {
    if ($this->cleanUp()) {
        error_log("Plugin Exemplo desativado com sucesso!");
    } else {
        error_log("Erro ao desativar o Plugin Exemplo.");
    }
}

/**
 * Realiza o backup conforme a configuração do plugin.
 *
 * @return void
 */
function realizarBackup() {
    error_log("Backup realizado com sucesso pelo Plugin Exemplo!");
    // Adicionar lógica de backup
}

/**
 * Verifica se esta é a primeira execução do plugin.
 *
 * @return bool True se for a primeira execução, false caso contrário.
 */
function firstRun() {
    $initialized = !$this->getConfig()->get('plugin_exemplo_initialized',
false);
    error_log("firstRun retornou " . ($initialized ? "true" : "false") . ".");
    return $initialized;
}

/**
 * Verifica se o plugin precisa ser atualizado.
 *
 * @return bool True se precisar de atualização, false caso contrário.
 */
function needUpgrade() {
    return false; // Lógica para determinar se o plugin precisa ser atualizado
}

/**
 * Configurações necessárias para a primeira execução do plugin.
```

```
* @return bool True se a configuração inicial foi bem-sucedida, false caso
contrário.
*/
function configureFirstRun() {
    $this->getConfig()->set('plugin_exemplo_initialized', true);
    return true;
}

/**
 * Lida com a lógica de atualização do plugin.
 *
 * @return void
 */
function configureUpgrade() {
    // Lógica de atualização do plugin
}

/**
 * Cria tabelas na base de dados.
 *
 * @return bool True se a criação das tabelas foi bem-sucedida, false caso
contrário.
*/
function createDBTables() {
    $sql = "CREATE TABLE IF NOT EXISTS " . EXAMPLE_TABLE . " (
        id INT(11) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(255) NOT NULL,
        description TEXT,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;";

    if (db_query($sql)) {
        error_log("Tabela " . EXAMPLE_TABLE . " criada com sucesso.");
        return true;
    } else {
        error_log("Falha ao criar tabela " . EXAMPLE_TABLE . ".");
        return false;
    }
}

/**
 * Função de limpeza ao desativar o plugin.
 *
 * @return bool True se a remoção das tabelas foi bem sucedida, false caso
contrário.
*/
function cleanUp() {
    $sql = "DROP TABLE IF EXISTS `". EXAMPLE_TABLE . "`;";

    if (db_query($sql)) {
        error_log("Tabela " . EXAMPLE_TABLE . " removida com sucesso.");
        return true;
    } else {
```

```
        error_log("Erro ao remover a tabela " . EXAMPLE_TABLE . ".");
        return false;
    }

    /**
     * Define o estado ativo atual na sessão para o plugin.
     *
     * @param bool $state O estado ativo atual.
     */
    private function setCurrentActiveStatePlugin($state) {
        $_SESSION[SESSION_PREVIOUS_STATE_PLUGIN_PREFIX] = $state;
    }

    /**
     * Recupera o estado ativo atual do plugin.
     *
     * @return bool|null O estado ativo atual ou null se não estiver definido.
     */
    private function getCurrentActiveStatePlugin() {
        return $_SESSION[SESSION_PREVIOUS_STATE_PLUGIN_PREFIX] ?? null;
    }
}
```

Resumo da Classe do Plugin

O código listado acima compõe um plugin de demonstração para o osTicket, projetado para mostrar a estrutura básica e algumas funcionalidades. As principais funcionalidades incluem:

- **Resposta a sinais do sistema:** O plugin reage a eventos gerados pelo osTicket, permitindo a execução de ações específicas, como quando um novo ticket é criado.
- **Gestão de estados do plugin:** Permite saber se é necessário remover ou manter as alterações feitas pelo plugin, dependendo do seu estado atual (ativado ou desativado).
- **Criação e remoção de tabelas na base de dados:** Demonstra como o plugin interage com a base de dados para criar e remover tabelas, facilitando o armazenamento e a gestão de dados específicos.
- **Configuração específica para cada instância:** Suporta múltiplas instâncias do plugin, permitindo configurações e estados distintos para cada uma delas. Cada instância pode realizar ações específicas baseadas nas suas configurações.
- **Primeira execução (first run):** Verifica se é a primeira vez que o plugin está a ser executado e realiza as configurações iniciais necessárias.
- **Atualizações (upgrades):** Verifica se o plugin precisa ser atualizado e aplica as configurações de atualização quando necessário.

Classe [PluginExemplo.php](#) em detalhe

Configuração

A classe `PluginExemplo` estende `Plugin` e utiliza a classe de configuração `PluginExemploConfig`. Isso permite que o plugin utilize as opções de configuração definidas na classe `PluginExemploConfig`, como a opção para ativar ou desativar backups automáticos.

```
var $config_class = 'PluginExemploConfig';
```

Inicialização

A função `bootstrap()` é chamada pelo `osTicket` para inicializar o plugin. É chamado sempre que um utilizador acede ao `OsTicket`. Esta função:

- Conecta-se ao sinal de criação de tickets (`model.created`) para executar uma função (`onTicketCreated`) sempre que um novo ticket é criado.

```
Signal::connect('model.created', array($this, 'onTicketCreated'), 'Ticket');
```

- Verifica se é a primeira vez que o plugin está a ser executado através da função `firstRun()`. Se for a primeira execução, chama `configureFirstRun()` para realizar as configurações iniciais.

```
if ($this->firstRun() && !$this->configureFirstRun()) {  
    error_log('Falha ao configurar a primeira execução do plugin.');
```

- Verifica se o plugin precisa ser atualizado através da função `needUpgrade()`. Se precisar, chama `configureUpgrade()`.

```
if ($this->needUpgrade()) {  
    $this->configureUpgrade();  
}
```

- Executa ações específicas da instância através da função `makeInstanceSpecificActions()`, como verificar a configuração de backup e realizar o backup se necessário.

```
$this->makeInstanceSpecificActions();
```

Funcionalidades

A função `realizarBackup()` implementa as funcionalidades de backup do plugin. Esta função é chamada se a configuração `plugin_exemplo_backup` estiver ativada e o plugin estiver ativo.

```
function realizarBackup() {
    error_log("Backup realizado com sucesso pelo Plugin Exemplo!");
    // Adicionar lógica de backup
}
```

Ativação

A função `isActive()` verifica se o plugin está ativo. Esta função também lida com a lógica de ativação e desativação do plugin:

- Obtém o estado atual do plugin (`currentlyActive`) e o estado anterior (`previouslyActive`).

```
$currentlyActive = parent::isActive();
$previouslyActive = $this->getCurrentActiveStatePlugin();
```

- Se o estado mudou, chama `handleActivationChange()` para lidar com a ativação ou desativação do plugin.

```
if ($previouslyActive !== $currentlyActive) {
    $this->handleActivationChange($currentlyActive);
    $this->setCurrentActiveStatePlugin($currentlyActive);
}
```

- Atualiza o estado atual do plugin na sessão através das funções `setCurrentActiveStatePlugin` e `getCurrentActiveStatePlugin`.

```
private function setCurrentActiveStatePlugin($state) {
    $_SESSION[SESSION_PREVIOUS_STATE_PLUGIN_PREFIX] = $state;
}

private function getCurrentActiveStatePlugin() {
    return $_SESSION[SESSION_PREVIOUS_STATE_PLUGIN_PREFIX] ?? null;
}
```

Primeira Execução

A função `firstRun()` verifica se é a primeira vez que o plugin está a ser executado, verificando a configuração `plugin_exemplo_initialized`.

```
function firstRun() {
    $initialized = !$this->getConfig()->get('plugin_exemplo_initialized', false);
    error_log("firstRun retornou " . ($initialized ? "true" : "false") . ".");
```

```
    return $initialized;
}
```

Atualização do Plugin

A função `needUpgrade()` verifica se o plugin precisa ser atualizado. No exemplo fornecido, esta função retorna sempre false, mas pode ser modificada para incluir lógica para verificação de versões.

```
function needUpgrade() {
    return false; // Lógica para determinar se o plugin precisa ser atualizado
}
```

Configuração Inicial

A função `configureFirstRun()` realiza as configurações necessárias para a primeira execução, como marcar o plugin como inicializado.

```
function configureFirstRun() {
    $this->getConfig()->set('plugin_exemplo_initialized', true);
    return true;
}
```

Atualização do Plugin

A função `configureUpgrade()` lida com a lógica de atualização do plugin. Esta função pode ser expandida para incluir scripts para por exemplo atualizar a base de dados.

```
function configureUpgrade() {
    // Lógica de atualização do plugin
}
```

Criação de Tabelas na Base de Dados

A função `createDBTables()` realiza a criação das tabelas na base de dados. Neste exemplo, é criada uma tabela chamada `example_table` com campos para `id`, `name`, `description` e `created_at`.

```
function createDBTables() {
    $sql = "CREATE TABLE IF NOT EXISTS " . EXAMPLE_TABLE . " (
        id INT(11) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(255) NOT NULL,
        description TEXT,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;"
```

```
if (db_query($sql)) {
    error_log("Tabela " . EXAMPLE_TABLE . " criada com sucesso.");
    return true;
} else {
    error_log("Falha ao criar tabela " . EXAMPLE_TABLE . ".");
    return false;
}
```

Limpeza ao Desativar o Plugin

A função `cleanUp()` lida com a remoção das tabelas e a limpeza dos dados quando o plugin é desativado. Ela executa uma consulta SQL para remover a tabela `EXAMPLE_TABLE` se ela existir.

```
function cleanUp() {
    $sql = "DROP TABLE IF EXISTS `". EXAMPLE_TABLE . "`;"

    if (db_query($sql)) {
        error_log("Tabela " . EXAMPLE_TABLE . " removida com sucesso.");
        return true;
    } else {
        error_log("Erro ao remover a tabela " . EXAMPLE_TABLE . ".");
        return false;
    }
}
```

Nota sobre a Função `pre_uninstall`

Nas versões mais antigas do osTicket, como a 1.9.3, era possível utilizar uma função chamada `pre_uninstall` para realizar ações específicas antes de desinstalar o plugin. No entanto, essa função não é mais suportada na versão 1.18 do osTicket. Portanto, caso esteja a utilizar a versão 1.18 ou superior, uma alternativa é implementar lógicas para desativar e ativar o plugin. Portanto, sempre que quiser desinstalar o plugin, é necessário desativá-lo primeiro.

Sinais

O osTicket utiliza um sistema de sinais (`signals`) para notificar os plugins sobre eventos específicos, permitindo que os programadores estendam a funcionalidade do sistema de forma modular. Este sistema permite que os plugins inscrevam-se para ouvir eventos específicos e respondam quando esses eventos ocorrem.

Funcionamento dos Sinais

A classe de roteamento de sinais do osTicket está definida em `[INSTALL_ROOT]/include/class.signal.php`. Esta classe oferece duas funções principais:

- **connect**: Permite que um plugin se inscreva para ouvir um sinal específico. Quando o sinal é emitido, todas as funções inscritas para esse sinal serão chamadas.
- **send**: Permite que um sinal seja emitido. Quando um sinal é emitido, todas as funções que se inscreveram para ouvir esse sinal serão notificadas e executadas.

A utilização dos sinais é bastante direta. Primeiro, inscreve-se para ouvir um sinal utilizando a função **connect**. Em seguida, em algum ponto do código, pode emitir esse sinal utilizando a função **send**.

Exemplo de Utilização

No exemplo fornecido do [PluginExemplo](#), o plugin conecta-se ao sinal **model.created**, que é emitido sempre que um novo ticket é criado. Neste caso, o plugin está configurado para ouvir eventos de criação de tickets e chamar a função **onTicketCreated** sempre que um novo ticket é criado.

```
Signal::connect('model.created', array($this, 'onTicketCreated'), 'Ticket');
```

A função **onTicketCreated** é então definida para realizar ações específicas quando um novo ticket é criado:

```
function onTicketCreated($ticket) {
    error_log('Novo ticket criado: ID ' . $ticket->getId());
    // Adicionar lógica adicional a ser executada quando um ticket é criado
}
```

Logs

Nesta demonstração acima, o plugin está a escrever os logs para o ficheiro de logs do Apache. Portanto, basta aceder ao ficheiro [apache/logs/error.log](#) para visualizar os mesmos.

Tabelas dos Plugins e das Suas Instâncias

No osTicket, quando um plugin é instalado, ele está representado nas tabelas **ost_plugin** e **ost_plugin_instance**. A primeira tabela, **ost_plugin**, contém os plugins instalados, enquanto a segunda tabela, **ost_plugin_instance**, contém as instâncias dos plugins. Cada instância tem um **plugin_id** que corresponde ao ID do plugin na tabela **ost_plugin**. É nestas tabelas que podemos consultar informações sobre os plugins e as suas instâncias.