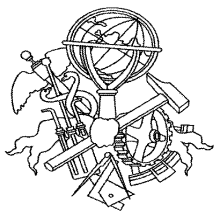
Plataforma de Gestão de Incidentes

Nuno Filipe Caldeira Leite de Faria



Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização de Telecomunicações
Departamento de Engenharia Electrotécnica
Instituto Superior de Engenharia do Porto

2009

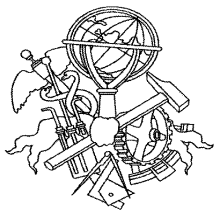
Este relatório satisfaz, os requisitos que constam da Ficha de Disciplina de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores

Candidato: Nuno Filipe Caldeira Leite de Faria, Nº 1030126, 1030126@isep.ipp.pt

Orientação científica: Paula Maria Marques Moura Gomes Viana, pmv@isep.ipp.pt

Empresa: EFACEC Sistemas de Electrónica, S.A.

Supervisão: Paulo Paixão, paulo.paixao@efacec.pt



Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização de Telecomunicações
Departamento de Engenharia Electrotécnica
Instituto Superior de Engenharia do Porto

22 de Outubro de 2009

Dedico este trabalho à Teresa.

Agradecimentos

Os meus agradecimentos são dirigidos a todos os que contribuíram para o sucesso deste projecto.

Agradeço ao Eng. Paulo Paixão, orientador na EFACEC, toda a disponibilidade demonstrada, através de explicações elucidativas acerca do que era pretendido durante todas as fases do projecto, revelando sempre muita abertura em relação a ideias e a conceitos novos.

Agradeço à Professora Doutora Paula Viana a responsabilidade científica e o apoio dado ao longo do desenvolvimento deste estudo, em especial na revisão do texto final.

Agradeço ao Mestre Rui Pedrosa pelas excelentes sugestões e opiniões para a resolução de problemas práticos de programação bem como pela camaradagem demonstrada.

Agradeço à Dr.ª Teresa Frias a forma incansável como seguiu este projecto, demonstrando sempre muito interesse e disponibilidade para discutir comigo questões sobre a conceptualização da aplicação.

Agradeço à Doutora Joana Matos Frias e à Mestre Maria José Matos Frias a colaboração nas traduções do resumo deste trabalho.

Agradeço a todas as pessoas do departamento de ID da Unidade de Transportes da EFACEC Sistemas de Electrónica, em particular ao Eng. Bruno Gil e ao Eng. Miguel Gomes, que não estando directamente relacionadas com este projecto, contribuíram de forma excepcional para a minha integração.

Por fim gostaria de agradecer às instituições ISEP e EFACEC por me terem proporcionado a oportunidade de desenvolver este projecto em âmbito empresarial.

Resumo

A EFACEC Sistemas de Electrónica desenvolve, comercializa produtos e fornece soluções globais, com uma forte componente de inovação, no âmbito das Tecnologias de Informação e Electrónica. Concretamente a EFACEC tem vindo a desenvolver ao longo dos anos soluções de apoio à exploração de redes de transportes, nomeadamente no domínio metro-ferroviário. Estas soluções estão já distribuídas por vários clientes (Metro do Porto, ReferTelecom, Metro de Tenerife, Metro de Dublin, etc.) e contemplam nomeadamente: informação ao público (sonora e visual), videovigilância, gestão de redes de transmissão. Estas soluções carecem, no entanto, de um sistema de gestão de incidentes, através do qual, em face dos eventos despoletados, o operador possa criar fichas de ocorrência. Estas fichas deverão ser automaticamente semi-preenchidas (de acordo com o contexto da ocorrência) e o seu conteúdo deverá incluir dados que o operador pretenda inserir e também imagens e extractos de vídeo relacionados. Este projecto surge da necessidade de existência de um módulo de Gestão de Ocorrências que possa ser integrado com a solução da EFACEC já existente. A aplicação tem o objectivo final de simplificar tarefas e eliminar o papel, pois até à data, nos potenciais utilizadores da aplicação, todo o trabalho de gestão de ocorrências é feito de forma manual, ou seja sem um sistema automático que permita dinamizar esta tarefa.

De forma a atingir os objectivos propostos, numa primeira fase fez-se a análise funcional de todos os requisitos da aplicação. Seguidamente foi elaborada uma especificação técnica utilizando-se para o efeito a linguagem UML o que permitiu um desenvolvimento rápido e eficaz da solução final que foi obtida tendo em conta factores como a optimização de custos no desenvolvimento de *software*, a portabilidade e suporte multi-plataforma tendo sido usado para o efeito *software open source* e tecnologias *Web* emergentes (AJAX). Por se tratar de uma aplicação *Web*, os factores usabilidade e apresentação gráfica foram alvo de uma atenção particular. As novas tecnologias de Georreferenciação e os Sistemas de Informação Geográfica constituem uma componente importante deste projecto tendo sido criado um sistema que possibilita de forma muito intuitiva associar dados a pontos geográficos, bem como gerir a informação daqueles dados de forma expedita e eficaz.

Palavras-Chave

AJAX, Georreferenciação, Java, Oracle, Sistemas de Informação Geográfica.

Abstract

EFACEC Sistemas de Electrónica (EFACEC Electronic Systems) develops and commercializes products concerning Electronic and Information Technologies, thus offering global solutions with a strong innovative feature. All over the years EFACEC has specifically developed several solutions related to the support of transportation networks, namely within the metro and railway fields. These solutions are already spread through several clients (Porto Metro, ReferTelecom, Tenerife Metro, Dublin Metro, etc.) and include: audio and video information to the public; video surveillance; transmission networks management. However, these solutions lack an Incident Management System through which, before the events, the operator would be able to generate records of occurrence. These records should be automatically semi-filled (according to the occurrence context) and its contents must include all the data the operator intends to insert, as well as related images and video extracts. This project stems from the need to create an Occurrence Management module which can be integrated in the already existing EFACEC solution. The ultimate aim of the application is to simplify tasks and eliminate paperwork, since until now all the occurrence management work has been handiwork, that is, with no automatic system permitting to fasten the job.

In order to obtain the aimed results, the first step of this research was the functional analysis of all the application requirements. Then, a technical specification was explored using UML language, which provided a rapid and effective development of the final solution. The final solution was achieved considering factors such as the optimization of expenses concerning the software development, portability and multi-platform support, for which open source software and emergent Web technologies (AJAX) were used. As it is a Web application, usability and graphic design were also taken into account with special attention. Georeferencing new technologies and Geographic Information Systems are a very important component of this research project, which involves the creation of a very intuitive system linking data to geographical points, as well as the successful and prompt management of the same data.

Keywords

AJAX, Georeferencing, Java, Oracle, Geographic Information Systems.

Résumé

EFACEC Sistemas de Electrónica (EFACEC Systèmes d'Electronique) développe, commercialise et fournit des solutions globales, ayant une forte composante innovatrice, dans le champ des Technologies de l'Information et de l'Electronique. Plus précisément EFACEC a développé au cours des ans des solutions d'appui à l'exploitation des réseaux de transports, particulièrement dans de domaine metro-ferroviaire. Ces solutions sont déjà distribuées auprès de plusieurs clients (Metro de Porto, ReferTelecom, Metro de Tenerife, Metro de Dublin, etc) et considèrent nommément : information au public (sonore et visuelle), vidéovigilance, gestion de réseaux de transmission. Cependant, ces solutions ont besoin d'un système de gestion d'incidents, à travers lequel, face aux évènements déchaînés, l'opérateur puisse créer des fiches d'occurrence. Ces fiches devront être automatiquement à demi remplies (en accord avec le contexte d'occurrence) et leur contenu devra inclure des données que l'opérateur prétende insérer bien que des images et des extraits vidéo en rapport avec l'occurrence. Ce projet surgit du besoin d'existence d'un module de Gestion d'Occurrences qui puisse être intégré à la solution EFACEC qui existe déjà. L'application a comme but final simplifier des tâches et éliminer le papier, puisque jusqu'à présent, chez les utilisateurs potentiels de l'application, tout le travail de gestion d'occurrences est accompli manuellement, c'est-à-dire sans un système automatique qui permette dynamiser cette tâche.

Pour atteindre les objectifs proposés, dans une première phase on a procédé à l'analyse fonctionnelle de toutes les conditions requises par l'application. Ensuite une spécification technique a été élaborée ; pour ça on a utilisé le langage UML ce qui a permis un développement rapide et efficace de la solution finale qui a été obtenue en considérant des facteurs comme l'optimisation des coûts dans le développement de software, la portabilité et le support multi-plateforme, en utilisant software open source et des technologies Web émergentes (AJAX) Comme il s'agissait d'une application Web, les facteurs usabilité et présentation graphique ont été objet d' une attention spéciale. Les nouvelles technologies de Georeferentiation et les Systèmes d'Information Géographique constituent une composante importante de ce projet ; on a donc créé un système qui rend possible associer

de forme très intuitive des données à des points géographiques, bien que gérer l'information de ces données de forme expéditive et efficace.

Mots-clés

AJAX, Georeferentiation, Java, Oracle, Systèmes d'Information Géographique.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

AJAX	–	Asynchronous Javascript And XML
API	–	Application Programming Interface
CGI	–	Common Gateway Interface
CORBA	–	Common Object Request Broker Architecture
CSS	–	Cascading Style Sheets
DHTML	–	Dynamic HTML
DOM	–	Document Object Model
EJB	–	Enterprise JavaBeans
HTML	–	Hypertext Markup Language
HTTP	–	Hypertext Transfer Protocol
IMAP	–	Internet Message Access Protocol
INOSS	–	Integrated Network Operations Support System
J2EE	–	Java 2 Platform, Enterprise Edition
J2SE	–	Java 2 Platform, Standard Edition
JSON	–	JavaScript Object Notation
JSTL	–	JavaServer Pages Standard Tag Library
JDBC	–	Java Database Connectivity
KML	–	Keyhole Markup Language
PDF	–	Portable Document Format
POP3	–	Post Office Protocol
RFC	–	Request for Comments
SIMS	–	Smart Incident Management System
SMTP	–	Simple Mail Transfer Protocol

UML	–	Unified Modeling Language
W3C	–	World Wide Web Consortium
XML	–	Extensible Markup Language

1 Introdução

O primeiro capítulo tem como objectivo apresentar o tema de desenvolvimento da Tese/Dissertação do Curso de Mestrado em Engenharia Electrotécnica e de Computadores, perfil de Telecomunicações, bem como o local de estágio, motivação e objectivos.

1.1 Contextualização

Este estágio foi proposto pela Unidade de Transportes da empresa EFACEC Sistemas de Electrónica S.A. do grupo EFACEC. Os seus mercados alvo estratégicos são: Metros Ligeiros, Ferrovias e Metros Pesados, Transportes Rodoviários e Aeroportos. De entre as soluções desenvolvidas e comercializadas por esta unidade, destacam-se as seguintes:

- Sistemas de Telecomunicações de Apoio à Exploração
- Sistemas de Alimentação
- Equipamentos e Sistemas para a Rede de Acesso dos Operadores
- Produção de Electrónica

Alguns dos clientes de destaque desta unidade são: ReferTelecom, Metro de Tenerife, Metro de Dublin e Metro do Porto.

A EFACEC, detém e comercializa uma aplicação específica no âmbito dos Sistemas de Telecomunicações de Apoio à Exploração denominada INOSSv2 (Integrated Network Operations Support System), que apesar de ser bastante completa para o seu âmbito de operação carece de um módulo particular – Gestão de Incidentes. Assim este estágio pretende servir para o desenvolvimento do referido módulo denominado SIMS (Smart Incident Management System).

1.2 Local de Estágio

O estágio foi realizado nas instalações da EFACEC Sistemas de Electrónica S.A. no pólo de Moreira da Maia (Zona Industrial da Maia).

1.3 Motivação

A candidatura a este estágio foi motivada pelo meu interesse particular pela programação *Web*. O facto de este estágio ter sido realizado na EFACEC e o seu âmbito principal ser o desenvolvimento de uma aplicação avançada para clientes específicos foi uma motivação adicional.

1.4 Objectivos

O objectivo principal deste estágio é o desenvolvimento de uma Plataforma (Aplicação *Web)* de Gestão de Incidentes.

Resumidamente, a aplicação a desenvolver consiste numa plataforma que permita a um utilizador, em face de determinados acontecimentos (por exemplo incidentes rodoviários, problemas em ferrovias, etc.), criar fichas de incidentes que ficam associados a pontos geográficos ou localizações pré-definidas. Estas fichas deverão ser automaticamente semi-preenchidas (de acordo com o contexto do incidente) e devem poder incluir dados que o utilizador pretenda inserir e também imagens ou vídeos relacionados. Deve também ser possível anexar notas a cada incidente. As fichas poderão ser criadas de duas formas: através de um Mapa dinâmico (pontos geográficos) ou através de um botão (em localizações pré-definidas). Adicionalmente deverá ser possível enviar *emails* com a descrição e conteúdo de determinado incidente. Deve existir a funcionalidade de exportar para ficheiros do tipo PDF (Portable Document Format) dados relevantes, tais como alterações de informação sobre determinado incidente, resultados de filtros aplicados a registos históricos, etc.

Uma das funcionalidades do INOSSv2 é a monitorização de determinados acontecimentos que ocorrem no âmbito da aplicação e que são denominados Eventos. O SIMS deve

também permitir visualizar estes mesmos eventos, bem como adicionar e gerir notas que sejam associadas aos mesmos (só para utilizadores com o perfil administrador).

Existirá uma área de administração onde será possível visualizar, alterar e remover dados que ficam associados a cada utilizador da aplicação.

A aplicação tem o objectivo final de simplificar tarefas e eliminar o papel, pois até à data, nos potenciais utilizadores da aplicação, todo o trabalho de gestão de ocorrências é feito de forma manual, ou seja sem um sistema automático que permita dinamizar esta tarefa.

2 Arquitectura global do Sistema e Especificação de requisitos

A aplicação desenvolvida funciona como um módulo da aplicação INOSSv2 - Assim, neste capítulo faz-se um resumo da aplicação INOSSv2 e explica-se a relação da aplicação criada (SIMS) com o INOSSv2.

2.1 O INOSSv2 – Integrated Network Operations Support System

O INOSSv2 é uma poderosa plataforma de operação e gestão centralizada de redes e serviços de telecomunicações. A sua arquitectura modular permite que lhe sejam adicionados módulos de operação de Transmissão, de Videovigilância, de Informação ao Público ou outros de acordo com o cenário de aplicação e atendendo às necessidades específicas de cada cliente. Através de uma interface gráfica baseada em Java, os operadores têm acesso a todas as funcionalidades necessárias à operação do sistema global.

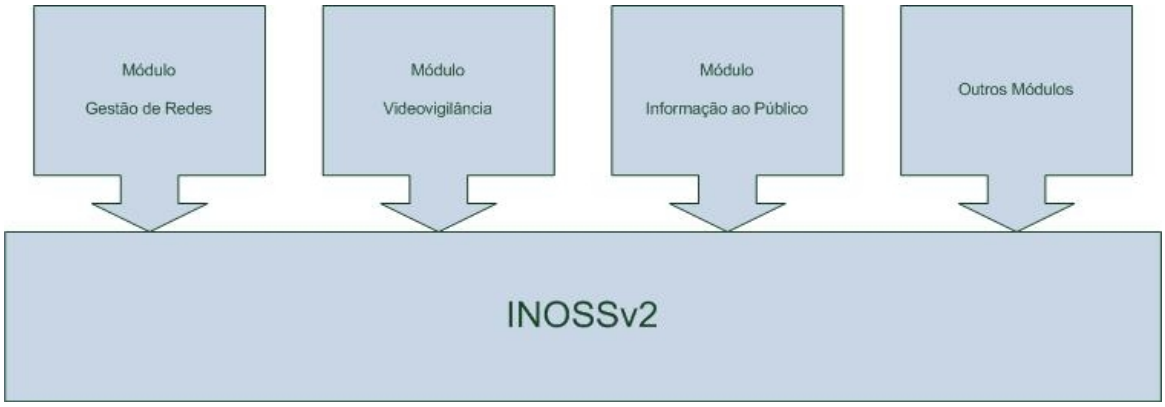


Figura 1 O INOSSv2

As funcionalidades globais desta aplicação são as seguintes:

- Plataforma de Gestão de Redes e de Telecomunicações
- Integração de módulos independentes de Transmissão, Videovigilância, Informação ao Público, etc..
- Possibilidade de redundância geográfica
- Processamento automático de alarmes em tempo real
- Edição independente das topologias de rede dos diferentes módulos
- Definição de níveis de acesso ao sistema e sua atribuição por operador e por aplicação
- Registo automático de acessos, operações e alarmes, por aplicação

2.1.1 Módulo Gestão de Redes

O módulo denominado "Gestão de Redes" permite a operação e gestão de redes e equipamentos de telecomunicações usando o protocolo SNMP (Simple Network Management Protocol).

As funcionalidades deste módulo são as seguintes:

- Operação e gestão centralizada com visualização do estado da rede
- Actualização automática da topologia activa da rede
- Suporte de estruturas de rede hierárquicas com vários níveis
- Disponibilização de alarmes em tempo real
- Integração de equipamentos de outros fabricantes
- Representação gráfica normalizada do nível de gravidade de alarmes

2.1.2 Módulo Videovigilância

Este módulo permite a captura, processamento, visualização e armazenamento de imagens bem como a operação dos equipamentos de imagem (câmaras e monitores), gravação (videogravadores analógicos e digitais) e sensores de actuação.

O módulo em questão tem as seguintes funcionalidades:

- Monitorização visual de espaços públicos ou privados
- Operação e gestão de equipamentos
- Representação gráfica dos alarmes
- Disponibilização de alarmes em tempo real
- Acções sobre o sistema despoletadas manualmente, automaticamente ou por evento

2.1.3 Módulo Informação ao Público

O módulo "Informação ao Público" permite a operação e gestão de sistemas de informação visual e sonora. Permite também actuar sobre equipamentos de teleindicação (painéis e monitores) e de sonorização (amplificadores e altifalantes), fornecendo aos utentes informação de carácter geral ou específico.

As funcionalidades deste módulo são as que se seguem:

- Difusão de informação visual e sonora
- Operação e gestão de equipamentos
- Representação gráfica de alarmes
- Disponibilização de alarmes em tempo real
- Integração com sistemas externos de seguimento de composições e itinerários

2.1.4 Âmbito de aplicação do INOSSv2

O INOSSv2 é vocacionado para clientes que necessitem de integrar sistemas de telecomunicações com outros sistemas (por exemplo transportes).

O INOSSv2 é habitualmente usado por grandes empresas do sector dos transportes. Alguns casos de sucesso são:

- REFER (Portugal)
- Metro de Tenerife (Tenerife)
- Metro de Dublin (Dublin)

2.2 O SIMS – Smart Incident Management System

A aplicação desenvolvida foi intitulada SIMS – Smart Incident Management System e é um sistema de informação destinado à simplificação de tarefas no contexto da gestão de incidentes, sendo vocacionado para empresas da área dos transportes rodoviários, ferroviários e metro-ferroviários.

Por ter sido construído como um módulo do INOSS, é muito fácil integrar o SIMS com o INOSS e instala-lo em clientes que utilizem o INOSS.

2.2.1 Análise / Levantamento de Requisitos

A aplicação tem dois níveis de acesso (Administrador e Operador) e é composta por duas áreas, a área funcional (acessível por todos os perfis) e a área de administração (acessível apenas por administradores).

No caso de o utilizador ser Administrador tem acesso às seguintes funcionalidades:

► Área Funcional:

- Criar Incidente (a partir de Botão): neste caso as localizações dos incidentes são conhecidas, isto é, estão guardadas em base de dados.
- Alterar estado dos incidentes: o estado de cada incidente varia no tempo.

○ Estados possíveis: 0- Iniciado, 1- Em execução, 2- Resolvido

● Consultar Registo Histórico de Incidentes: o utilizador pode consultar os incidentes criados ao longo do tempo, tendo disponíveis filtros e ordenações que lhe permitirão uma consulta mais eficiente.

● Filtrar o Registo Histórico de Incidentes e Exportar para PDF o resultado da aplicação dos filtros: existe a possibilidade de filtrar o Registo Histórico de Incidentes por causa de Nota associada ao incidente, utilizador que criou a Nota, data de criação da Nota e texto da Nota associada ao incidente.

● Consultar Registo das alterações dos Incidentes: sempre que se faz uma alteração num incidente, esta deverá ser guardada, permitindo ao utilizador consultar todas as alterações efectuadas.

● Exportar para PDF o Registo de alteração de Incidente: sempre que existe uma alteração nos dados de um incidente, estes ficam guardados em Histórico. Assim existe a possibilidade de exportar para PDF estes dados.

● Gerir Nota associada a Incidente: cada incidente poderá ter uma Nota associada. Esta Nota é caracterizada por uma data (de criação), uma causa, um utilizador, e poderá ter uma ou mais observações. As operações possíveis sobre a Nota são:

○ Criar Nota: associar uma Nota a um Incidente.

○ Visualizar Nota.

○ Editar Nota: adicionar observações à Nota.

○ Apagar Nota

● Anexar ficheiros a Incidente: o utilizador pode associar a um incidente ficheiros com os formatos: .pdf, .txt, .wmv, .mpg, .jpeg, .gif. Estes ficheiros podem ser visualizados quando se consulta o conteúdo de um incidente.

● Mapa: ao utilizador é disponibilizado um Mapa dinâmico que terá as seguintes funcionalidades:

○ Criar incidente a partir de marcador: o utilizador marca uma localização no Mapa a partir da qual pode criar um incidente ficando registado no mesmo o ponto geográfico (latitude, longitude e endereço).

○ Menu de zoom e navegação interno: possibilidade de fazer zoom sobre áreas do Mapa e navegar em todas as direcções.

○ *Reset* do mapa para uma região pré-definida por utilizador: está associada a cada utilizador um ponto geográfico que é usado para inicializar o Mapa, centrando-o neste ponto quando o utilizador entra na aplicação ou a seu pedido.

● Tabela associada ao Mapa com a descrição dos 10 últimos incidentes criados: ao lado do Mapa estará disponível uma tabela com os dez últimos incidentes a partir dos quais é possível, centrar o Mapa na localização que lhe está associada.

● Envio de *emails*: a aplicação permite que, a qualquer altura, o utilizador possa pedir o envio de um *email* a partir de um incidente, com a descrição do mesmo e para destinatários pré-definidos (parametrizados no utilizador).

● Consultar Registo Histórico de Eventos: o INOSSv2 gera eventos (registos automáticos de várias operações realizadas pelo sistema) através do CORBA. Estes eventos ficam automaticamente guardados em base de dados. Deverá ser possível visualizar de forma assíncrona todos os eventos gerados, tendo o utilizador disponíveis filtros e ordenações que lhe permitirão uma consulta mais eficiente.

● Filtrar o Registo Histórico de Eventos e Exportar para PDF o resultado da aplicação dos filtros: existe a possibilidade de filtrar o Registo Histórico de Eventos por causa de Nota associada ao Evento, utilizador que criou a Nota, data de criação da Nota e texto da Nota associada ao Evento.

● Gerir Nota associada a Evento: da mesma forma que é possível associar Notas a Incidentes, também é possível associar Notas a Eventos. As operações disponíveis são idênticas:

○ Criar Nota

○ Visualizar Nota

○ Editar Nota

○ Apagar Nota

► Área de Administração:

● Criar dados associados a utilizador: é possível associar os seguintes dados a um utilizador, sendo estes usados nas funcionalidades descritas anteriormente:

○ Região de inicialização do mapa

○ Endereços de *email* para os quais podem ser enviados alertas

● Editar dados associados a utilizador: é possível editar os dados previamente criados

● Eliminar dados associados a utilizador: é possível eliminar os dados previamente criados

No caso de o utilizador ser operador tem acesso a todas as funcionalidades de um administrador excepto: Área de Administração, Consultar Registo Histórico de Eventos (na Área Funcional) e Gerir Nota associada a Evento (na Área Funcional).

A tabela 1 permite uma melhor compreensão dos requisitos da aplicação pois permite fazer a comparação entre os dois perfis (Administrador e Operador).

Em face destes requisitos fez-se um estudo/levantamento das tecnologias mais avançadas que existem actualmente no mercado sempre com a preocupação de usar quando possível *software open source* de forma a minimizar custos. Este estudo do estado da arte foi compilado e será detalhado no capítulo seguinte.

Área Funcional		**Administrador**	**Operador**
Criar Incidente		S	S
Alterar estado dos Incidentes		S	S
Consultar Registo Histórico de Incidentes		S	S
Filtrar o Registo Histórico de Incidentes. Exportar para PDF o resultado da aplicação dos filtros		S	S
Consultar Registo das alterações dos Incidentes		S	S
Exportar para PDF o Registo de alteração de Incidente		S	S
Gerir Nota associada a Incidente	Criar Nota	S	S
	Visualizar Nota	S	S
	Editar Nota	S	S
	Apagar Nota	S	S
Anexar Ficheiros a Incidente		S	S
Mapa	Criar Incidente a partir de marcador	S	S
	Menu de zoom e navegação interno	S	S
	Reset do mapa para uma região pré-definida por utilizador	S	S
Envio de *emails*		S	S
Consultar Registo Histórico de Eventos		S	N
Filtrar o Registo Histórico de Eventos. Exportar para PDF o resultado da aplicação dos filtros		S	N
Gerir Nota associada a Evento	Criar Nota	S	N
	Visualizar Nota	S	N
	Editar Nota	S	N
	Apagar Nota	S	N
Área de Administração		**Administrador**	**Operador**
Criar dados associados a utilizador	Criar ponto de inicialização do mapa	S	N
	Criar endereços de *email* para os quais podem ser enviados alertas	S	N
Editar dados associados a utilizador	Editar ponto de inicialização do mapa	S	N
	Editar endereços de *email* para os quais podem ser enviados alertas	S	N
Eliminar dados associados a utilizador	Eliminar ponto de inicialização do mapa	S	N
	Eliminar endereços de *email* para os quais podem ser enviados alertas	S	N

Tabela 1 Requisitos da aplicação

3 Tecnologias

Ao longo deste capítulo serão apresentadas as tecnologias utilizadas na implementação da aplicação, finalizando com uma breve descrição das tecnologias que se consideraram concorrentes para a realização de projectos deste âmbito.

3.1 Tecnologias utilizadas

3.1.1 J2EE - Java 2 Platform, Enterprise Edition

A plataforma J2EE permite desenvolver aplicações empresariais multi-camada. A lógica de aplicação é dividida em componentes de acordo com a sua função, e os vários componentes que constituem uma aplicação J2EE podem estar instalados em diferentes máquinas dependendo da camada a que pertencem [JET].

A fig.2 mostra duas aplicações J2EE multi-camada divididas nas camadas seguintes:

- Os componentes da camada cliente são executados na máquina cliente
- Os componentes da camada *Web* são executados no servidor J2EE
- Os componentes da camada de negócio são executados no servidor J2EE
- O *software* da camada EIS – Enterprise Information Systems é executado no servidor EIS

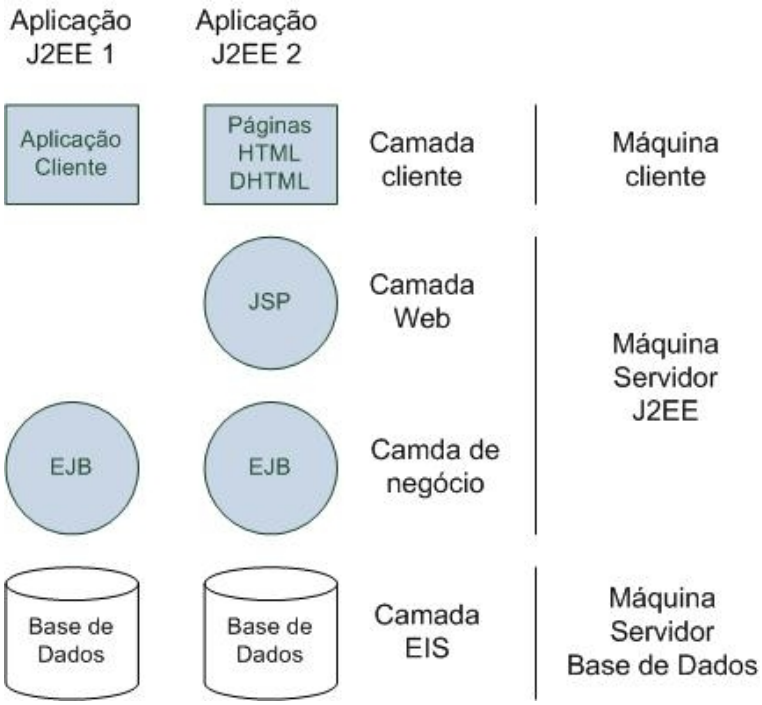


Figura 2 Exemplo de aplicações J2EE

Apesar de uma aplicação J2EE habitualmente consistir em três ou quatro camadas como mostrado na fig.2, as aplicações multi-camada são geralmente consideradas como sendo aplicações de três camadas porque estão distribuídas por três localizações distintas: as máquinas clientes, o servidor J2EE e a máquina que tem o servidor de base de dados.

As aplicações J2EE são constituídas por componentes. Um componente J2EE é uma unidade funcional de *software* que é integrada numa aplicação J2EE com as suas classes e que comunica com outros componentes. A especificação J2EE define os seguintes componentes:

- Clientes de aplicações e Applets (executados no cliente)
- As tecnologias Java Servlet e JavaServer Pages (executadas no servidor)
- Enterprise JavaBeans (executados no servidor)

Esta plataforma simplifica o desenvolvimento de aplicações empresariais porque as baseia em componentes modulares normalizados, fornecendo um conjunto completo de serviços

para esses componentes e manipulando detalhes das aplicações de forma automática sem a necessidade de programação complexa. A plataforma J2EE tem a vantagem dos vários recursos da plataforma J2SE (Java 2 Platform, Standard Edition), tais como a portabilidade, uma API (Application Programming Interface) para acesso a bases de dados, a tecnologia CORBA (Common Object Request Broker Architecture) para interacção com recursos empresariais já existentes, e um modelo de segurança para protecção de dados. A plataforma J2EE fornece total suporte para o uso de EJB (Enterprise JavaBeans), Java Servlets, Java Server Pages e a tecnologia XML (Extensible Markup Language). Adicionalmente a especificação J2EE assegura interoperabilidade com Web Services [JET] [PER04] [WEA04].

A J2SE fornece a *framework* essencial da linguagem sobre a qual o J2EE é construído. É portanto o núcleo da J2EE.

Os serviços standard J2EE são os seguintes:

- HyperText Transfer Protocol/HyperText Transfer Protocol Secure sockets (HTTP/HTTPS)
- Java Transaction API (JTA)
- Remote Method Invocation to Internet Inter-ORB Protocol (RMI-IIOP)
- Java Database Connectivity (JDBC)
- Java Message Service (JMS) 1.1
- JavaMail
- Java Naming and Directory Interface (JNDI)
- JavaBeans Activation Framework (JAF)
- Java API for XML Parsing (JAXP)
- J2EE Connector Architecture
- Security Services
- Web Services

- Management
- Deployment
- Java Authorization Service Provider Contract for Containers (JACC)

3.1.1.1 Java Servlet

As Java Servlets são uma solução eficiente e poderosa para criar conteúdo dinâmico para a *Web*. Uma Servlet é uma extensão genérica do servidor *Web*, ou seja é uma classe Java que pode ser carregada dinamicamente para expandir as funcionalidades de um servidor. As Servlets são habitualmente usadas com servidores *Web* e substituem por completo o tradicional mecanismo CGI (Common Gateway Interface). Como as Servlets operam apenas no domínio do servidor, não requerem suporte para Java no cliente *Web*. Ao contrário das aplicações que utilizam o mecanismo CGI e por isso usam múltiplos processos para gerir programas separados e/ou pedidos separados, as Servlets são geridas por *threads* separados dentro do servidor *Web*. Isto significa que as Servlets são muito eficientes quando comparadas com o tradicional mecanismo de CGI.

A API Java Servlet fornece um interface standard que permite a programadores lidar com pedidos de aplicações clientes e gerar respostas apropriadas a esses pedidos. Assim, esta API fornece um bloco fundamental para o desenvolvimento de aplicações do lado do servidor. A API Java Servlet define uma plataforma genérica de pedidos e respostas para gerir pedidos HTTP e gerar respostas HTTP através da *Web* [FAL03].

A tecnologia Java Servlet tem acesso a toda a família de APIs de Java. Como é uma expansão da tecnologia Java, herda todas as suas características de base, tais como reutilização e portabilidade do código. Desta forma é assegurada a independência da plataforma física, do sistema operativo e do servidor *Web* adoptado. Esta tecnologia apresenta, no entanto, uma desvantagem na medida em que não separa, do lado do servidor, a geração da parte da apresentação da parte que implementa a lógica do negócio. Este aspecto pode, contudo, ser ultrapassado através do uso conjugado com outras tecnologias (e.g.: JavaServer Pages) [MAL02].

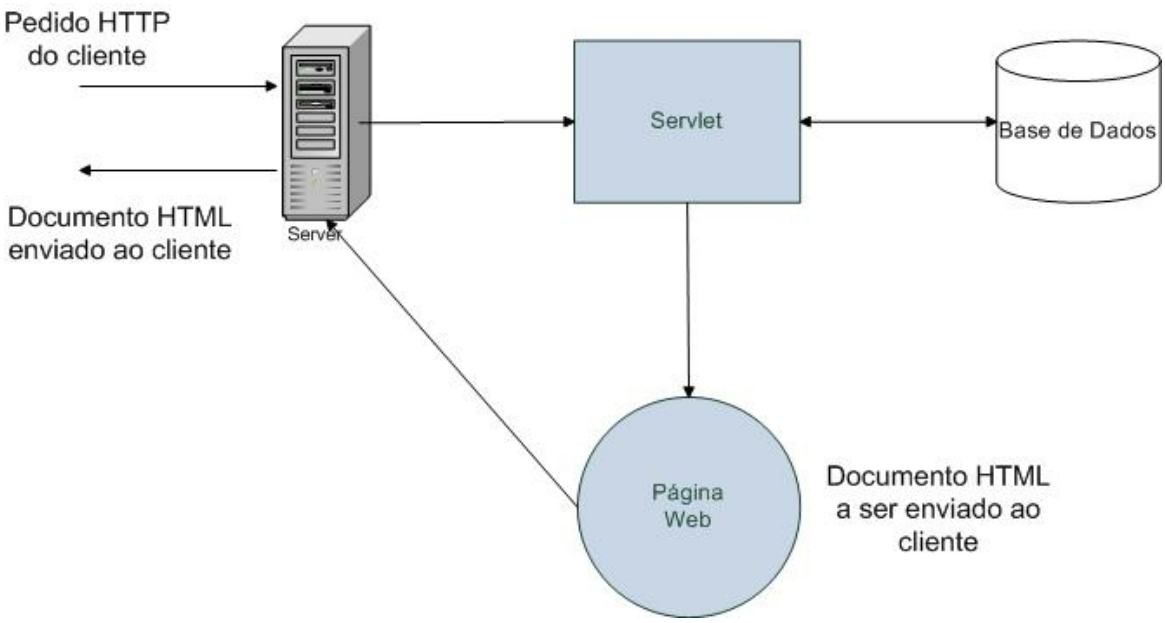


Figura 3 Mecanismo de funcionamento de uma Servlet

As Servlets usam classes e interfaces de dois *packages*: javax.servlet e javax.servlet.http. O *package* javax.servlet contem as classes que suportam Servlets que são genéricas e independentes do protocolo. Estas classes são estendidas pelas classes no *package* javax.servlet.http para terem funcionalidades específicas do protocolo HTTP. Todas as Servlets têm de implementar o *interface* javax.servlet.Servlet. A maioria das Servlets implementam-no estendendo uma das duas classes especiais: javax.servlet.GenericServlet ou javax.servlet.http.HttpServlet. Uma Servlet independente do protocolo é uma sub-classe da GenericServlet enquanto que uma servlet HTTP Servlet é uma sub-classe da HttpServlet que é por sua vez uma sub-classe da GenericServlet com a funcionalidade HTTP extra. Ao contrário de um programa Java normal, e tal como um applet, uma Servlet não tem um método main. Em vez disso, certos métodos de uma Servlet são invocados pelo servidor para processar o tratamento dos pedidos. Uma Servlet genérica deve sobrepor este método para tratar os pedidos de forma adequada. O método `service` aceita dois parâmetros: Um Objecto pedido e um Objecto resposta. O Objecto pedido refere-se ao pedido em questão e o Objecto resposta é usado para retornar uma resposta [HAL01] [HAL03].

3.1.1.2 JavaServer Pages

As JSP (JavaServer Pages) são componentes numa aplicação *Web* ou numa aplicação J2EE, compostas por uma parte estática em HTML (HyperText Markup Language) que define o aspecto da apresentação de interface com o utilizador e outra parte dinâmica que encapsula a lógica do negócio (Java) [HAL01] [HAL03].

Como componentes de uma aplicação J2EE, as JSP são executadas num servidor e respondem a pedidos de clientes. O protocolo usado pelos clientes para invocar páginas HTML e JSP numa aplicação J2EE é o HTTP (Hypertext Transfer Protocol).

Quando um cliente envia um pedido a uma JSP, o servidor *Web* entrega o pedido ao contentor de JSPs, e este determina qual a classe de implementação da JSP em questão que deve lidar com o pedido. Da invocação de uma JSP resulta a criação automática de uma Servlet que irá proceder a todas as operações do lado do servidor necessárias ao preenchimento da parte dinâmica do documento.

Para criar uma página JSP que possa responder a pedidos de clientes são necessários três processos: primeiro é necessário escrever a JSP, depois é necessário fazer o *deployment* para o servidor e por último, num determinado instante, a página é traduzida e compilada numa classe Java (Servlet). Isto pode acontecer antes de a página ser carregada no servidor, ou pode acontecer no instante em que o cliente faz o pedido. Uma JSP serve pedidos como uma Servlet, assim o ciclo de vida é determinado pela tecnologia Java Servlet.

Quando um pedido é mapeado numa página JSP, ele é manipulado por uma Servlet especial que primeiro determina se a Servlet a ser gerada pela JSP é mais antiga ou não do que a JSP em questão. Se for, dá-se a tradução da JSP para uma Servlet e a classe é compilada.

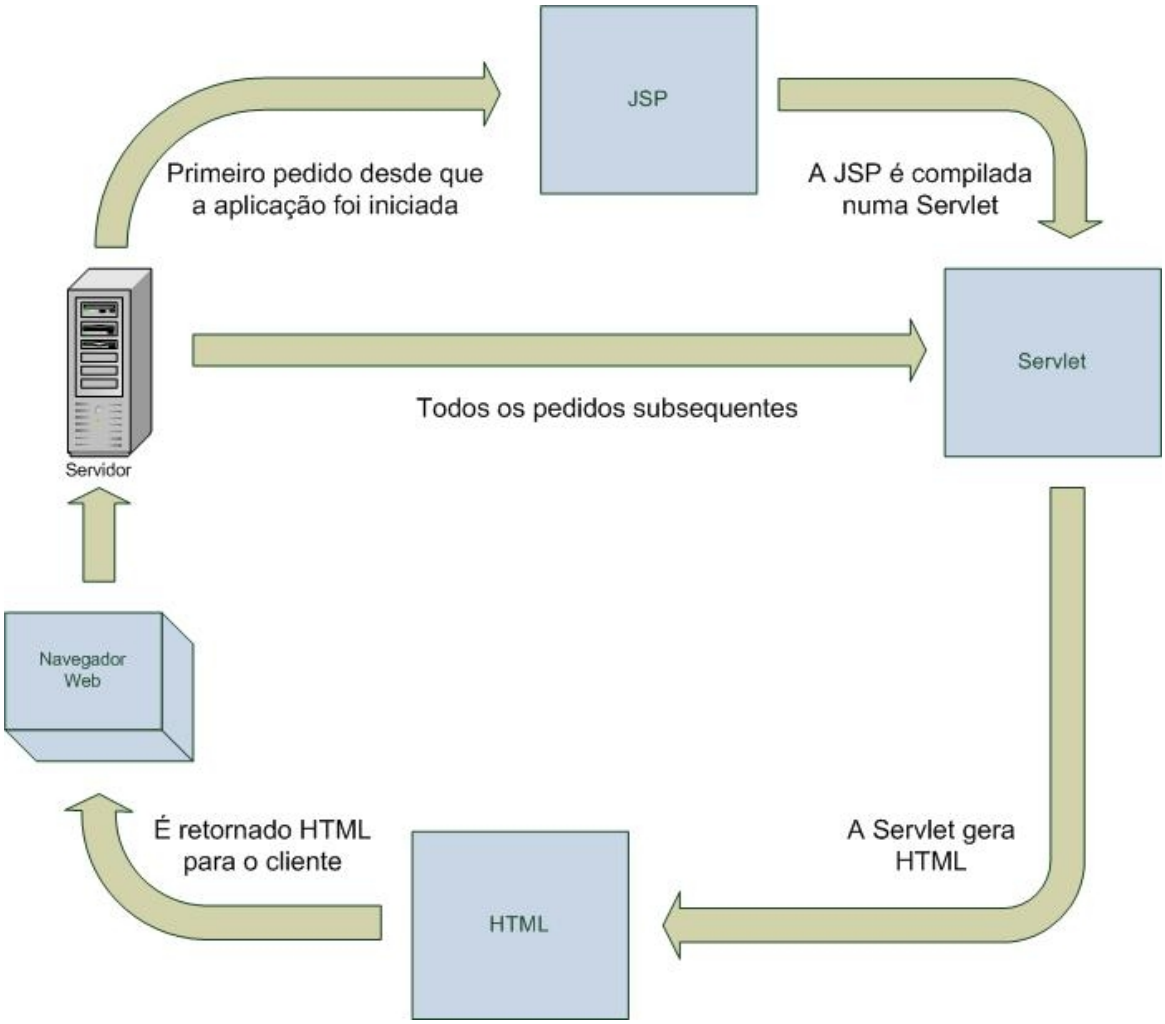


Figura 4 **Mecanismo de funcionamento de uma JSP**

Com o uso de JSPs é possível implementar a camada de apresentação e a camada de lógica de negócio de forma separada, deixando aos designers apenas a preocupação de interface gráfico através do uso de *tags* com o formato HTML e XML e de *scriplets* escritos em linguagem Java. Assim sendo, a lógica do negócio é suportada pelos componentes residentes do lado do servidor (por exemplo, JavaBeans ou Enterprise JavaBeans) a que o documento JSP tem acesso através de *tags* com formatação XML e dos *scriplets* referidos [MAL02] [MAL04].

Esta utilidade é sintetizada no *site* da Sun da seguinte forma: "*Web developers and designers use JavaServer Pages technology to rapidly develop and easily maintain information-rich, dynamic web pages that leverage existing business systems*."

Como já foi referido anteriormente, o objectivo da tecnologia JSP é produzir conteúdo dinâmico para a *Web*. Esta capacidade é implementada ao embeber lógica programática com dados *template* que em conjunto produzem o conteúdo em questão. A lógica programática pode ser classificada nos seguintes elementos [HAL01] [HAL03]:

- Elementos de *Scripting*
  - Comentários
  - Declarações
  - *Scriplets*
  - Expressões
  - Expressões EL (*Expression language*)
- Directivas
  - *page directives*
  - *include directives*
  - *taglib directives*
- Elementos de acção (*Action Elements*)
  - *Standard actions*
  - *Custom actions*
  - *JSTL (JavaServer Pages Standard Tag Library) actions*

3.1.1.3 Java Beans

A implementação da lógica do negócio reside do lado do servidor e deve ser baseada na definição de componentes reutilizáveis (JavaBeans ou EJB).

Os JavaBeans e os EJB são duas entidades distintas, mas devido às suas semelhanças (e por razões de marketing) partilham o mesmo nome. Os JavaBeans são componentes construídos em Java que podem ser usados em qualquer camada da aplicação. Por sua vez os EJB são componentes especiais baseados no servidor e são usados para construir a lógica do negócio e a funcionalidade de acesso aos dados numa aplicação.

Um JavaBean é apenas uma classe Java. Não necessita de estender qualquer classe base ou implementar qualquer interface. No entanto para ser um JavaBean, a classe deve obedecer a um conjunto de regras definidas pela especificação JavaBeans:

- A classe deve ter um construtor sem argumentos;

- Um *Bean* pode ter um método público para ser usado como *setter* de determinada propriedade. Este método tem a seguinte assinatura:

```
public void setPropertyName(PropertyType value);
```

- Um *Bean* pode ter um método público para ser usado como *getter* de determinada propriedade. Este método tem a seguinte assinatura:

```
public PropertyType getPropertyName();
```

Os métodos que um JavaBean exporta são os métodos de Java que podem ser invocados por outros componentes ou a partir de um ambiente de *scripting*. Os eventos constituem uma forma de notificação de ocorrências relevantes entre componentes. Quando a fonte de um evento detecta alguma ocorrência importante invoca o método de escuta apropriado do objecto destino.

3.1.1.4 JavaMail API

A API JavaMail é constituída por um conjunto de APIs abstractas que modelam um sistema de *email*. Esta API fornece uma *framework* independente da plataforma e independente do protocolo para construir aplicações clientes de *email* baseados na tecnologia Java. A API JavaMail modela um sistema de *email* de uma forma abstracta. As

mensagens, o envio de mensagens e a recepção de mensagens são representados por objectos genéricos. Devido a esta abstracção a API não restringe o programador a protocolos específicos de *email*. Por exemplo, o objecto usado para receber um *email* pode ser usado para receber um *email* em todos os diferentes protocolos.

A API JavaMail é uma representação de alto nível dos componentes básicos de um sistema de *email*. Estes componentes são representados por classes abstractas no *package* javax.mail. Por exemplo, a classe abstracta javax.mail.Message representa uma mensagem de *email*. Esta classe declara métodos abstractos responsáveis pelo *set* e *get* dos vários tipos de campos de informação que constitui uma mensagem de *email*. A classe abstracta javax.mail.Folder representa a pasta de mensagens e declara métodos abstractos para a recepção de mensagens numa pasta, para mover mensagens entre pastas e para apagar mensagens de uma pasta. Estas classes são todas abstractas porque não fazem assumpções sobre como um *email* é guardado ou transferido entre máquinas. Por exemplo, não assumem que as mensagens estão a ser enviadas usando SMTP (Simple Mail Transfer Protocol) ou que têm a estrutura especificada por exemplo pelo RFC 822. Subclasses especiais destas classes especializam as classes abstractas para protocolos e formatos de *email* específicos [HAR04].

A API JavaMail usa o padrão abstract factory design. Este padrão permite escrever código baseado em super classes abstractas sem a preocupação dos detalhes de baixo nível. O protocolo e formatos utilizados e associados à implementação concreta das classes são determinados em grande parte pelas primeiras linhas de código onde se define qual o protocolo a usar. Mudar o nome do protocolo é quase 90% do trabalho que se tem para mudar uma classe quando se pretende alterar o protocolo a usar (ex:.mudar de POP (Post Office Protocol) para IMAP (Internet Message Access Protocol)) [idem].

Os fornecedores de serviço implementam protocolos particulares. Um fornecedor de serviço é um grupo concreto de subclasses das classes abstractas pertencentes à JavaMail API que especializam a API genérica para um protocolo particular e formato de *email* específico. Estas subclasses estão provavelmente organizadas num único *package*. Algumas destas (IMAP, SMTP) são fornecidas pela Sun com a sua referência de implementação. Outras estão disponíveis através de terceiros. O objectivo da API Java Mail é esconder do programador estes detalhes de baixo nível. Assim, os programas são

escritos não para aceder a um servidor IMAP por exemplo, mas para apenas comunicarem com a API JavaMail [idem].

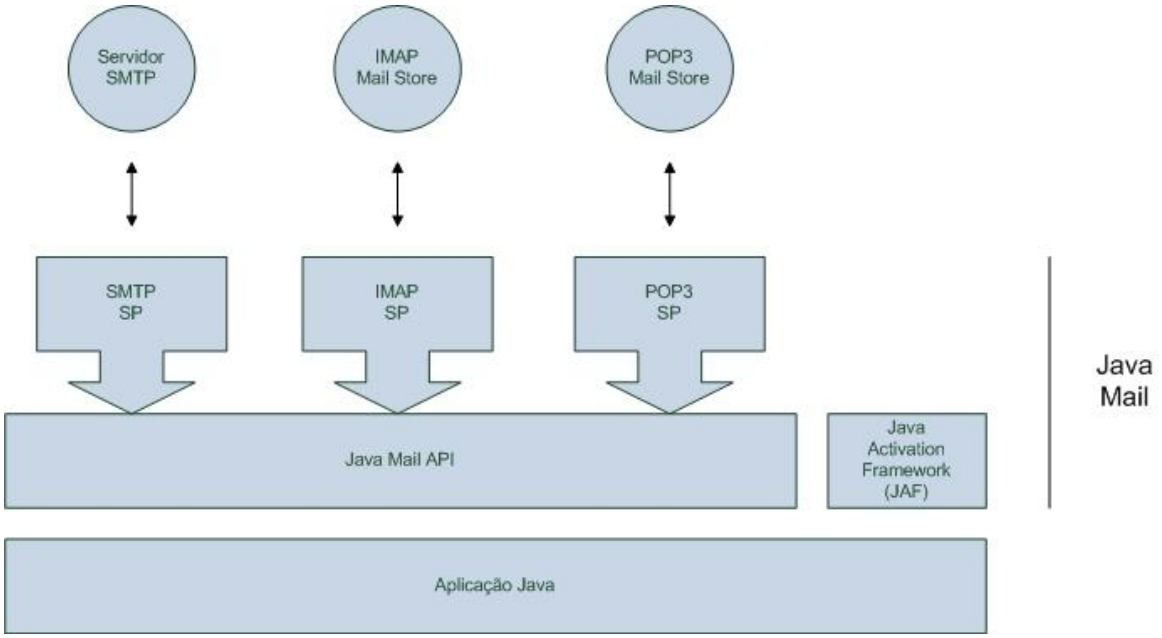


Figura 5 A API *JavaMail*

3.1.2 AJAX

O AJAX (Asynchronous Javascript And XML) tem tido um crescimento exponencial nos últimos anos. A história do seu nascimento está associada ao ActiveX control chamado "XMLHttpRequest" na aplicação Outlook da Microsoft, mas no entanto só em Fevereiro de 2005 é que a tecnologia foi baptizada com o termo AJAX por Jesse James Garret no artigo "Ajax: A New Approach to Web Applications" [GAR], altura a partir da qual houve uma explosão de interesse por esta tecnologia. O recente interesse em *mash-ups* (mistura de conteúdos provenientes de vários *websites* numa só página) tem uma afinidade especial com o AJAX. O AJAX é uma tecnologia disruptiva, isto é, surgiu e modificou por completo a forma habitual como as aplicações são construídas.

A figura seguinte mostra o *workflow* de uma aplicação "pré-AJAX". Este *workflow* segue um padrão do tipo *work-wait*. Isto é, em qualquer instante temporal, o cliente está a apresentar informação ou está à espera que o servidor retorne uma resposta. Do ponto de vista do utilizador, a experiência é muito pontuada. Durante os períodos de espera, o utilizador não pode interagir com a aplicação *Web*, para além de apenas poder ler informação que está prestes a ser substituída pela resposta do servidor. Da perspectiva da usabilidade é extremamente problemático. Cada período de espera é uma interrupção pouco apreciada por qualquer utilizador. E no entanto os períodos de espera são muito frequentes. A maior parte das aplicações *Web* requer contacto frequente com o servidor. Este modelo é claramente despropositado para qualquer tipo de actividade que implique muito processamento [GAR].

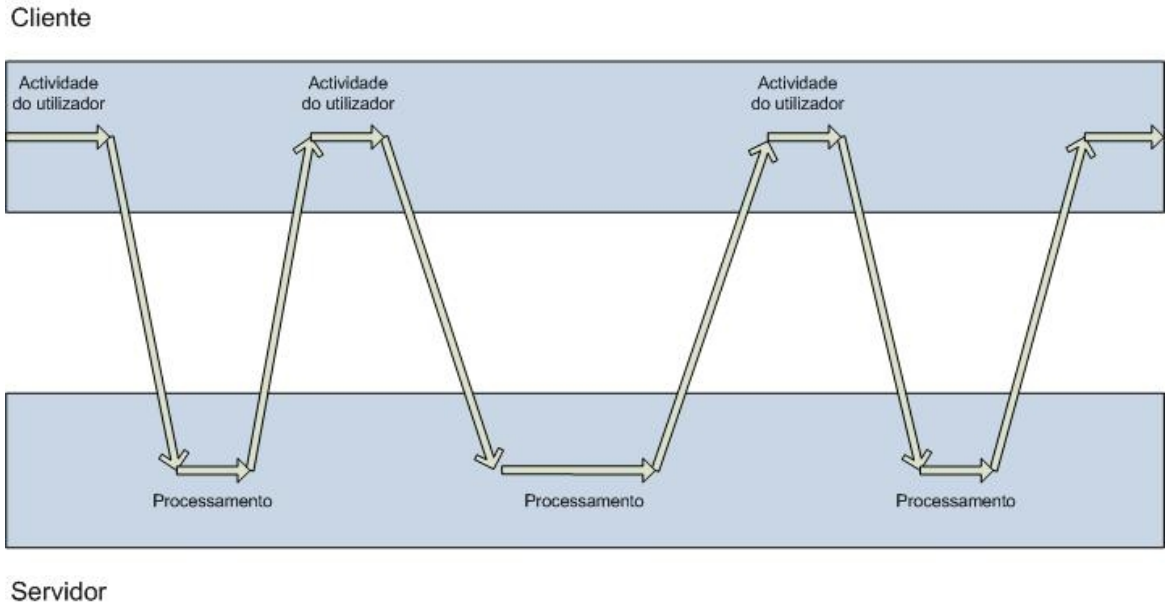


Figura 6 ***Workflow* de uma aplicação pré-AJAX**

Em contraste a figura seguinte mostra o *workflow* de uma aplicação AJAX.

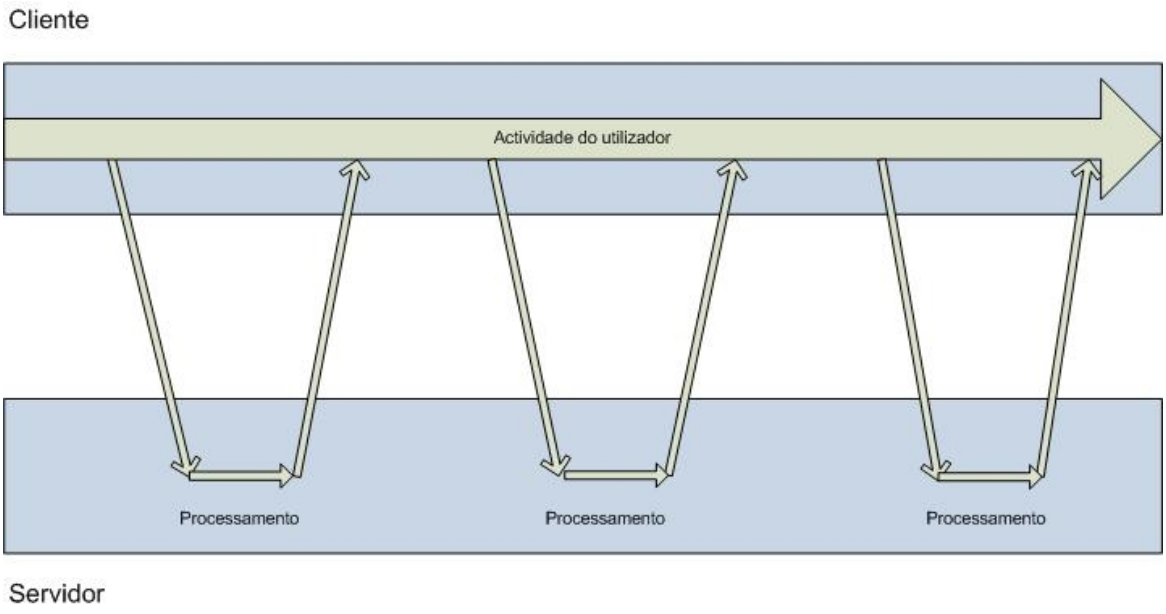
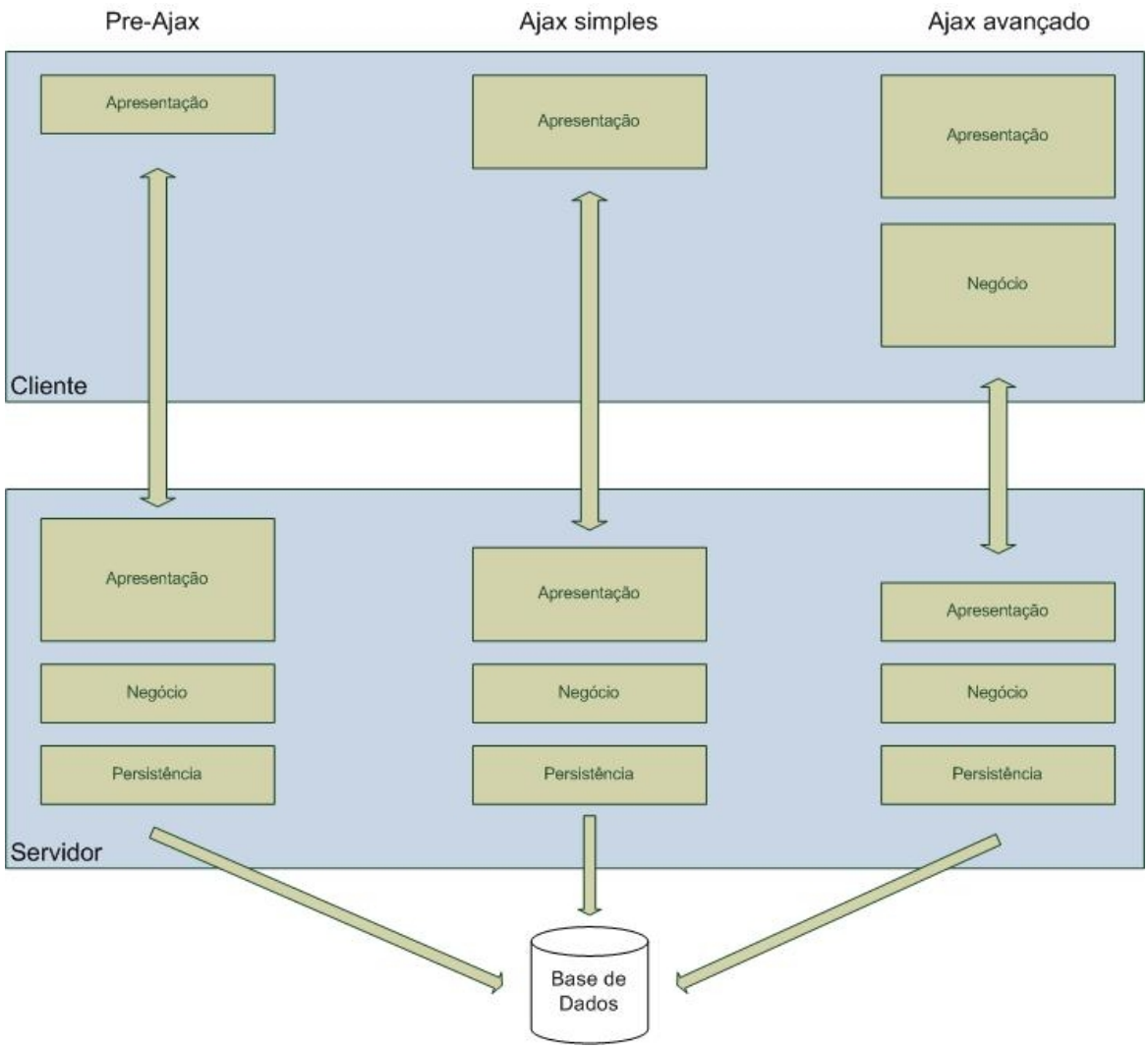


Figura 7 ***Workflow*** **de uma aplicação AJAX**

A diferença existente entra as fig. 6 e 7 mostra claramente que uma aplicação construída usando AJAX se torna muito mais eficiente em termos de usabilidade, de facto os tempos de espera entre acções podem ser reduzidos a um mínimo praticamente imperceptível por parte do utilizador.

O facto de existir processamento no lado do servidor não inviabiliza que o utilizador possa continuar a agir com a aplicação. Torna-se portanto uma experiência muito mais "amigável".

A figura seguinte ilustra a evolução da arquitectura de aplicações *Web*:



-

Figura 8 **Evolução da arquitectura de aplicações *Web***

Na arquitectura "pré-AJAX" (coluna da esquerda), toda a acção ocorre no servidor, e o *browser* age apenas como cliente passivo aceitando o conteúdo HTML. Na coluna do meio está ilustrado o impacto de introduzir algum AJAX relativamente simples na aplicação. Digamos que o servidor ainda controla todos os aspectos do *workflow*, mas os *hyperlinks* e os formulários agora fazem pedidos de conteúdo HTML que são usados para actualizar partes do ecrã em vez de fazer um refrescamento completo. As respostas do servidor são mapeadas por código JavaScript que lê a resposta e rearranja o DOM (Document Object Model). A camada de apresentação começa a ficar cada vez mais fina, à medida que o

JavaScript adicionado serve para encaminhar o conteúdo recebido do servidor gerando cada vez mais respostas focadas em vez de construir a página toda de cada vez que um pedido ao servidor é feito. Uma vez que uma aplicação bem construída tenderá a gerar respostas de uma forma modular, o facto de se introduzir AJAX não vai implicar uma reescrita completa da aplicação. No entanto, o padrão de pedidos e respostas ao servidor provavelmente irá mudar [CRD07].

À medida que avançamos para um novo conjunto de aplicações que o AJAX potenciou aumenta a necessidade de mudar as camadas da arquitectura. No caso do uso extremo de AJAX (coluna da direita), o código JavaScript é suficientemente complexo para ser dividido ele próprio em camadas. Neste caso, a camada de apresentação do lado do cliente tem controlo sobre o *workflow* do utilizador. O código do lado do cliente também mantém o modelo parcial da maioria das entidades de domínio, e a camada de apresentação do JavaScript tenderá a comunicar com estas entidades em vez de comunicar directamente com o servidor. Do lado do servidor, podemos constatar que a camada de apresentação é cada vez mais reduzida. As suas responsabilidades são fornecer mecanismos para lidar com os principais *use cases* para a aplicação. Pode ainda controlar os dados sobre o interface HTTP. O controlo do fluxo e a apresentação visual do conteúdo foi delegado em grande parte para as camadas JavaScript do cliente [idem].

O AJAX não é um produto específico e não é um conjunto de instruções. Em vez disso, o AJAX é de forma resumida, o uso de um objecto específico do JavaScript chamado XMLHttpRequest combinado com eventos JavaScript e DHTML (Dynamic HTML).

Na construção de aplicações *Web* clássicas, o protocolo HTTP é usado para comunicar entre o *browser* e o servidor. A forma principal de interacção do utilizador com a aplicação é através de *hiperlinks* e formulários HTML, e ambos geram pedidos HTTP no *browser*. Uma limitação destes métodos de interacção é que estes populam automaticamente a página *Web* corrente com a resposta. Isto é, são desenhados para retornarem conteúdo na *Web*. Por vezes é necessário retornar dados e não conteúdo, ou apenas conteúdo específico para ser inserido em determinadas zonas da página. O objecto XMLHttpRequest foi desenvolvido com o propósito de ser uma solução para este problema, permitindo controlo programático sobre pedidos HTTP. Assim o objecto XMLHttpRequest permite-nos fazer pedidos ao servidor e receber a respectiva resposta, em vez de o *browser* construir de novo a página sempre que é feito um novo pedido ao servidor.

3.1.2.1 Frameworks de JavaScript e AJAX

Nos próximos dois pontos referem-se as *frameworks* escolhidas para implementar parte da camada de apresentação do sistema de Gestão de Ocorrências. Apesar de existirem bastantes disponíveis, optou-se pelo jQuery e pelo Prototype, devido à facilidade de implementação e por cumprirem todos os requisitos necessários para o cumprimento dos requisitos quer gráficos, quer de funcionalidade do lado do cliente

3.1.2.1.1 jQuery

A biblioteca jQuery fornece uma camada de abstracção para *Web scripting* sendo portanto muito útil em qualquer situação em que é necessário realizar esta tarefa do lado do cliente. A sua natureza expansível significa que os seus possíveis usos e funções estão sempre a crescer pois estão constantemente a ser desenvolvidos *plugins* para fornecerem novas funcionalidades. As funcionalidades de base satisfazem as seguintes necessidades [BIB08] [CHA07]:

- Aceder a elementos de uma página *Web*: Sem uma biblioteca de *JavaScript*, seriam precisas muitas linhas de código para aceder a qualquer elemento da árvore do DOM e localizar porções específicas da estrutura de um documento HTML. O jQuery oferece um mecanismo seleccionador (*Selectors*) que retorna exactamente a parte do documento que está a ser inspeccionado ou manipulado.

- Modificar a aparência de uma página *Web*: As CSS (Cascading Style Sheets) oferecem um método poderoso para modificar a forma como um documento é apresentado. No entanto estas falham quando os *Browsers* não suportam os mesmos standards. O jQuery ultrapassa este facto porque fornece o mesmo standard para todos os *Browsers*. Adicionalmente, o jQuery consegue modificar as classes ou propriedades de estilo individuais aplicadas a porções do documento mesmo depois de a página ter sido apresentada.

- Alterar o conteúdo de uma página *Web*: O jQuery permite alterar parte ou a totalidade do conteúdo da apresentação de uma página *Web* de forma dinâmica. O texto pode ser mudado, as imagens podem ser inseridas, as listas podem ser reordenadas, etc.

- Responder à interacção de um utilizador com a página.

● Adicionar animação a uma página: A biblioteca jQuery facilita esta tarefa providenciando um conjunto de efeitos (*fades*, *wipes*, etc.) e também um *toolkit* para criar novos efeitos visuais.

● Retornar informação de um servidor sem refrescar a página: Este padrão de código é conhecido por AJAX, e permite ao programadores *Web* criar *Rich Internet Applications*. A biblioteca jQuery remove deste processo a complexidade específica da programação do lado do cliente e permite aos programadores concentrarem-se apenas nas funcionalidades do servidor.

● Simplificar tarefas comuns do JavaScript.

3.1.2.1.2 Prototype

O Prototype fornece um conjunto de extensões de linguagem para o JavaScript, para o ambiente do *browser* e para o objecto XMLHttpRequest. Pode parecer estranho o facto de uma biblioteca JavaScript poder estender a linguagem na qual foi escrita, mas é exactamente isso que o Prototype faz. O JavaScript fornece um mecanismo conhecido por herança baseada em protótipo (daí deriva nome Prototype). As principais funcionalidades do Prototype são [CRA07]:

● AJAX: A *framework* Prototype permite trabalhar com chamadas AJAX de uma forma muito simples. Para além de simples pedidos, é também possível de uma forma muito intuitiva receber código retornado do servidor para posterior tratamento.

● Extensões ao DOM: A parte principal da *framework* Prototype é as suas extensões ao DOM. O Prototype adiciona muitos métodos que podem ser extremamente convenientes retornados pela função `$()`: por exemplo, para obter o elemento com o ID, dar-lhe um nome de classe e apresentá-lo, basta escrever: `$('comments').addClassName('active').show()`.

O elemento 'comments' não tinha os métodos aqui apresentados em JavaScript nativo. Isto mostra o poder desta framework.

● JSON (*JavaScript Object Notation*): é um formato de troca de dados e é uma alternativa rápida e eficaz ao uso de XML nos pedidos AJAX.

● Suporte ao mecanismo de herança: Em versões mais antigas desta *framework* apenas havia suporte para a criação de Classes. Actualmente existe também suporte ao mecanismo de herança, o que torna a programação orientada a objectos dentro desta *framework* bastante mais robusta.

3.1.3 Oracle

Uma base de dados Oracle é uma colecção de dados tratada como uma unidade. O objectivo de uma base de dados é guardar e devolver informação relacionada. Um servidor de base de dados é a chave para resolver problemas de gestão da informação. Em geral um servidor gere de forma eficaz uma grande quantidade de dados num ambiente multi-utilizador, permitindo que vários utilizadores possam em paralelo aceder aos mesmos dados. Um servidor de base de dados também impede o acesso não autorizado e fornece soluções eficientes para a recuperação de falhas [LON04].

A base de dados Oracle é a primeira para computação empresarial em grelha, que é a forma mais flexível e eficaz de gerir informação e aplicações. Com esta arquitectura, cada novo sistema pode ser rapidamente criado a partir do conjunto de componentes disponíveis. Não existe o risco de picos de carga, porque mais capacidade pode muito facilmente ser alocada ou realocada do conjunto de recursos conforme as necessidades [GRE04].

Os dados de estruturas lógicas, tais como tabelas ou índices, estão fisicamente guardados em ficheiros de dados alocados para a base de dados. Assim, no Oracle existem as seguintes estruturas físicas [ROD05]:

● Datafiles: cada base de dados Oracle tem um ou mais Datafiles físicos. Estes ficheiros contêm todos os dados da base de dados.

● Control Files: cada base de dados Oracle tem um Control File. Este ficheiro contém entradas que especificam a estrutura física da base de dados. Por exemplo, contém a seguinte informação:

- Nome da base de dados

- Nomes e localização de Datafiles e ficheiros Redo Log

- Data da criação da base de dados

● Redo Log Files*:* cada base de dados Oracle tem um conjunto de dois ou mais ficheiros Redo Log. A configuração destes ficheiros é conhecida como Redo Log para a base de dados. Um ficheiro Redo Log é constituído por entradas Redo (também conhecidas como registos Redo).A função primária do Redo Log é guardar todas as mudanças feitas aos dados. Se uma falha acontecer que impeça que dados modificados sejam escritos permanentemente nos ficheiros de dados, então as alterações podem ser obtidas do Redo Log e portanto o trabalho nunca é perdido.

● Archive Log Files: é possível fazer automaticamente o arquivo do Redo Log. O Oracle arquiva automaticamente os ficheiros quando estiver no modo "ARCHIVELOG".

● Parameter Files: estes ficheiros contêm uma lista de parâmetros de configuração para essa instância e para a base de dados.

● Alert Files e Trace Log Files: cada servidor e processo em *background* pode escrever para um Trace Log File associado. Quando um erro interno é detectado por um processo, é guardada a informação sobre o erro no Trace Log File associado. Alguma da informação escrita neste ficheiro serve para o administrador da base de dados, enquanto que outra informação é destinada aos serviços de suporte da Oracle. A informação de *trace* é usada para afinar aplicações e instâncias. O Alert File, é um ficheiro de *trace* especial. Este ficheiro é um registo cronológico de mensagens e erros.

● Backup Files: restaurar um ficheiro corresponde a substitui-lo pelo Backup File. Tipicamente, este restauro dá-se quando existe falha dos dispositivos ou quando erros dos utilizadores apagaram ou destruíram os ficheiros originais.

No Oracle existem as seguintes estruturas lógicas [LON04]:

● Tablespaces: uma base de dados é dividida em unidades lógicas de armazenamento chamadas Tablespaces, que agrupam estruturas lógicas relacionadas. Por exemplo, Tablespaces habitualmente agrupam todos os objectos da aplicação para simplificar tarefas administrativas.

● Oracle Data Blocks: no nível mais baixo, os dados de uma base de dados Oracle são guardados em Data Blocks. Um Data Block corresponde a um número específico de bytes de espaço em disco na base de dados física. O tamanho de bloco standard é especificado pelo parâmetro de inicialização DB_BLOCK_SYZE. Adicionalmente é possível especificar até cinco outros tamanho de bloco. Uma base de dados Oracle usa e aloca espaço livre da base de dados nos Data Blocks.

● Extents: um outro o nível de espaçamento lógico de dados é um Extent. Um Extent é um número específico de Data Blocks contíguos, obtidos numa alocação única, usado para guardar tipos de informação específicos.

● Segments: acima dos Extents, o nível lógico de armazenamento é denominado de Segment. Um Segment é um conjunto de Extents alocados para uma certa estrutura lógica.

Um *schema* é uma colecção de objectos da base de dados. Um *schema* pertence a um utilizador e tem o mesmo nome que o utilizador. Os Schema Objects são as estruturas lógicas que directamente se referem aos dados da base de dados. Os Schema Objects incluem estruturas tais como tabelas, vistas e índices. (Não existe relação entre um Tablespace e um Schema. Os objectos no mesmo Schema podem estar em diferentes Tablespaces, e um Tablespace pode conter objectos de diferentes Schemas).

Alguns dos Schema Objects mais comuns estão definidos a seguir [ROD05]:

● Tabelas: são a unidade básica de armazenamento numa base de dados Oracle. As tabelas contêm todos os dados acessíveis a um utilizador. Cada tabela tem colunas e linhas. Por exemplo, uma tabela de empregados, pode ter uma coluna chamada número do empregado e cada linha dessa coluna é o número do empregado.

● Índices: os índices são estruturas opcionais associadas a tabelas. Os índices podem ser criados para aumentar a performance do tratamento de dados. Tal como o índice deste documento ajuda a uma navegação rápida, também os índices Oracle providenciam acesso ao caminho para os dados da tabela.

● Vistas: as Vistas são apresentações de dados numa ou mais tabelas ou outras vistas. Uma vista pode ser considerada um *query*. As vistas na verdade não contêm dados, em vez disso, elas derivam os dados das tabelas nas quais são baseadas.

● Clusters **:** os Clusters são grupos de uma ou mais tabelas fisicamente guardadas em conjunto porque partilham colunas e são habitualmente usadas em conjunto. Como as linhas são guardadas juntas, o tempo de acesso ao disco diminui.

● Sinónimos: um sinónimo é um *alias* para qualquer tabela, vista, vista materializada, sequência, *procedure*, função, *package*, tipo, tipo de objecto definido pelo utilizador ou outro sinónimo. Porque um sinónimo é simplesmente um *alias*, não requer outro armazenamento para além da sua definição no dicionário de dados.

O SQL é a linguagem de programação que define e manipula a base de dados. As bases de dados SQL são relacionais, o que significa que os dados são guardados com um conjunto de relações simples.

Todas as operações sobre a informação numa base de dados Oracle é feita usando instruções SQL. Uma instrução SQL deve ser o equivalente a uma frase SQL do género:

```
SELECT ultimo_nome, departamento_id FROM colaboradores;
```

Pode ser vista como um programa ou instrução simples mas no entanto potente. As instruções podem ser divididas nas seguintes categorias [SIL07]:

● Linguagem de Definição de Dados (DDL-*Data Definition Language*) : estas instruções criam, alteram, mantêm e apagam Schema Objects. Também estão incluídas instruções que permitem a um utilizador dar privilégios para aceder à base de dados e objectos específicos da base de dados.

● Linguagem de Manipulação de Dados (DML-*Data Manipulation Language*) : este tipo de instruções manipula dados. Por exemplo seleccionar, inserir, alterar e eliminar linhas de uma tabela são operações DML. A instrução mais comum do SQL é o SELECT, que retorna dados de uma base de dados.

● Instruções de Transacção-Controle: estas instruções gerem as mudanças feitas pelas instruções DML. Elas permitem a um utilizador agrupar alterações em transacções lógicas. Exemplos incluem COMMIT, ROLLBACK e SAVEPOINT.

● Instruções de Controlo de Sessão: permitem a um utilizador controlar as propriedades da sessão corrente, incluindo ligar ou desligar *roles* e mudar as configurações de linguagem. Dois exemplos são ALTER SESSION e SET ROLE.

● Instruções de Controlo de Sistema: alteram as propriedades de uma instância da base de dados Oracle. O único existente é ALTER SYSTEM. Permite que um utilizador mude configurações tais como o número mínimo de servidores partilhados, terminar sessões entre outras tarefas.

● Instruções de SQL Embebidas: incorporam DDL, DML e instruções de transacção-controle num programa de linguagem procedimental, tais como as utilizadas com pré-compiladores Oracle. Os exemplos incluem OPEN,CLOSE, FETCH e EXECUTE.

O PL/SQL é uma extensão da linguagem SQL, combinando a simplicidade e flexibilidade do SQL com a funcionalidade procedimental de uma linguagem de programação estruturada, com instruções tais como IF…THEN, WHILE e LOOP. As vantagens a considerar com o uso de PL/SQL são [LON04]:

● O PL/SQL pode ser guardado de uma forma centralizada numa base de dados. Assim, o tráfego de rede entre aplicações e a base de dados é reduzida e como tal a performance geral do sistema é melhorada.

● Blocos de PL/SQL podem ser enviados por uma aplicação para a base de dados, fazendo operações complexas sem excessivo tráfego de rede.

● O acesso aos dados pode ser controlado por código PL/SQL guardado. Neste caso os utilizadores podem aceder aos dados apenas da forma que os programadores desejarem, a não ser que tenham mais privilégios do que os habituais.

● O Oracle suporta PL/SQL Server Pages, e portanto a lógica da aplicação pode ser invocada directamente a partir das páginas *Web*.

O PL/SQL consiste em stored procedures, funções, pacotes, *triggers* e transacções.

Funções e stored procedures são conjuntos de instruções SQL e PL/SQL agrupados como uma entidade unitária para resolver um problema específico ou para efectuarem um conjunto de tarefas relacionadas. Os stored procedures e funções são idênticos, excepto no facto de que as segundas retornam um valor e os primeiros não [idem].

Packages encapsulam e guardam stored procedures, funções, variáveis, entre outros e ficam construídos como uma unidade na base de dados. Eles oferecem funcionalidades extra, por exemplo, variáveis globais de pacote podem ser declaradas e usadas por qualquer stored procedure do mesmo. Também aumentam a performance como o caso de todos os objectos do *package* serem analisados (sintaxe), compilados e carregados em memória uma só vez [idem].

Cada valor de coluna numa instrução SQL tem um tipo de dados, que é associado a um formato específico de armazenamento.

Os tipos de dados do Oracle são [ROD05]:

- Character
- Numeric
- DATE / TIME
- LOB (Large Object)
- RAW and LONG RAW
- ROWID and UROWID

3.1.4 iText

A biblioteca iText permite a geração dinâmica de documentos do tipo PDF de uma forma simples. É ideal para programadores que necessitem de expandir as funcionalidade de aplicações *Web* através da criação de forma dinâmica de ficheiros *read-only* com a extensão .pdf. Através do uso das classes que constituem esta biblioteca é possível por exemplo [ITE]:

● Gerar dinamicamente ficheiros PDF a partir de ficheiros XML e Bases de Dados.

● Gerar dinamicamente ficheiros PDF a partir de dados que tenham sido inseridos em formulários HTML.

● Utilizar formulários PDF

● Adicionar assinaturas digitais a documentos PDF

3.1.5 Google Maps API

O Google Maps é um serviço de pesquisa e visualização de mapas dinâmicos na *Web*. A Google Maps API permite embeber Google Maps numa página *Web* utilizando para isso JavaScript. Esta API fornece um variado conjunto de utilidades para manipular mapas e adicionar conteúdo ao mapa através de vários serviços, permitindo a criação de mapas robustos em qualquer aplicação *Web*. A API permite entre outras funcionalidades criar marcadores para determinado ponto geográfico, modificar o mapa em função de determinada interacção com o utilizador, associar dados provenientes de ficheiros do tipo XML [GMD].

A API Google Maps é compatível com os seguintes *browsers*:

● IE 6.0+ (Windows)

● Firefox 2.0+ (Windows, Mac, Linux)

● Safari 3.1+ (Mac, iPhone)

● Chrome (Windows)

As classes que constituem esta API são as que se seguem [GMR]:

GMap2

Tabela 2 Core Class

GBounds	GInfoWindowTab	GMapOptions
GBrowserIsCompatible	GKeyboardHandler	GMapPane
GDraggableObject	GLanguage	GPoint
GDraggableObjectOptions	GLatLng	GSize
GInfoWindow	GLatLngBounds	GUnload
GInfoWindowOptions	GLog	G_API_VERSION

Tabela 3 Base Classes

GEvent	GEventListener

Tabela 4 Event Classes

GControl	GHierarchicalMapTypeControl	GMapUIOptions
GControlAnchor	GMapType	GMenuMapTypeControl
GControl	GMapTypeControl	GNavLabelControl
GControlPosition	GMapTypeOptions	

Tabela 5 Control Classes

GCopyright	GMercatorProjection	GScreenOverlay
GCopyrightCollection	GOverlay	GScreenPoint
GGroundOverlay	GPolyEditingOptions	GScreenSize
GIcon	GPolyStyleOptions	GTileLayer
GLayer	GPolygon	GTileLayerOptions
GMarker	GPolygonOptions	GTileLayerOverlay
GMarkerManager	GPolyline	GTileLayerOverlayOptions
GMarkerManagerOptions	GPolylineOptions	
GMarkerOptions	GProjection	

Tabela 6 Overlay Classes

GAdsManager	GGoogleBar	GStreetviewLink
GAdsManagerOptions	GGoogleBarAdsOptions	GStreetviewLocation
GAdsManagerStyle	GGoogleBarLinkTarget	GStreetviewOverlay
GClientGeocoder	GGoogleBarListingTypes	GStreetviewPanorama
GDirections	GGoogleBarOptions	GStreetviewPanorama.ErrorValues
GDirectionsOptions	GGoogleBarResultList	GStreetviewPanoramaOptions
GDownloadUrl	GPov	GTrafficOverlay
GFactualGeocodeCache	GRoute	GTrafficOverlayOptions
GGeoAddressAccuracy	GStep	GTravelModes
GGeoStatusCode	GStreetviewClient	GXml
GGeoXml	GStreetviewClient.ReturnValues	GXmlHttp
GGeocodeCache	GStreetviewData	GXslt

Tabela 7 Service Classes

Todas as classes referidas anteriormente têm métodos que permitem manipular objectos criados no Mapa.

Se quisermos por exemplo centrar o Mapa num determinado ponto geográfico, iremos usar o método `setCenter(center:GLatLng,zoom?:Number,type?:GMapType)` da classe principal – Gmap2. Se quisermos obter o ponto geográfico em que o Mapa está centrado, usaremos o método `getCenter()` da classe principal – Gmap2 que retorna um objecto do tipo GlatLng [GMD].

Toda a informação sobre as classes e respectivos métodos pode ser encontrada no manual de referência desta API [GMR].

A criação do Mapa mais elementar faz-se da seguinte forma:

```
<!DOCTYPE html "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

      <head>

    <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
    <title>Google Maps JavaScript API Example</title>
    <script src="http://maps.google.com/maps
file=api&amp;v=2&amp;key=abcdefg&sensor=true_or_false"
```

```
            type="text/javascript"></script>

    <script type="text/javascript">

       function initialize()
       {
              if (GBrowserIsCompatible())
              {

                     var map = new Gmap2(document.getElementById("map_canvas"));

                     map.setCenter(new GLatLng(37.4419, -122.1419), 13);

                     map.setUIToDefault();
        }
    }
    </script>

  </head>

       <body onload="initialize()" onunload="GUnload()">

              <div id="map_canvas" style="width: 500px; height: 300px"></div>

       </body>
</html>
```

Quando a página HTML em questão é carregada, é chamada a função `initialize()`. Esta função verifica se o *browser* que está a ser utilizado é compatível com a API. Se for, cria um objecto `map` do tipo Gmap2. Seguidamente usa os métodos `setCenter(center:GLatLng,zoom?:Number,type?:GMapType)` e `setUIToDefault()` da classe principal – Gmap2 de que o objecto `map` é uma instância.

3.1.6 Google Earth API

O Google Earth é uma aplicação de informação geográfica baseada num globo virtual, o que permite a visualização de mapas em três dimensões. Foi criada uma API para esta aplicação de forma a ser possível embebe-la numa página *Web*.

A Google Earth API tal como a Goolge Maps API tem um conjunto variado de funcionalidades que permite uma interacção avançada com o utilizador. É possível criar marcadores para localizações geográficas, navegar em 3 dimensões pelo globo, associar dados através de ficheiros KML (Keyhole Markup Language), etc.

3.1.7 Ambiente de desenvolvimento utilizado

O Netbeans é um ambiente de desenvolvimento *open source* e multiplataforma que funciona em qualquer sistema operativo que suporte a JVM (Java Virtual Machine). É uma ferramenta desenhada para permitir aos programadores realizarem as todas as tarefas inerentes ao desenvolvimento de *software* na criação de aplicações *desktop*, empresariais, *Web* e para computação móvel.

Tem uma estrutura modular, o que permite que este IDE possa ser moldado ao gosto e necessidades de cada programador. É integralmente escrito em Java e suporta as linguagens Java, C, C++, Ruby e PHP. Suporta também linguagens de marcação como XML e HTML.

A versão base tem os seguintes recursos:

- Editor de código fonte
- Ferramentas de *build*
- Suporte a desenvolvimento colaborativo
- Integração com bases de dados
- Ferramentas para controlo de versão

Existem vários pacotes disponíveis para *download*, em função da sua utilização sendo sempre possível adicionar *plugins* a cada um destes pacotes.

3.1.8 Servidor *Web* utilizado

O Apache Tomcat é um contentor de Servlets desenvolvido pela Apache Software Foundation que implementa as especificações Java Servlet e JavaServer Pages da Sun Microsystems e fornece um ambiente de servidor HTTP onde o código Java pode ser executado. O Tomcat inclui ferramentas para configuração e gestão, e também pode ser configurado editando os ficheiros XML de configuração. Os utilizadores têm acesso ao código fonte e à forma binária do Tomcat sob uma licença Apache [WITEN] [WITPT].

3.2 Tecnologias concorrentes

As tecnologias descritas nos seguintes pontos são equivalentes, na grande maioria das funcionalidades, às tecnologias usadas para a construção desta aplicação. Por exemplo, é sobejamente conhecida a concorrência entre a tecnologia Java (Sun Microsystems) e a tecnologia .Net (Microsoft Corporation) que oferecem produtos com características globalmente similares. Assim, é feita uma descrição resumida dessa tecnologias e é invocada o motivo pela não escolha dessas tecnologias.

3.2.1 PHP

O PHP (Hypertext Preprocessor) é uma linguagem código aberto de *scripting* para uso geral e foi originalmente desenvolvida para a *Web* tendo tido bastante sucesso entre os programadores de aplicações *Web*. A sua sintaxe é muito parecida à de outras linguagens orientadas a objectos, tais como: C++ e Java. O PHP suporta entre outras as bases de dados mais comuns: Oracle, mySQL, MSSQL. Os protocolos SNMP, POP3, IMAP são também suportados pelo PHP.

O motivo da não escolha desta tecnologia foi o facto de o INOSS ser construído com base na tecnologia Java e portanto o uso da tecnologia PHP não seria uma continuação lógica.

3.2.2 .NET

A plataforma Microsoft .NET é uma plataforma que pode ser instalada em computadores com o sistema operativo Windows. É constituída por uma grande colecção de bibliotecas e uma máquina virtual que gere a execução dos vários programas que forem criados para esta plataforma. O ambiente de desenvolvimento normalmente usado para criar aplicações em .NET é o Visual Studio que suporta as seguintes linguagens: C++, C#, VB.NET e ASP.NET

O motivo da não escolha desta tecnologia foi o facto de o INOSS ser construído com base na tecnologia Java e também o facto da tecnologia Microsoft não ser código aberto o que encareceria os custos do projecto.

4 A aplicação

Neste capítulo faz-se uma descrição detalhada da aplicação desenvolvida no âmbito deste projecto, SIMS, desde o modelo que serviu à sua implementação, até ao seu *interface*. No contexto de algumas funcionalidades será referida a utilização de determinadas tecnologias para melhor justificar o seu propósito.

4.1 Modelo

O modelo escolhido foi o modelo MVC (Model View Controloller). Este é um modelo cliente/servidor de 3 camadas habitualmente usado em programação de aplicações *Web*. O uso correcto desta arquitectura isola a lógica de negócio do *interface* do utilizador, resultando numa aplicação em que é mais fácil alterar quer o aspecto gráfico da aplicação quer a sua lógica de negócio sem que as alterações afectem cada uma das camadas lógicas.

***M**odel* representa a informação, isto é, os dados da aplicação.

À ***V**iew* correspondem os elementos do interface visual, isto é, esta camada renderiza a camada *Model* numa forma ajustada para a interacção com o utilizador. Nesta camada existem componentes tais como texto, campos de *input*, formulários, etc;

***C**ontroller* é responsável pela comunicação entre os dados e as regras de negócio usadas para manipular os dados de e para o *Model*, ou seja, processa e responde a eventos (habitualmente acções do utilizador) e pode indirectamente invocar alterações no *Model*.

A interacção entre as três camadas pode ser vista na figura seguinte (as linhas a sólido representam interacções directas e as linhas a tracejado representam as interacções indirectas):

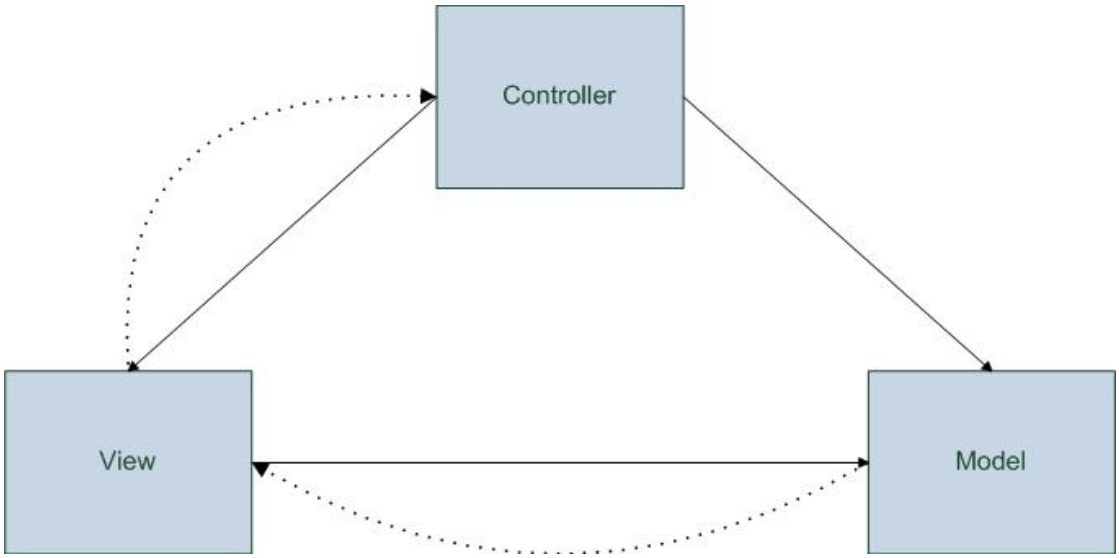


Figura 9 O modelo MVC

4.2 Infra-estrutura tecnológica

A aplicação fica instalada num servidor *Web* (Apache Tomcat). O repositório de informação estará num servidor HTTP (Apache). A base de dados é gerida por um servidor de base de dados (Oracle 10g). Por último, o envio de *emails* será processado por um servidor de *mail* (em fase de testes foi utilizado o servidor de e*mail* da Efacec mas a aplicação está preparada para trabalhar com qualquer outro como se poderá perceber mais à frente).

Na figura abaixo apresenta-se a estrutura montada para desenvolvimento e implementação desta aplicação:

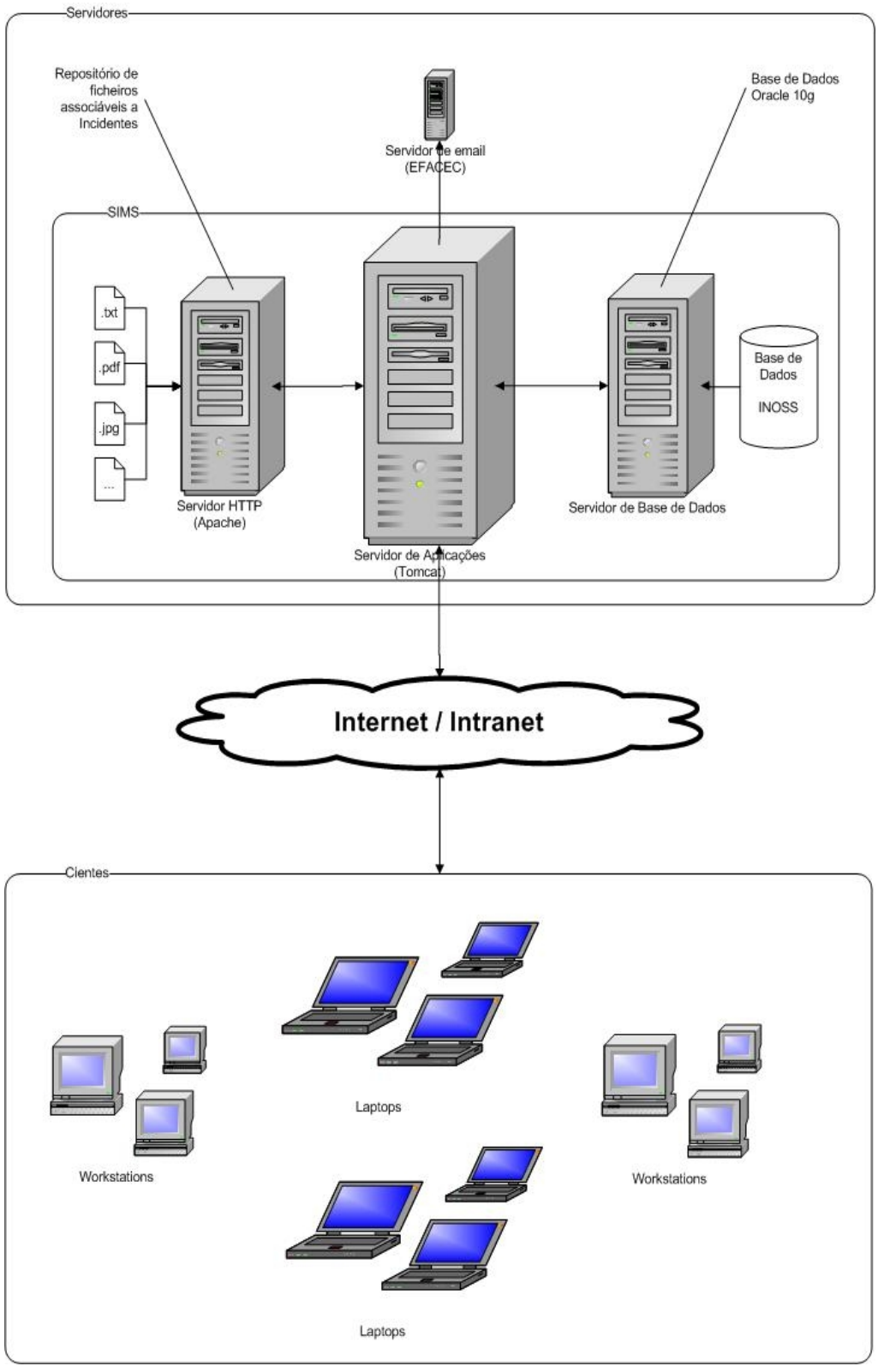


Figura 10 Infra-estrutura tecnológica da aplicação

4.3 Esquema de Navegação

Como já foi referido no capítulo 2, existem dois perfis de utilizadores: Administrador e Operador. É importante neste ponto realçar que estes perfis já se encontram parametrizados na tabela T_USERS da base de dados INOSS_SYSTEM (este facto será explicado com maior detalhe no próximo ponto). Os esquemas de navegação para cada perfil são os que se seguem.

4.3.1 Perfil Administrador

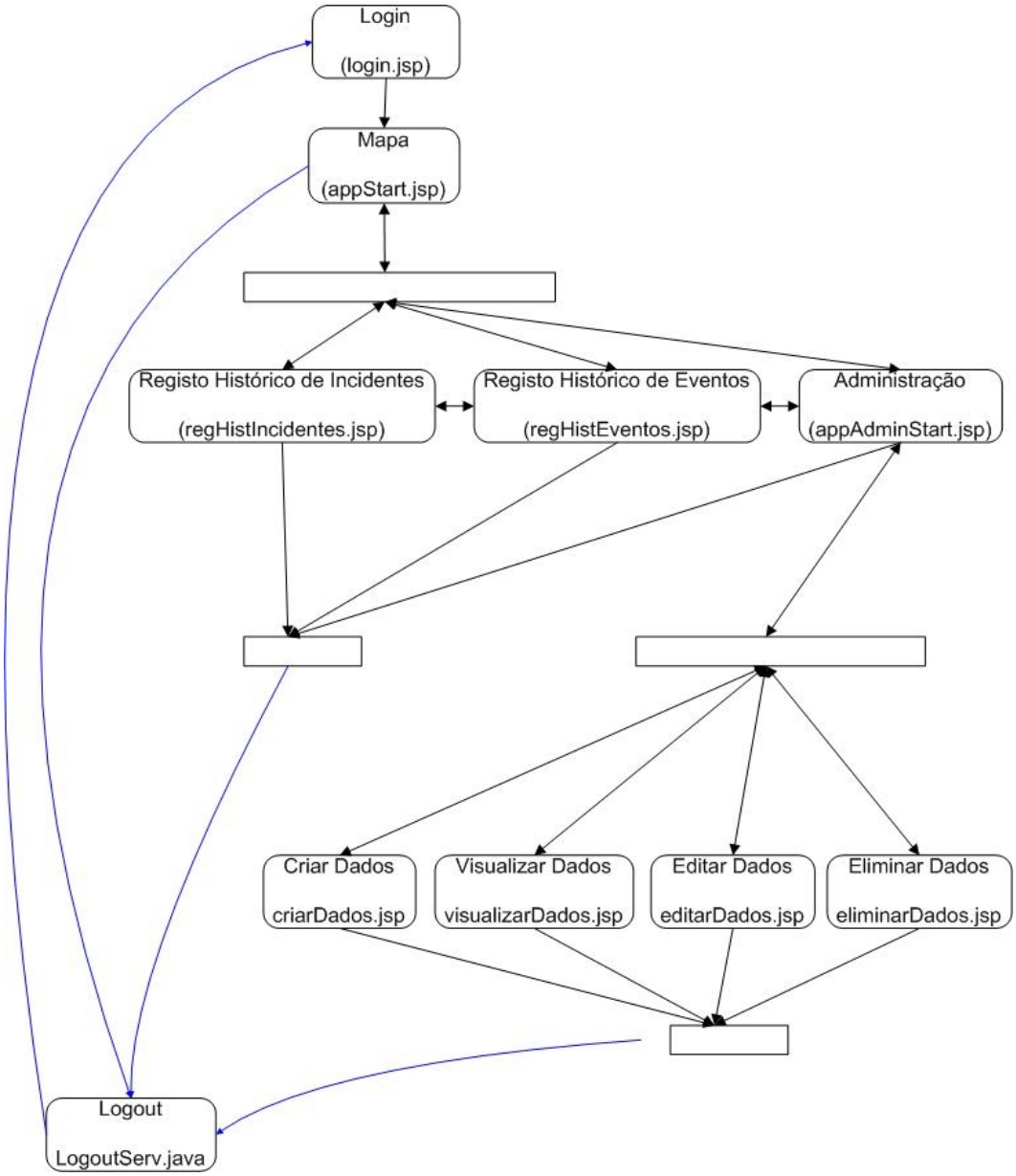


Figura 11 **Esquema de navegação para o perfil Administrador**

Neste caso o utilizador tem acesso à área funcional e também à área de administração. É portanto um utilizador com acesso à totalidade das funcionalidades da aplicação.

4.3.2 Perfil Operador

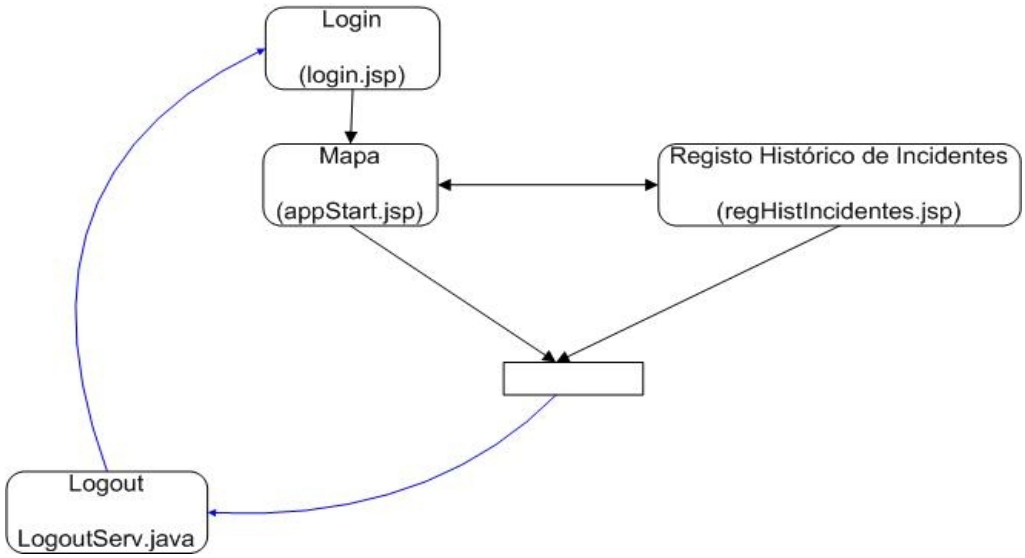


Figura 12 Esquema de navegação para o perfil Operador

Um utilizador com o perfil operador tem acesso apenas a uma parte das funcionalidades da área funcional e não tem acesso à área de administração.

4.4 Base de dados

O INOSSv2 é constituído por um conjunto de bases de dados, cada uma delas com propósitos específicos. Esta aplicação só utiliza uma base de dados particular: INOSS_SYSTEM. Para implementar as funcionalidades da especificação de requisitos foi necessário criar um conjunto de tabelas e uma função PL SQL.

A fig. 13 representa uma parte da base de dados INOSS_SYSTEM do INOSSv2. As tabelas que não estão sombreadas foram criadas especificamente para esta aplicação, sendo o restante algumas tabelas nativas da base de dados INOSS_SYSTEM e com as quais a aplicação SIMS também trabalha.

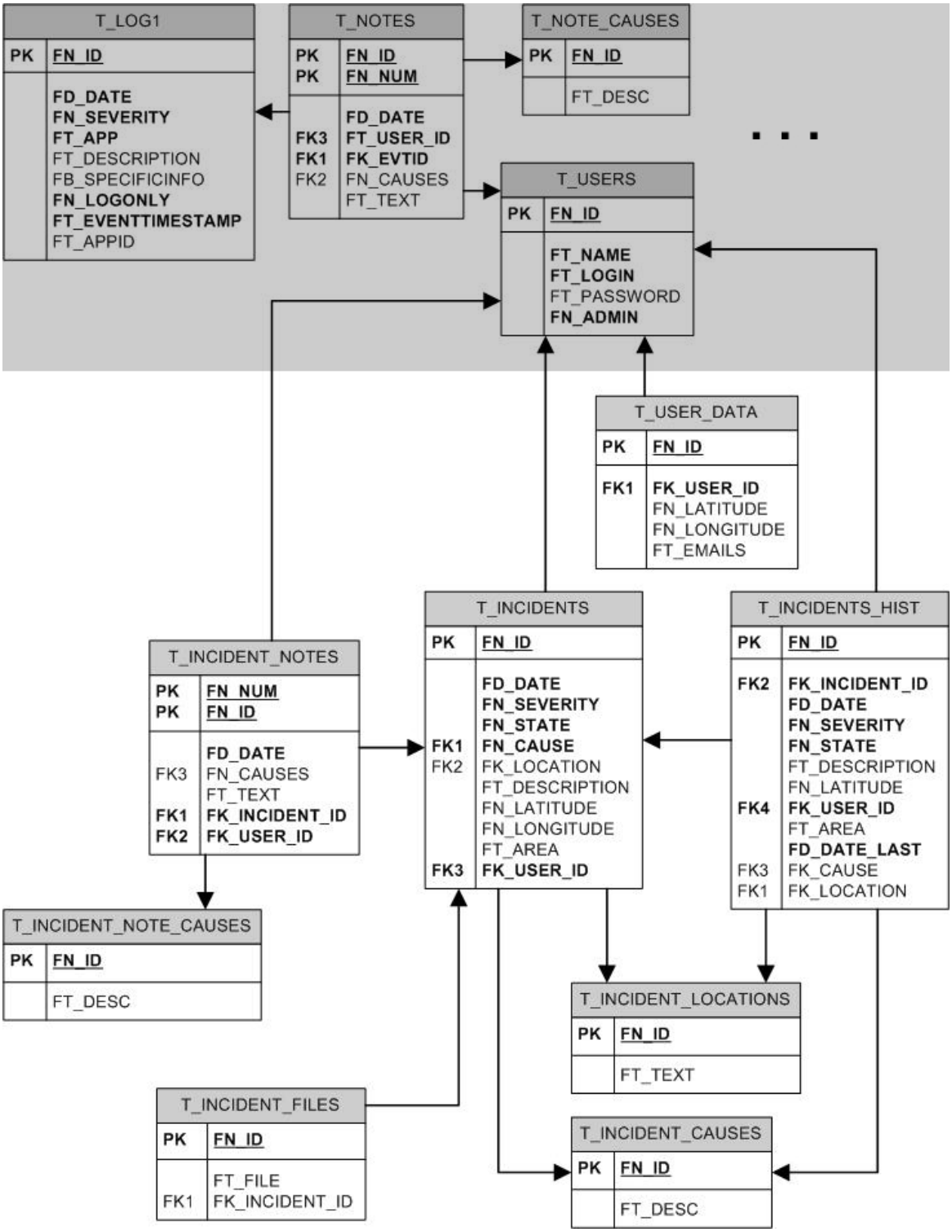


Figura 13 Diagrama relacional da base de dados

As tabelas criadas foram as seguintes:

- T_USER_DATA: guarda os dados (latitude e longitude do ponto de inicialização do mapa e endereços de *email*) que são criados na área de administração.
- T_INCIDENTS: guarda os dados de um determinado Incidente.
- T_INCIDENTS_HIST: guarda as alterações de um determinado Incidente.
- T_INCIDENT_CAUSES: contém as causas que podem ser associadas a um Incidente.
- T_INCIDENT_LOCATIONS: contém as localizações que podem ser associadas a um Incidente.
- T_INCIDENT_FILES: guarda os nomes de ficheiros que foram associados a determinado Incidente.
- T_INCIDENT_NOTES: guarda os dados de uma Nota que foi associada a um Incidente.
- T_INCIDENT_NOTE_CAUSES: contém as causas que podem ser associadas a uma Nota.

Criou-se também a seguinte função:

- INSERT_INCIDENT_NOTE: Associa uma Nota a um Incidente

O motivo da criação desta função prende-se com o facto de o SQL envolvido ser bastante complexo (mais do que uma instrução necessária) e a tarefa ser muito comum. De forma a optimizar uma operação que se esperava ser muito pesada, envolvendo ainda a ligação JDBC, optou-se por implementar a operação na própria base de dados, aproveitando a funcionalidade que o Oracle oferece.

4.5 Estrutura do projecto

O projecto é constituído por seis pastas principais e pelo ficheiro build.xml

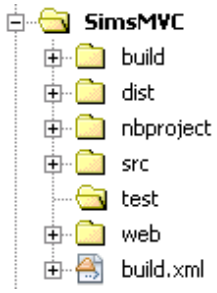


Figura 14 Estrutura do projecto

Vamos apenas referir o conteúdo das pastas web e src, porque são as que contêm o código da aplicação. As outras pastas têm uma função acessória, não sendo por isso relevante a explicação do seu conteúdo.

- web: Esta pasta é constituída por 10 sub-pastas e pelos ficheiros *.jsp e *.html

    - web\WEB_INF\lib: contém todos os ficheiros com a extensão .jar, que constituem as bibliotecas das tecnologias iText e JavaMail.

    - web\META-INF: contém o ficheiro context.xml.

    - web\css: contém todos os cascading style sheets utilizados na aplicação pelos *plugins* do jQuery.

    - web\images: contém todas as imagens utilizadas na aplicação.

    - web\jQuery: contém os ficheiros que constituem as bibliotecas jQuery e jQuery UI.

    - web\javaScript: contém os ficheiros com a extensão .js responsáveis pelo funcionamento do *plugin* que disponibiliza a data.

    - web\js: contém os ficheiros com a extensão .js responsáveis pelo funcionamento do *plugins* do JQuery.

○ web\prototype: contém os ficheiros com a extensão .js responsáveis pelo funcionamento do prototype.

○ web\resources: contém o ficheiro sims.properties que guarda as propriedades do projecto.

○ web\theme: contém os ficheiros .css do jQuery UI.

● src: Esta pasta é constituída por 2 sub-pastas: src\conf e src\java

A pasta src\java contém todas a *packages* utilizadas na aplicação:

○ src\java\admin: Contém a classe que permite lidar com os dados da área de administração.

○ src\java\base: Contém a classe que permite aceder às propriedades do projecto.

○ src\java\event: Contém todas as classes que lidam com os eventos.

○ src\java\files: Contém todas as classes que permitem realizar operações sobre os ficheiros que podem ser associados a incidentes.

○ src\java\incident: Contém todas as classes que lidam com os incidentes.

○ src\java\login: Contém todas as classes que lidam com o mecanismo de login e logout.

○ src\java\mail: Contém todas as classes que gerem o envio de *emails* de alerta.

○ src\java\markers: Contém todas as classes necessárias para caracterizar um marcador do mapa que represente um incidente.

○ src\java\navega: Contém todas as classes que gerem a navegação pela aplicação.

○ src\java\note: Contém todas as classes que lidam com as notas que se podem associar a incidentes e eventos.

○ src\java\pdf: Contém todas as classes que servem para a exportação de dados para PDF.

4.6 Interface com o utilizador

Nesta secção apresenta-se o interface da aplicação desenvolvida, acompanhando sempre que possível, com diagramas concebidos na fase de análise deste projecto (*use cases* ou *activity diagrams*) e que serviram de base para o desenvolvimento das funcionalidades descritas.

Figura 15 ***Use Case*** **Administrador**

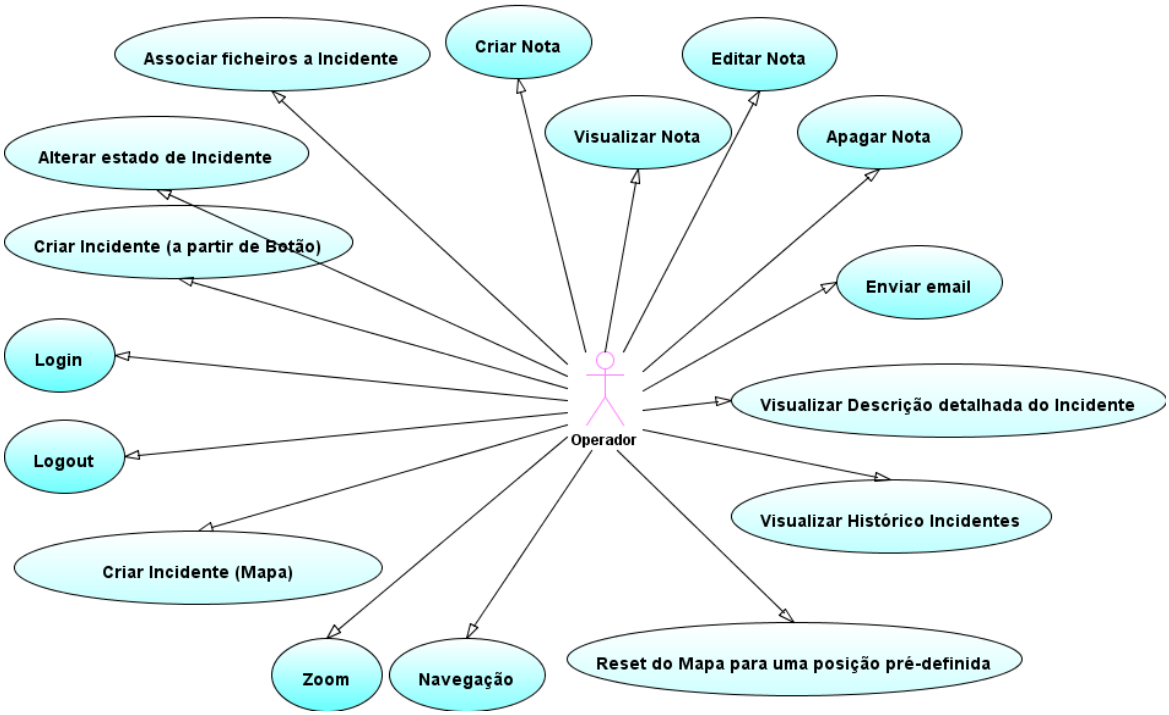


Figura 16 ***Use Case*** **Operador**

Na fase de análise foram detalhadas todos as funcionalidades da aplicação recorrendo a uma modelação UML, sendo os *use cases* parte dessa técnica. Aqui apresentam-se apenas os que se consideram mais importantes para se ter uma visão global, não se considerando relevante para esta tese a inclusão de todos os outros (aproximadamente 50 como se pode verificar pelas figuras acima).

4.6.1 Login

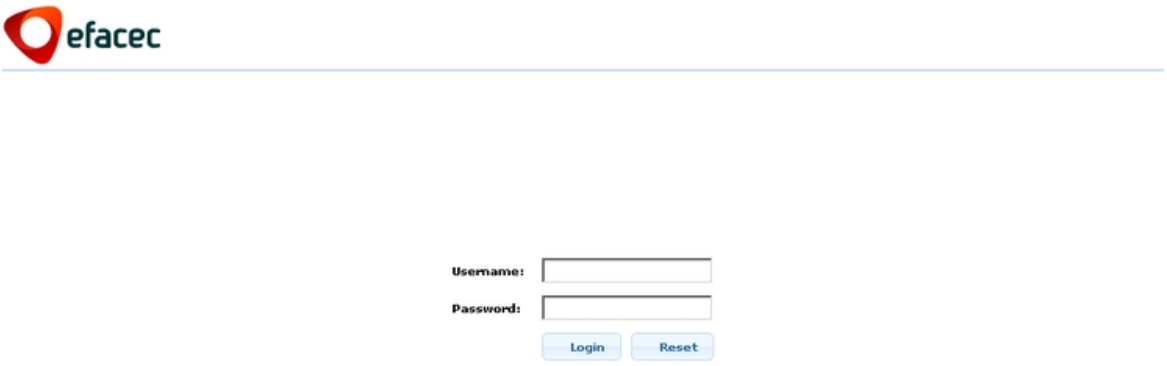


Figura 17 Login da aplicação

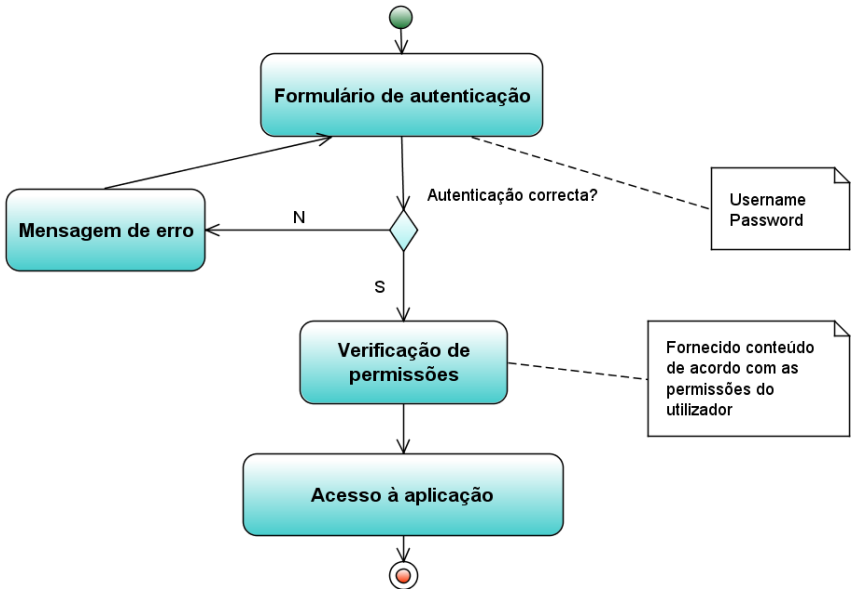


Figura 18 *Activity Diagram* : Login da aplicação

A fig. 17 apresenta a página de entrada na aplicação onde é disponibilizado um formulário de *login* que o utilizador deve preencher introduzindo um *username* e uma *password* válidos, sendo que em caso de erro aparecerá uma mensagem.

Em função do tipo de utilizador (operador ou administrador) que fizer login será disponibilizado ao longo da aplicação conteúdo adequado a cada um dos tipos de utilizador.

Nesta página é carregado um ficheiro (sims.properties) que contém as propriedades do projecto (*urls* dos servidores, endereço da base de dados e configurações do JavaMail).

4.6.2 Mapa (Home)

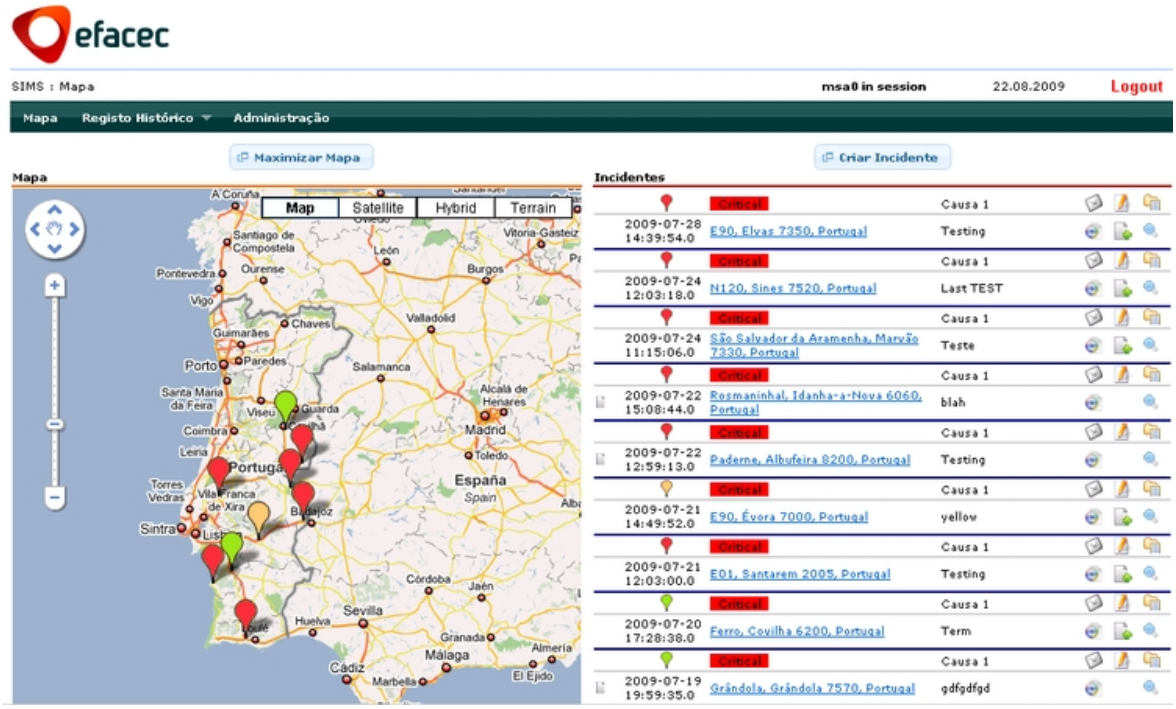


Figura 19 Mapa (Home)

O corpo principal da página é constituído por um Mapa dinâmico, dois botões - Maximizar Mapa e Criar Incidente - e uma Tabela.

O botão Maximizar Mapa abre uma janela (*Modal window*) com uma versão maximizada do Mapa em tudo igual ao apresentado inicialmente mas com a funcionalidade extra de vista *Earth* (vista que lança uma instância embebida da aplicação Google Earth). A partir do botão Criar Incidente é apresentado um formulário *(Modal window)* onde deverão ser

preenchidos os dados relativos a determinado incidente (esta operação está detalhada mais à frente).

O Mapa dinâmico, implementado com a Google Maps API, é um mapa que oferece as funcionalidades nativas de navegação, zoom e vistas (*Map*, *Satelite*, *Hybrid*, *Terrain*), e a funcionalidade de menu contextual desenvolvida especificamente para este projecto (por exemplo a criação de Incidente a partir de um ponto do mapa).

Figura 20 Mapa (Detalhe)

O código para a apresentação do Mapa encontra-se na camada de apresentação e de servidor:

No ficheiro appStart.jsp o código envolvido na apresentação do Mapa é o seguinte:

● Criação do Mapa

```
...
                var map = new GMap2(document.getElementById("map"));
                <%
                if(lat==null || lon==null)
                {
                    %>
                            map.setCenter(new GLatLng(39.48,-8.10),6);
                    <%
                }
                else
                {
                    %>
                            map.setCenter(new GLatLng(<% out.println(lat.trim());
%>,<% out.println(lon.trim()); %>),6);
                    <%
                }
                %>
                map.setUIToDefault();
                var mapControl = new GMapTypeControl();
                map.addControl(mapControl);
...
```

Este segmento de código representa a criação do objecto mapa (instância da classe Gmap2). O Mapa é centrado no ponto com a latitude: 39.48º e longitude: -8.10º no caso de o utilizador em sessão não ter um ponto de *reset* associado. Seguidamente define-se o UI do Mapa e é criada a barra de navegação.

● Popular Mapa com marcadores

```
...

                GDownloadUrl("viewXML.jsp", function(data)
                {
                    var xml = GXml.parse(data);
                    var markers =
xml.documentElement.getElementsByTagName("marker");
                    for (var i = 0; i < markers.length; i++)
                    {
                        var id = markers[i].getAttribute("id");
                        var point = new
GLatLng(markers[i].getAttribute("latitudeBD"),markers[i].getAttribute("longitudeB
D"));

...
                        var marker = createMarker(id, point, area,
estado,descricao, causa, gravidade,hasNote);

                        map.addOverlay(marker);
                    }
                });
...
```

Nesta fase a aplicação vai ler um ficheiro externo (viewXML.jsp) que vai gerar dinamicamente um ficheiro do tipo XML. Este XML contém a lista de todos os marcadores com os dados que caracterizam cada Incidente. A partir deste dados são criados os marcadores no Mapa.

```
...
function createMarker(id, pointP, area, estado, descricao, causa, gravidade,
hasNote){
   ...
       if(estado=="0"){
              marker = new GMarker(pointP, markerOptionsRed);
              estadoS="Iniciado";
       }
       else{
              ...
       }

       if (gravidade==1){
              sevS="Critical";
              cellColor="red";
       }
       else{
              ...
       }


       if(hasNote=='true'){
              var html=...
       }
       else{
              var html=...
       }
       GEvent.addListener(marker, 'click', function({
              marker.openInfoWindowHtml(html);});

       return marker;

}
...
```

Na criação de cada um destes marcadores é também gerado o HTML correspondente à janela de informação e acções (InfoWindowHtml) que lhe estão associados.

● Menu contextual

```
...

                var contextmenu = document.createElement("div");
                contextmenu.style.visibility="hidden";
                contextmenu.style.background="#C3D9FF";
                contextmenu.style.border="1px solid #C3D9FF";


                contextmenu.innerHTML ='<a
href="javascript:criarOcorrencia()"><div class="context">  Criar
ocorrência  <\/div><\/a>'

                    + ...

                    + '<a href="javascript:centreMapHere()"><div
class="context">  Centre map here  <\/div><\/a>';
```

```
                map.getContainer().appendChild(contextmenu);

                // === listen for singlerightclick ===
                GEvent.addListener(map,"singlerightclick",function(pixel,tile) {

            ...
                    var pos = new GControlPosition(G_ANCHOR_TOP_LEFT, new
GSize(x,y));
                    pos.apply(contextmenu);
                    contextmenu.style.visibility = "visible";
                });

...


}

...
```

O menu contextual é composto por um conjunto de funções JavaScript, disponibilizadas através de links (href). É implementado um listener para o evento de right-click do rato sobre o mapa, apresentando o menu e guardando as coordenadas do ponto em memória para mais tarde serem utilizadas na criação de ocorrência.

Detalha-se agora o ficheiro viewXML.jsp que foi referido anteriormente como sendo usado na criação de marcadores.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<%@ page contentType="text/xml" %>
<%@ page import="markers.*" %>
<jsp:useBean id="portfolio" class="markers.MarkersBean" />

<%
    java.util.Iterator folio =portfolio.getPorfolio();
    Marker marker = null;
%>
<markers>
<%
    while (folio.hasNext())
    {
        %>
        <% marker = (Marker)folio.next(); %>

        <marker id="<%=marker.getId() %>" gravidade="<%=marker.getGravidade() %>"
estado="<%=marker.getEstado() %>" descricao="<%=marker.getDescricao() %>"
causa="<%=marker.getCausa() %>" latitudeBD="<%=marker.getLatitudeBD() %>"
longitudeBD="<%=marker.getLongitudeBD() %>" area="<%=marker.getArea() %>"
hasNote="<%=marker.getHasNote() %>"/>

    <%
    }
    %>
</markers>
```

Aqui é usado o método `getPortfolio` da classe MarkersBean para devolver da base de dados a lista dos incidentes que foram criados a partir do mapa. É. gerado um xml que tem como raíz `<markers>` e a lista é percorrida para gerar os nós `<marker>`, cada um

representativo de um marcador (e como tal de um incidente) e com os atributos correspondentes.

A classe MarkersBean tem apenas um propriedade do tipo Vector (implementa lista) que é inicializado no construtor do objecto (nova instância). Esta inicialização consiste na execução de várias instruções SQL que obtêm os dados necessários da base de dados e criação de vários objectos do tipo Marker (bean) que são acrescentados ao vector `portfolio`.

```
...

public class MarkersBean implements java.io.Serializable
{
    private Vector portfolio=new Vector();

    public MarkersBean()
    {

   ...

            //Apanha numero de marcadores

            rs = s.executeQuery("select count (*) from (select * from(select *
from (select T_INCIDENTS.FN_ID, FD_DATE, FN_SEVERITY, FN_STATE, FT_DESCRIPTION,
FT_DESC, FT_TEXT, T_INCIDENTS.FN_LATITUDE, T_INCIDENTS.FN_LONGITUDE,
T_USERS.FT_LOGIN, FT_AREA from T_INCIDENTS inner join T_INCIDENT_CAUSES  on
T_INCIDENTS.FK_CAUSE=T_INCIDENT_CAUSES.FN_ID left join T_INCIDENT_LOCATIONS on
T_INCIDENTS.FK_LOCATION=T_INCIDENT_LOCATIONS.FN_ID inner join T_USERS on
T_INCIDENTS.FK_USER_ID=T_USERS.FN_ID order by T_INCIDENTS.FN_ID desc) where
rownum <=10) where FT_TEXT IS NULL)");

            if(rs.next())
            {
                numMarc=rs.getInt(1);
            }

             //Apanha conteúdo dos marcadores

            rs = s.executeQuery("select * from (select T_INCIDENTS.FN_ID,
FN_SEVERITY, FN_STATE, FT_DESCRIPTION, FT_DESC, T_INCIDENTS.FN_LATITUDE,
T_INCIDENTS.FN_LONGITUDE, T_INCIDENTS.FT_AREA from T_INCIDENTS inner join
T_INCIDENT_CAUSES on T_INCIDENTS.FK_CAUSE=T_INCIDENT_CAUSES.FN_ID where
FN_LATITUDE IS NOT NULL order by T_INCIDENTS.FN_ID desc) where rownum
<="+numMarc);

            while(rs.next())
            {
                id=rs.getInt(1);
                gravidade=rs.getInt(2);
                estado=rs.getInt(3);
                descricao=rs.getString(4);
                causa=rs.getString(5);
                latitudeBD=rs.getBigDecimal(6);
                longitudeBD=rs.getBigDecimal(7);
                area=rs.getString(8);


                    //NOTA

            ...
```

```
                portfolio.addElement(new
Marker(id,gravidade,estado,descricao,causa,latitudeBD,longitudeBD,area,hasNote));


            }

      ...
    }

    public Iterator getPorfolio()
    {
        return portfolio.iterator();
    }

}
```

O ficheiro Marker.java é um JavaBean e representa um marcador:

```
...

public class Marker implements java.io.Serializable
{
   ...

    public Marker(int id, int gravidade, int estado, String descricao, String
causa, BigDecimal latitudeBD, BigDecimal longitudeBD, String area, boolean
hasNote)
    {
        this.id=id;
        this.gravidade=gravidade;
        this.estado=estado;
        this.descricao=descricao;
        this.causa=causa;
        this.latitudeBD=latitudeBD;
        this.longitudeBD=longitudeBD;
        this.area=area;
        this.hasNote=hasNote;

    }
      //Getters e Setters

    public void setId(int i)
    {
        id=i;
    }

    public int getId()
    {
        return id;
    }

...

    public void setHasNote(boolean b)
    {
        hasNote=b;
    }

    public boolean getHasNote()
    {
        return hasNote;
    }
}
```

A tabela anexa ao mapa lista os últimos dez Incidentes criados e é refrescada sempre que alguma acção que a altere seja efectuada. De facto, através do uso da tecnologia AJAX foi possível criar um modelo que apenas refresca certas partes da página, o que leva a um maior nível de usabilidade e de desempenho.

Figura 21 Tabela (Detalhe)

Cada conjunto de duas linhas delimitada por um separador azul, representa os dados de um Incidente:

- Estado: - Iniciado, - Em execução, - Terminado

- Gravidade
- Causa
- Data de criação
- Localização – Pode ter uma de duas origens: seleccionada pelo utilizador na criação de Incidente a partir do botão (T_ICIDENT_LOCATIONS) ou preenchida automaticamente quando a ocorrência é criada a partir do mapa. Neste último caso, o utilizador pode agir sobre o *link* associado provocando que o mapa fique centrado na mesma.
- Descrição

O ícone do lado esquerdo  representa a existência de Nota associada a cada Incidente.

Os ícones do lado direito permitem as acções:

- Enviar *email* de alerta 
- Editar Incidente 
- Visualizar Histórico das alterações de Incidente 
- Associar ficheiro 
- Criar Nota 
- Visualizar Incidente 

A tabela encontra-se no ficheiro smallIncidentTable.jsp que por sua vez está embebido na página appStart.jsp possibilitando desta forma o refrescamento assíncrono da tabela.

4.6.3 Criar Incidente (Mapa)

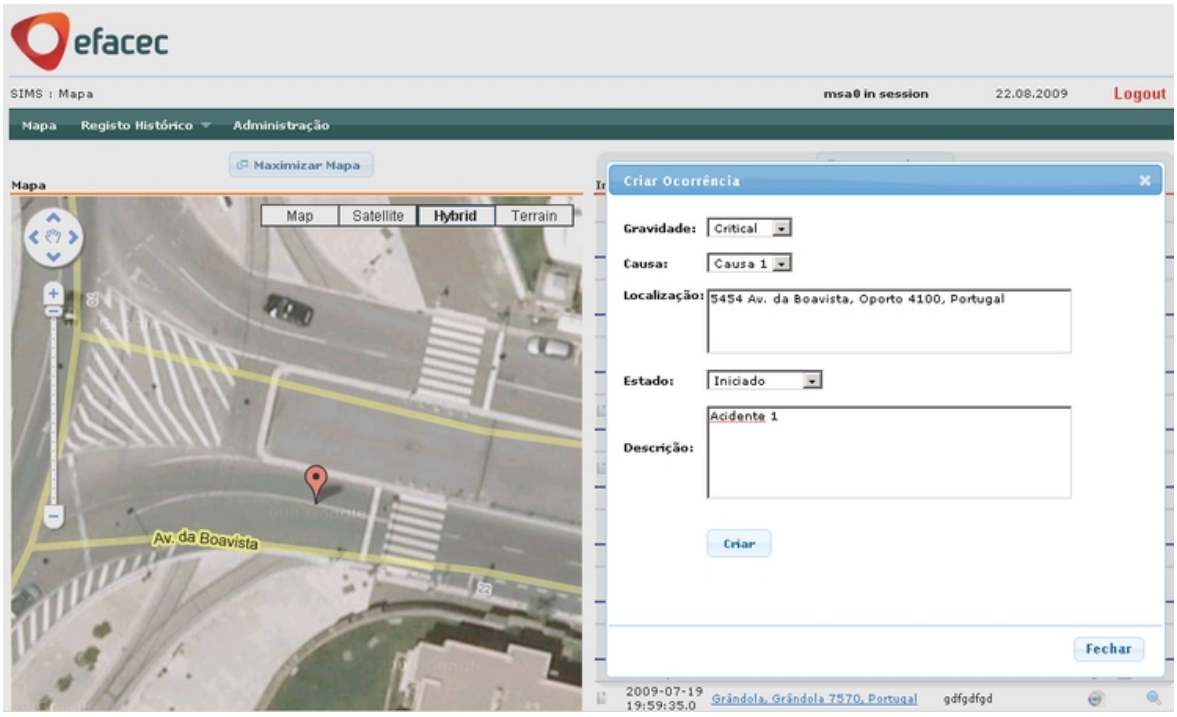


Figura 22 Criar Incidente (Mapa)

Esta acção é iniciada através da opção "Criar Incidente" disponibilizada pelo menu contextual referido no ponto 4.5.2. É disponibilizado um formulário (*Modal Window*) que o utilizador deve preencher. Os campos deste formulário são:

- Gravidade – Info, Warning, Minor, Major, Critical
- Causa – Combobox preenchida a partir dos dados da tabela T_INCIDENT_CAUSES
- Localização – campo preenchido automaticamente através da utilização da Google Maps API
- Estado – Iniciado, Em Execução, Terminado
- Descrição – Texto livre

Quando se submete o formulário os dados são enviados de forma assíncrona para o servidor, sendo que em caso de sucesso a tabela de Incidentes é refrescada.

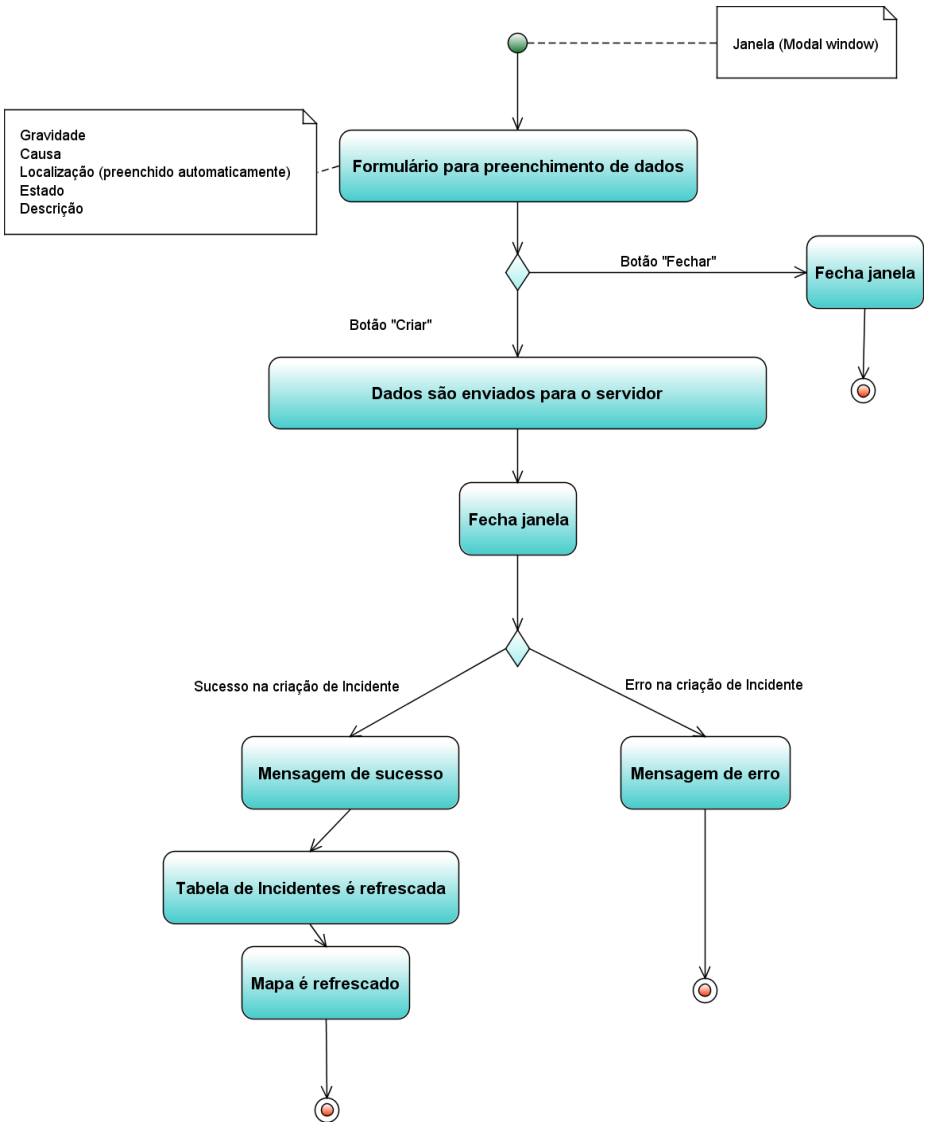


Figura 23 ***Activity Diagram*** **: Criar Incidente (Mapa)**

Para obter o endereço correspondente às coordenadas geográficas obtidas no ponto de criação de Ocorrência é usado um mecanismo de *reverse geocoding* (obtenção de endereço a partir de coordenadas geográficas).

```
...
       function criarOcorrencia()
      {
              var point = map.fromContainerPixelToLatLng(clickedPixel)

             var latitude=point.lat();
             var longitude=point.lng();

             var location = new GLatLng(latitude, longitude);

             var html = latitude + "<br>" +  longitude +" <br><input
type='button' value='Criar' onclick='map.closeInfoWindow();'/>";

              geocoder.getLocations(location, showAddress);

              contextmenu.style.visibility="hidden";

       }

...
```

Esta é a função invocada na criação de Ocorrência. Através dos métodos disponíveis nos objectos do Google Maps, é obtido o ponto geográfico com base na latitude e longitude relativos ao ponto clicado no mapa. É invocado o método `getLocations` da Google Maps API que deverá devolver além do status, uma estrutura do tipo Placemark. A função `showAddress` encarregar-se-à de tratar da resposta devolvida pelo método e de apresentar a janela de formulário e criar o marcador (temporário) em caso de sucesso.

```
...
       function showAddress(response)
       {

              if (!response || response.Status.code != 200) {
                     alert("Status Code:" + response.Status.code);
              }
              else {
                     place = response.Placemark[0];

                     var area = place.address;

                     var point = map.fromContainerPixelToLatLng(clickedPixel)

                     var latitude=point.lat();
                     var longitude=point.lng();
                     var location = new GLatLng(latitude, longitude);

                     markerDummy = new GMarker(location);
                     map.addOverlay(markerDummy);

                     abreModal(latitude,longitude,area);
              }


       }

...
```

A função `abreModal` é responsável pela criação da *Modal Window* que pode ser vista na fig. 22

```
...
            function abreModal(latitude,longitude,area)
            {
                jQuery('#formOcorrencia1').resetForm();

                document.getElementById("frmLat").value = latitude;
                document.getElementById("frmLon").value = longitude;

                document.getElementById("area").value = area;

                jQuery('#dialogOcorrencia1').dialog('open');

                jQuery('#mensagemErroCriarOcorrenciaMapa').hide();

             }

...
```

Ao longo de toda a aplicação, a apresentação de *Modal Windows* é feita utilizando a função `dialog` da biblioteca jQuery UI.

Na instrução `jQuery('#dialogOcorrencia1').dialog('open');` é executada a função `dialog` sobre o elemento HTML `dialogOcorrência1` (div), encapsulando-o numa *Modal Window* com determinadas características (apresentadas abaixo) e mostrando-a através do argumento `open` (é usado o argumento `close` para fechar).

```
...


      jQuery("#dialogOcorrencia1").dialog({
             modal: true, overlay: { opacity: 0.5, background: "black" } ,
             autoOpen:false,
             draggable: false,
             resizable: false,
             width : 470,
             position: [515,140],
             buttons:
             {
                    "Fechar": function()
                    {
                           jQuery(this).dialog("close");

                    }
             }
      });


...
```

Quando o utilizador carrega no botão “Criar” é invocado o método `ajaxForm`, um mecanismo que permite implementar uma pré validação do formulário antes de serem enviados os dados para o servidor e um outro que permite lidar com a resposta do servidor após envio dos dados.

```
jQuery('#formOcorrencia1').ajaxForm(optionsCriarOcorrenciaMapa);
```

O argumento desta função é uma estrutura composta pela declaração da função a invocar em cada um dos dois mecanismos:

```
  var optionsCriarOcorrenciaMapa = {

          beforeSubmit:  showRequestCriarOcorrenciaMapa,  // pre-submit callback
          success:       showResponseCriarOcorrenciaMapa  // post-submit callback

                        };
```

Na função `showRequestCriarOcorrenciaMapa`, é validado o campo descrição do formulário. Caso não haja erros os dados são enviados para o servidor, altura em que a base de dados será actualizada.

A função `showResponseCriarOcorrenciaMapa` trata a resposta enviada pelo servidor. Esta resposta é do tipo String e pode ser “Sucesso” ou “Erro”. No primeiro caso o mapa é re-inicializado com os dados actualizados e a tabela que se encontra ao seu lado é refrescada.

```
function carregaTabela()
{
       new Ajax.Updater('smallIncidentTable',
                        'smallIncidentTable.jsp',
                        {evalScripts:true});
}
```

A método `Updater` (da classe Ajax da biblioteca Prototype) faz um pedido AJAX e actualiza o conteúdo de um contentor baseado na resposta enviada pelo servidor.

Esta sequência de acções é utilizada na submissão de formulários ao longo de toda a aplicação.

4.6.4 Criar Incidente (Botão)

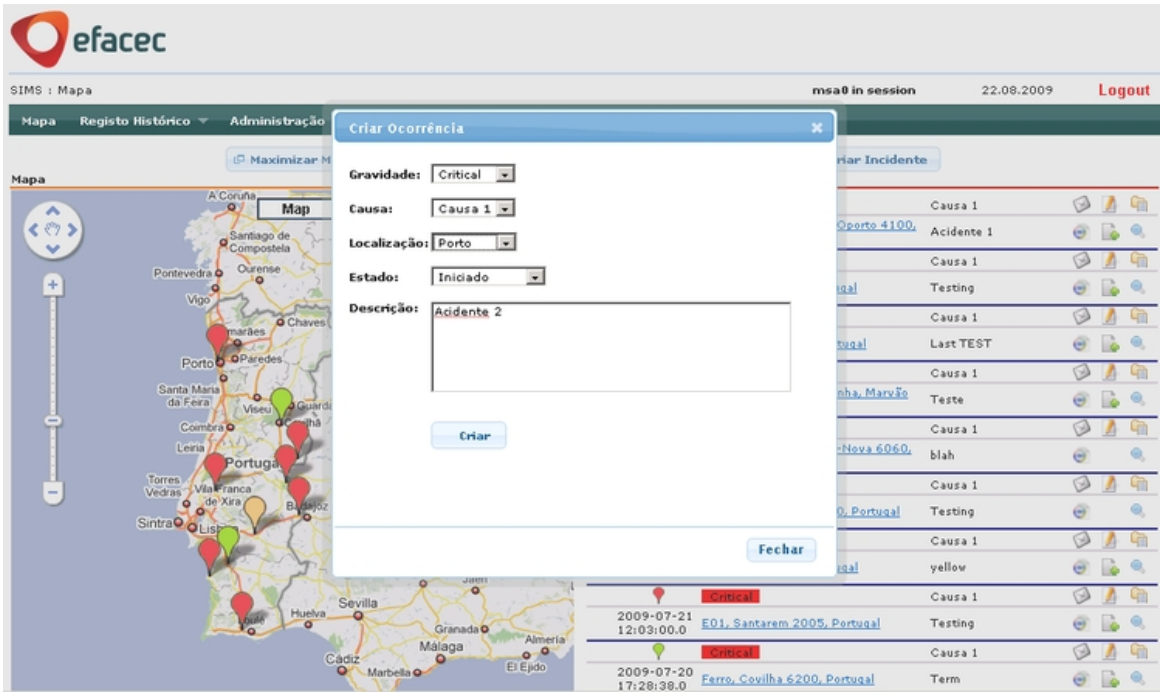


Figura 24 Criar Incidente (Botão)

Este caso de criação de Incidente é em tudo análogo ao anterior excepto as localizações serem pré-definidas (estão guardadas em base de dados - T_INCIDENT_LOCATIONS) e utilizador ter que escolher uma delas para associar ao incidente.

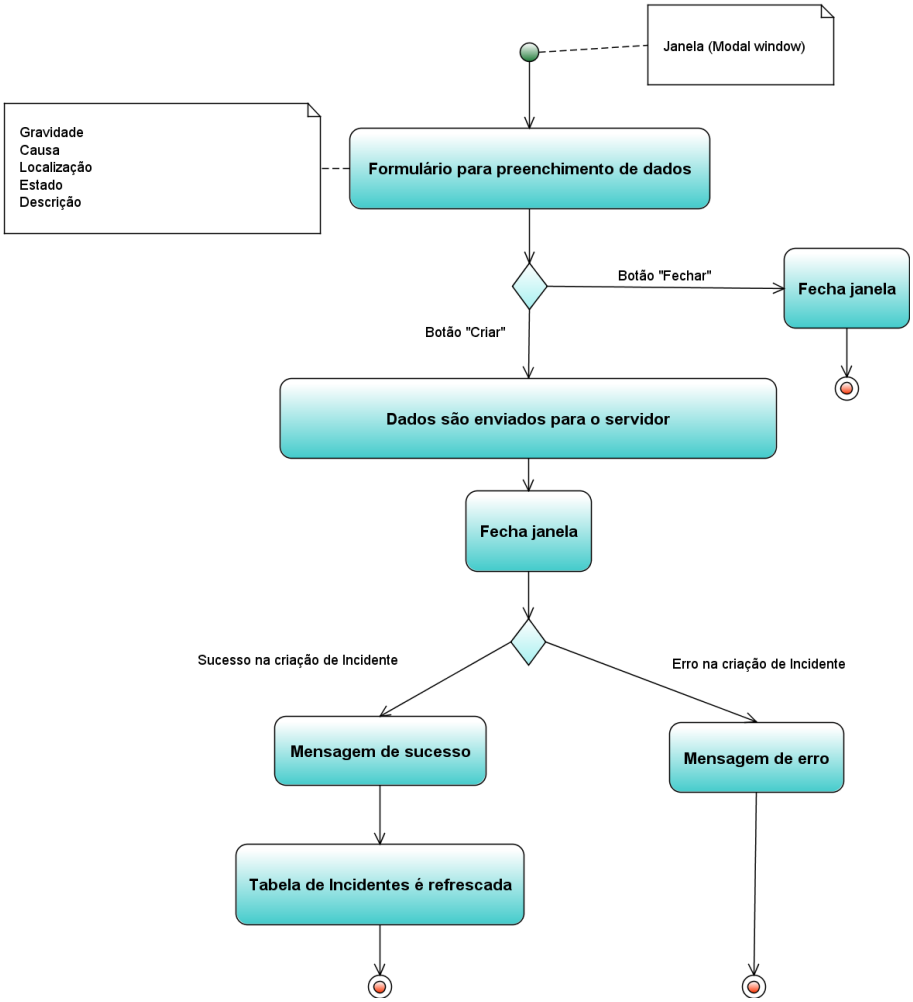


Figura 25 ***Activity Diagram*** **: Criar Incidente (Botão)**

4.6.5 Alterar Incidente

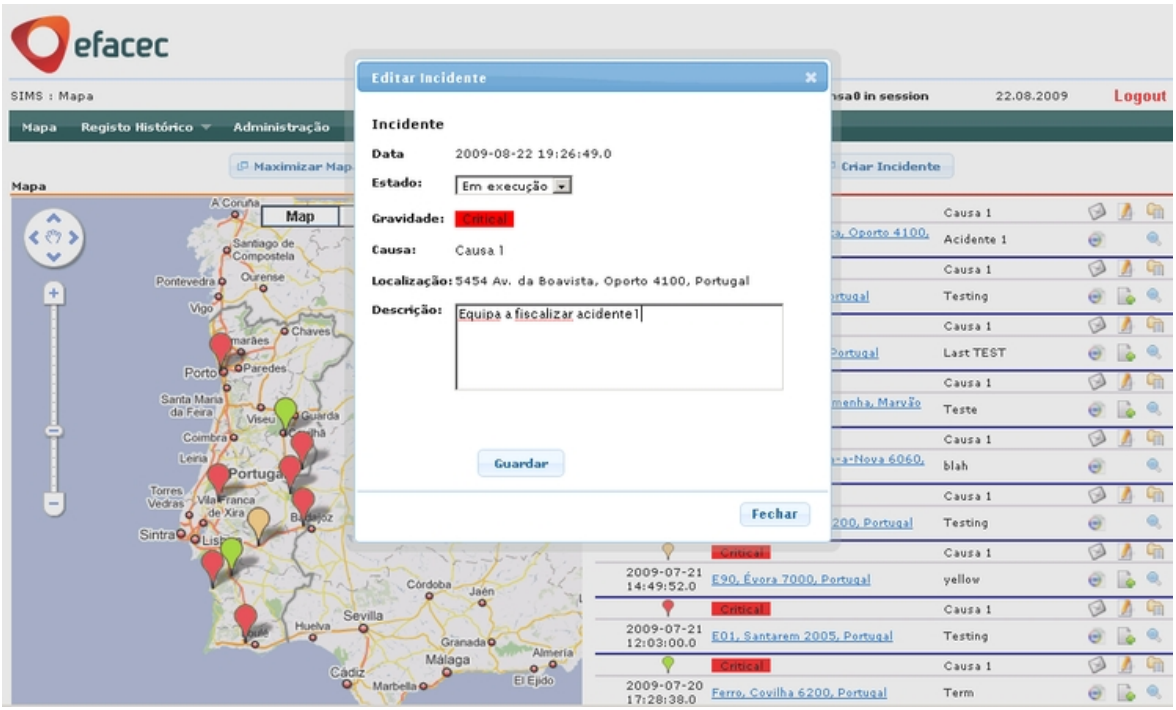


Figura 26 Alterar Incidente

É possível alterar o Estado e a Descrição de um determinado Incidente. Sempre que isto é feito, as alterações são guardadas em base de dados, sendo depois possível visualizar o Histórico das alterações de um Incidente.

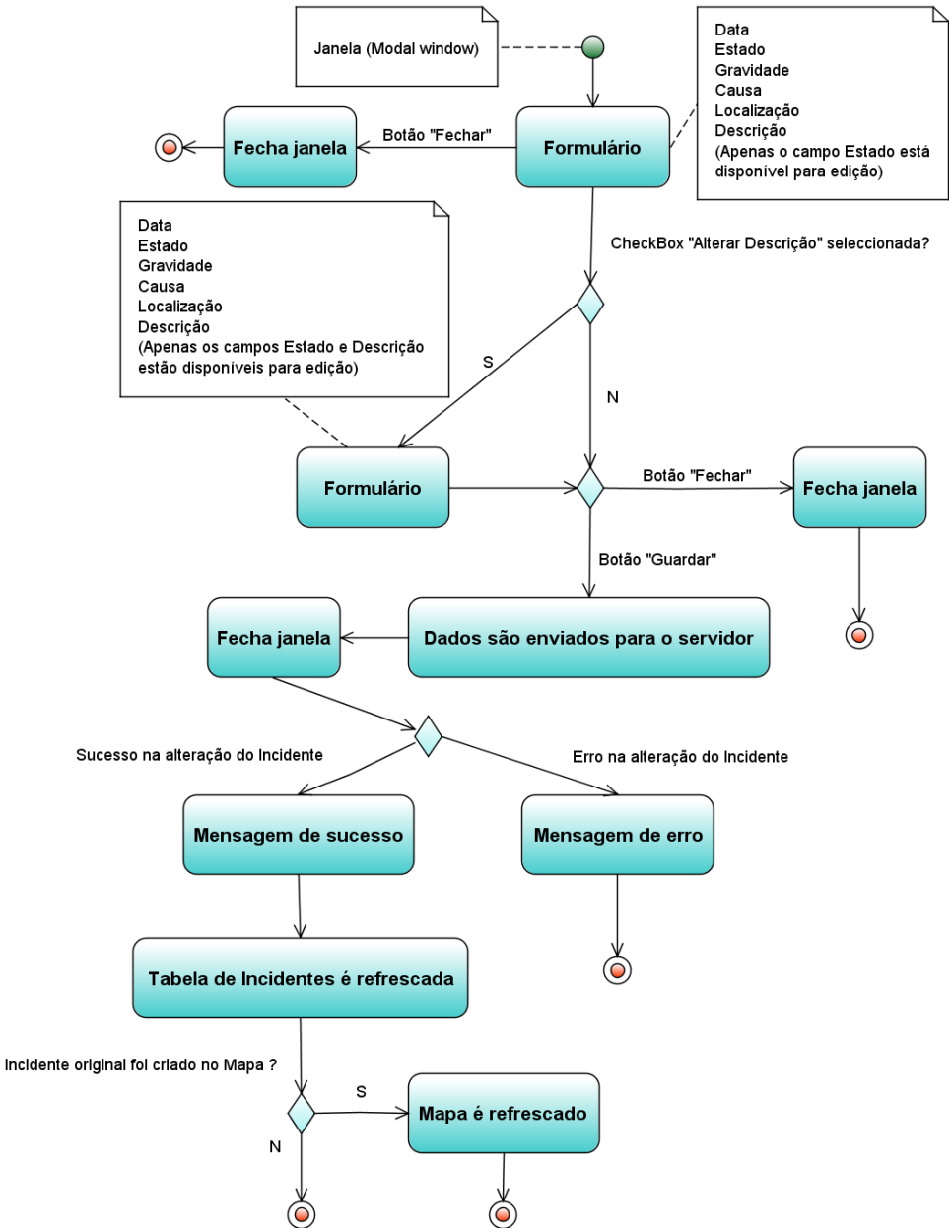


Figura 27 ***Activity Diagram*** **: Alterar Incidente**

4.6.6 Histórico de Alterações de Incidente

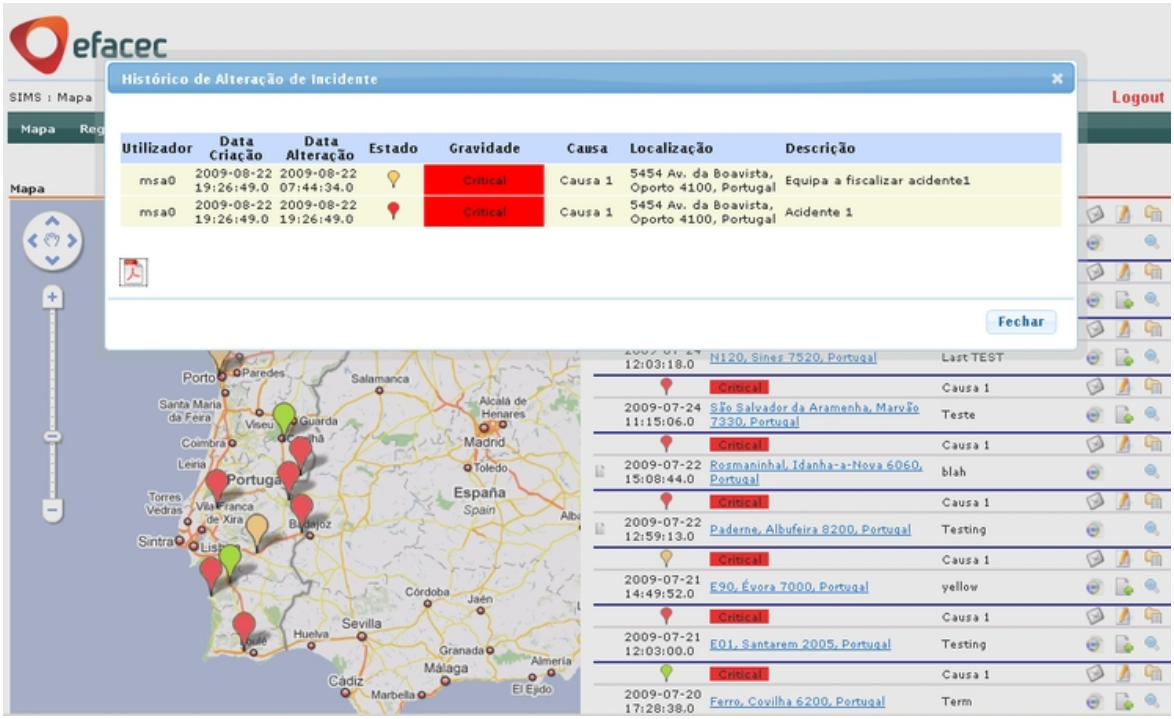


Figura 28 Histórico de Alterações de Incidente

No Histórico de Alterações de Incidente estão disponíveis todas as alterações de determinado Incidente, incluindo o utilizador e data/hora de alteração. É também possível exportar para PDF este Histórico.

Este PDF é gerado a partir de uma Servlet que para além dos habituais métodos `doGet()` e `doPost()` tem também um método com a seguinta assinatura:

```
public void makePdf (HttpServletRequest request, HttpServletResponse response,
String methodGetPost)
```

```
...

public class OutSimplePdf extends HttpServlet
{

    private static final long serialVersionUID = 2788260006560387781L;


    public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        makePdf(request, response, "GET");
    }
```

```
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        makePdf(request, response, "POST");
    }

    public void makePdf(HttpServletRequest request, HttpServletResponse response,
String methodGetPost) {

        String incId = request.getParameter("incId");

       ...

       }

...

}
```

O método `makePdf` obtém o Id do Incidente, e a partir deste pode aceder à base de dados obtendo os dados necessários para caracterizar o Histórico das alterações de Incidente usando para isso três JavaBeans: HistIncidentHandlerBean.java, IncidentHandlerBean.java e UserHandlerBean.java.

É importante referir que esta Servlet utiliza também várias classes disponibilizadas na *package* com.lowagie.text que faz parte de uma biblioteca *open source* que possibilita a conversão Java para PDF.

O princípio de criação de um documento PDF é mostrado a seguir:

```
public void makePdf(HttpServletRequest request, HttpServletResponse response,
String methodGetPost) {

...

            Document document = new Document();
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            PdfWriter.getInstance(document, baos);
            document.open();

...

}
```

Para a criação de uma tabela com oito colunas foi usado o seguinte extracto de código:

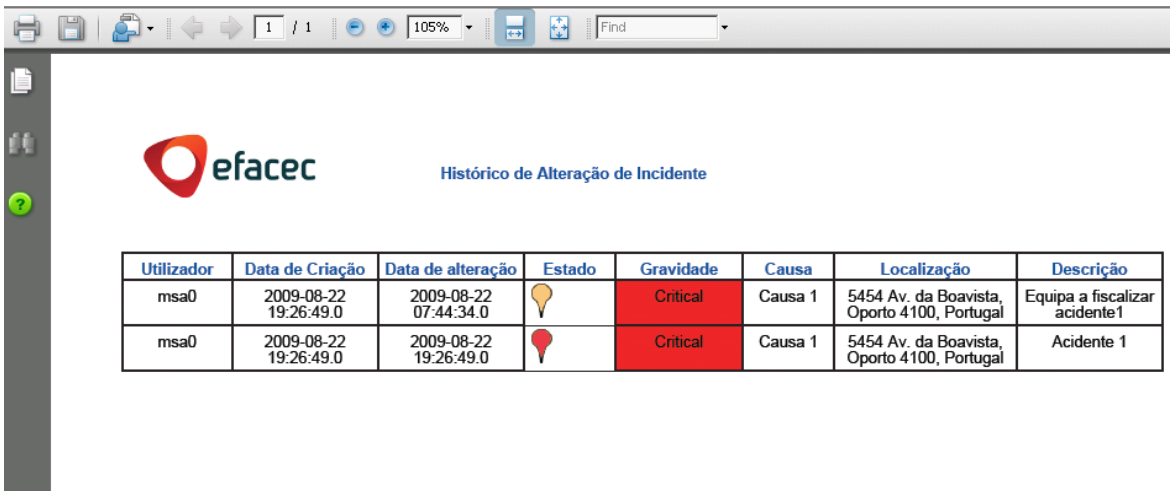
```
public void makePdf(HttpServletRequest request, HttpServletResponse response,
String methodGetPost) {

        ...

            int NumColumns = 8;

            PdfPTable datatable = new PdfPTable(NumColumns);

            int headerwidths[] = {6, 8, 8, 5, 7, 5, 10, 8}; // percentage

            datatable.setWidths(headerwidths);

            datatable.setWidthPercentage(100); // percentage

            datatable.getDefaultCell().setPadding(3);

            datatable.getDefaultCell().setBorderWidth(1);

            datatable.getDefaultCell().setHorizontalAlignment(Element.ALIGN_CENTE
R);



            datatable.addCell(new
Phrase("Utilizador",FontFactory.getFont("Helvetica", 8, Font.BOLD, new
Color(0x00, 0x52,0xa5))));

...
            datatable.addCell(new
Phrase("Descrição",FontFactory.getFont("Helvetica", 8, Font.BOLD, new Color(0x00,
0x52,0xa5))));



            datatable.setHeaderRows(1); // this is the end of the table header

            datatable.getDefaultCell().setBorderWidth(1);


...

}
```

Para popular a tabela é usado um ciclo `for`

```
public void makePdf(HttpServletRequest request, HttpServletResponse response,
String methodGetPost) {

       ...

            HistIncidentHandlerBean dataHist = new HistIncidentHandlerBean();
            UserHandlerBean userBean = new UserHandlerBean();
            java.util.Vector records = new java.util.Vector();
```

```
            records = dataHist.getHist(incId);


            estado = 0;
            int causeId = 0;
            int sev = 0;
            String sevS = "";
            String cellColor = "";
            String login="";
            String loc="";

            for (int i = 0; i < records.size(); i++)
            {

                HistIncidentHandlerBean myhistincbean = (HistIncidentHandlerBean)
records.elementAt(i);

                login=myhistincbean.getLogin();

                datatable.addCell(new
Phrase(userBean.getUNFromId(login),FontFactory.getFont("Helvetica", 8,
Font.NORMAL, new Color(0x00, 0x00,0x00))));

  ...
                datatable.addCell(new
Phrase(myhistincbean.getDescricao(),FontFactory.getFont("Helvetica", 8,
Font.NORMAL, new Color(0x00, 0x00,0x00))));

            }

...


}
```

O resultado obtido pode ser visto na figura seguinte:



Utilizador	Data de Criação	Data de alteração	Estado	Gravidade	Causa	Localização	Descrição
msa0	2009-08-22 19:26:49.0	2009-08-22 07:44:34.0		Critical	Causa 1	5454 Av. da Boavista, Oporto 4100, Portugal	Equipa a fiscalizar acidente1
msa0	2009-08-22 19:26:49.0	2009-08-22 19:26:49.0		Critical	Causa 1	5454 Av. da Boavista, Oporto 4100, Portugal	Acidente 1

Figura 29 **Exportação para PDF (Hist. de alterações de Incidente)**

4.6.7 Criar Nota

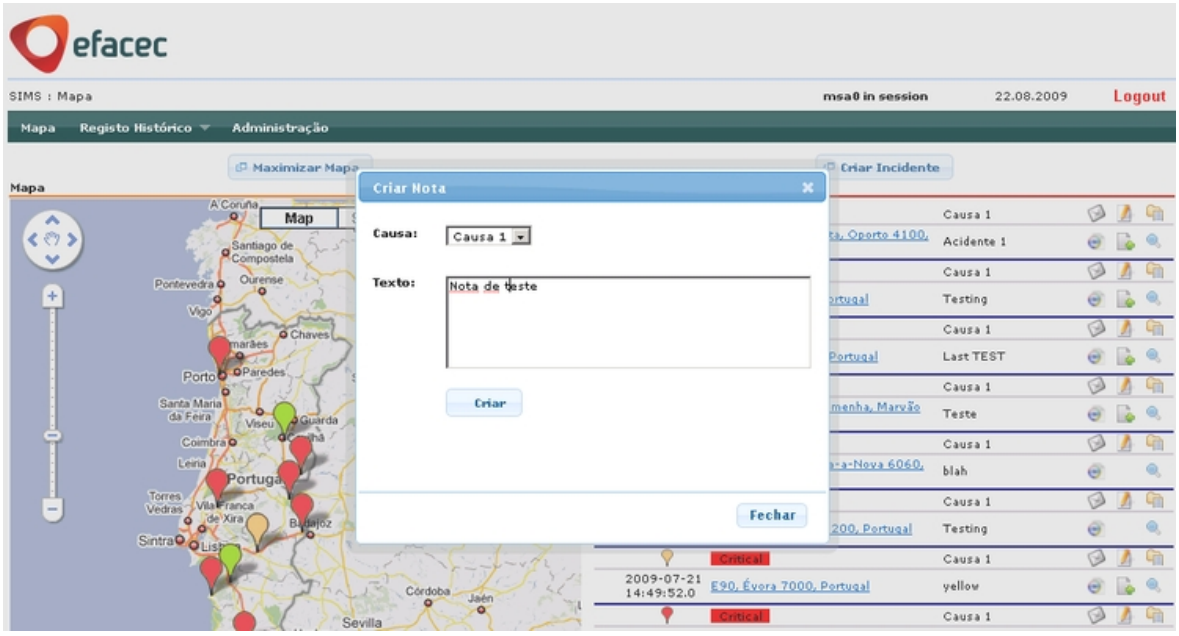


Figura 30 Criar Nota

Esta funcionalidade permite associar um Nota a determinado Incidente. O formulário que o utilizador preenche é constituído pelos seguintes campos:

- Causa - Combobox preenchida a partir dos dados da tabela T_INCIDENT_NOTE_CAUSES, representa a causa para a criação da nota;
- Texto – Texto livre

Quando o formulário é submetido com sucesso, a tabela de Incidentes é refrescada, aparecendo do lado esquerdo o ícone que assinala a presença de Nota.

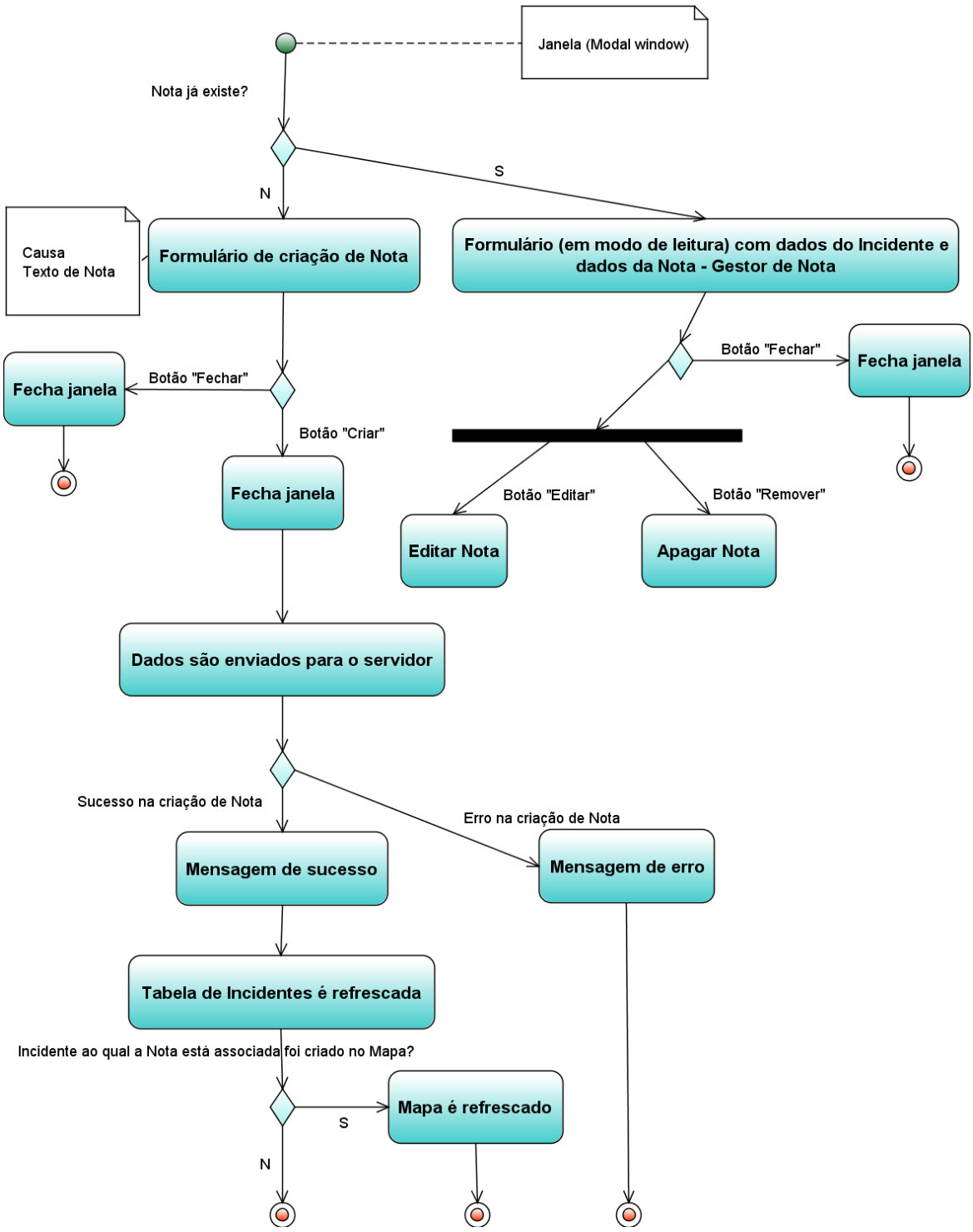


Figura 31 ***Activity Diagram*** **: Criar Nota**

4.6.8 Gestor de Nota

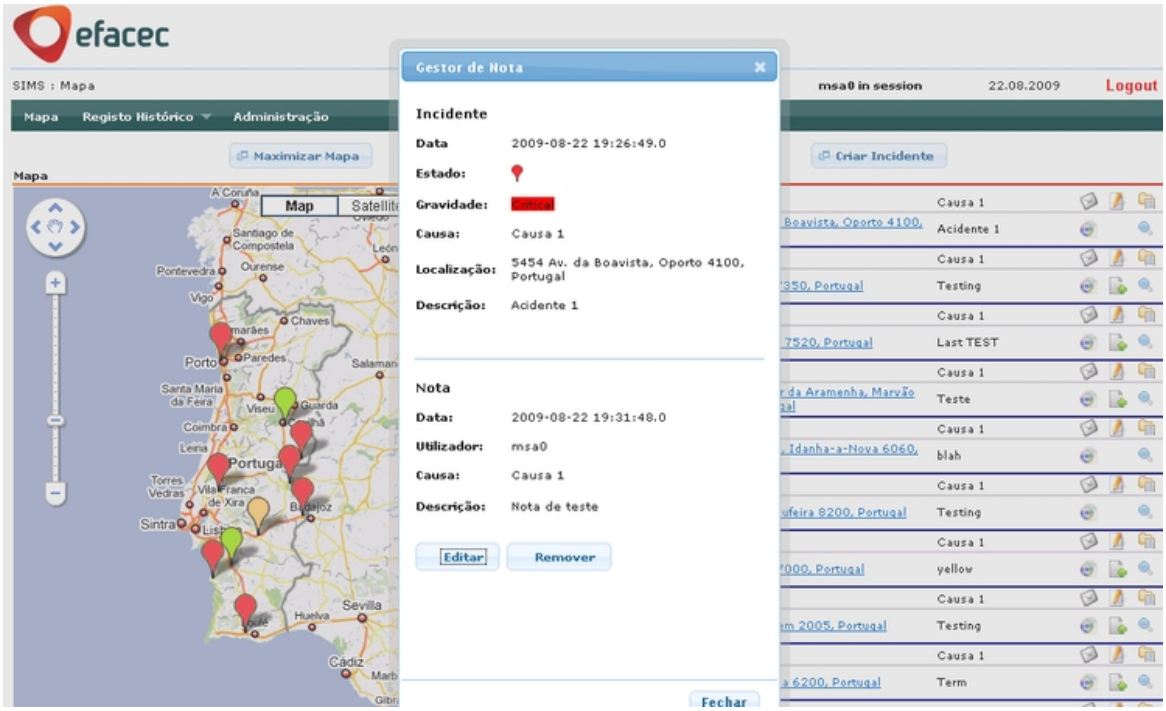


Figura 32 Gestor de Nota

O Gestor de Nota permite como o nome indica, gerir Nota associada a um determinado Incidente. Assim, a partir desta funcionalidade é possível Editar Nota e Remover Nota.

Para aceder a qualquer uma destas duas acções é aberta uma janela com os dados do incidente na parte superior e os dados da nota na parte inferior. São no fim disponibilizados os botões Editar e Remover.

A edição da nota consiste apenas em acrescentar texto, não sendo possível alterar qualquer dado criado originalmente. Sempre que é aberto o formulário os textos anteriores estão sempre visíveis.

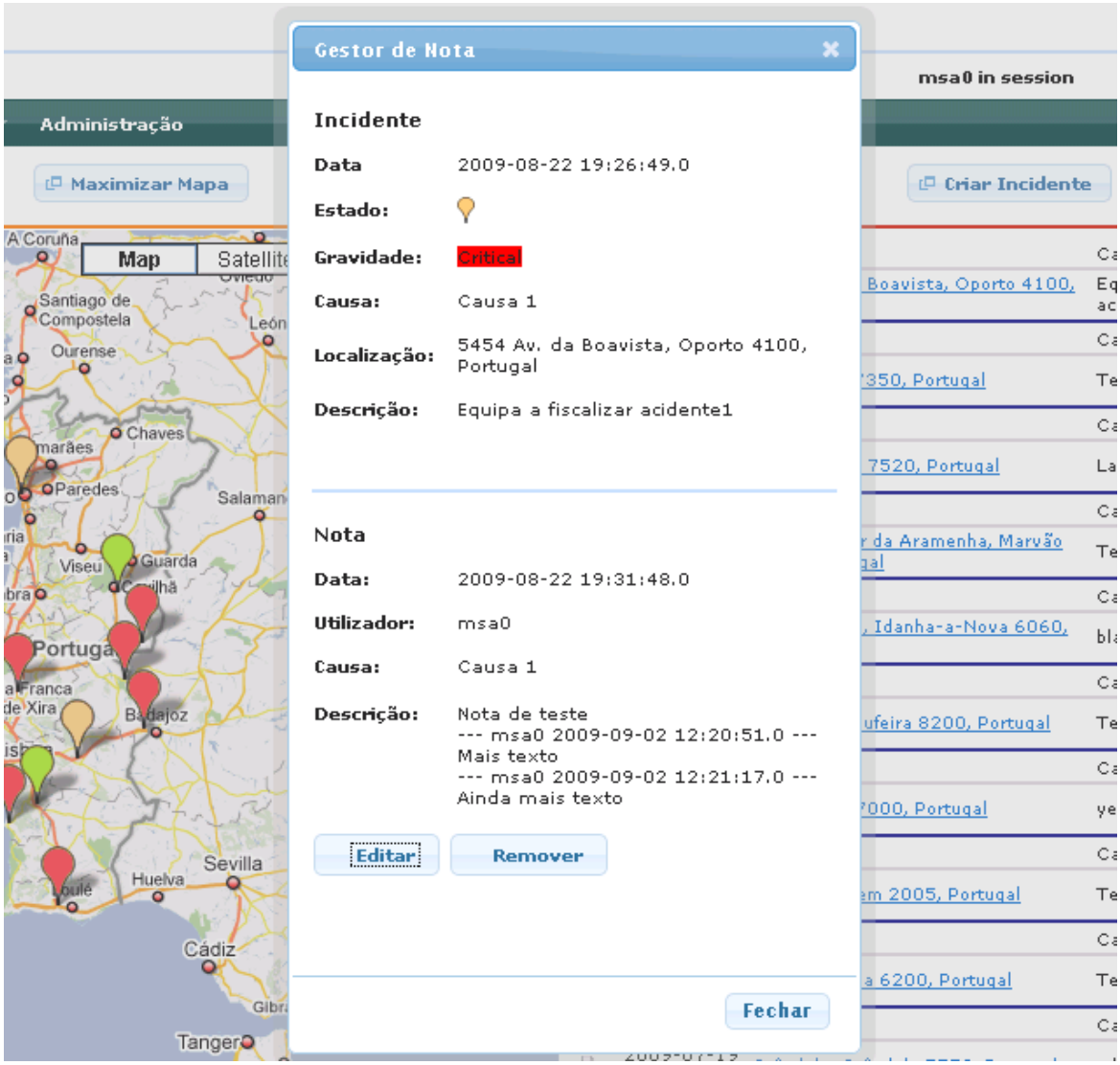


Figura 33 Gestor de Nota (depois da acção Editar Nota)

4.6.9 Anexar ficheiro

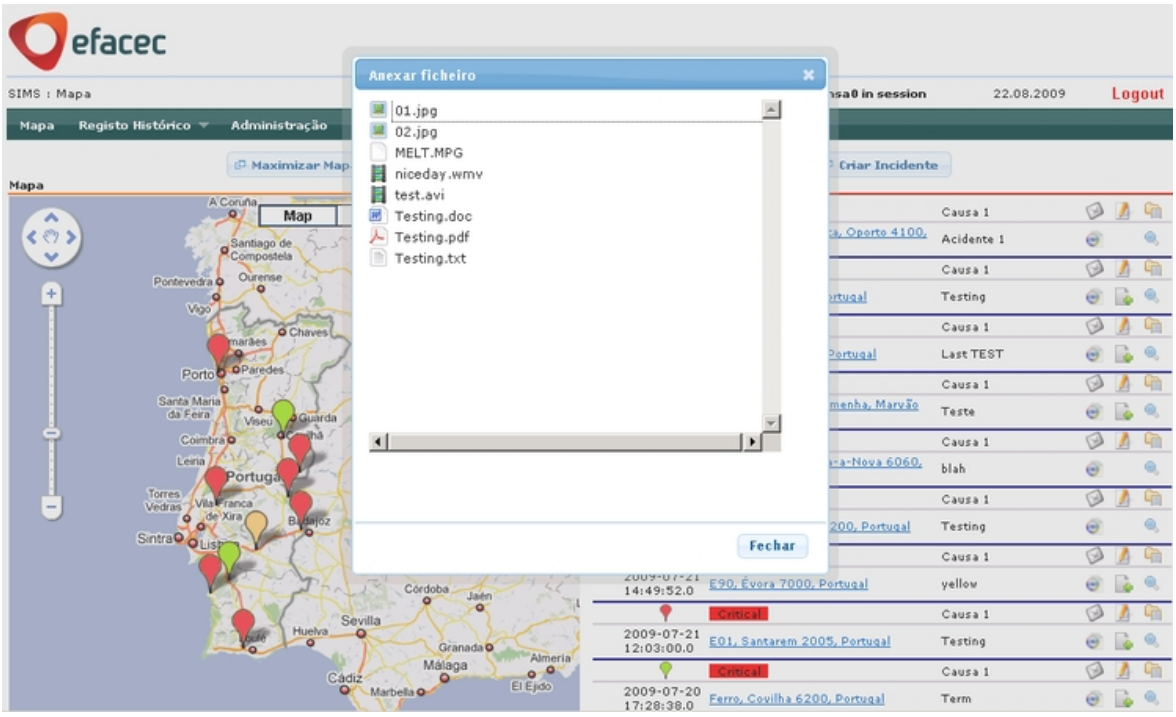


Figura 34 Anexar ficheiro

É possível associar ficheiros a determinado Incidente. Estes ficheiros encontram-se num repositório de um servidor Apache e cuja raiz está parametrizada no ficheiro de propriedades sims.properties, carregado na entrada da aplicação.

O *link* para estes ficheiros fica depois disponível na Visualização de Incidente e no Gestor de Nota, sendo assim possível visualiza-los quando pretendido.

4.6.10 Enviar alerta (*email*)

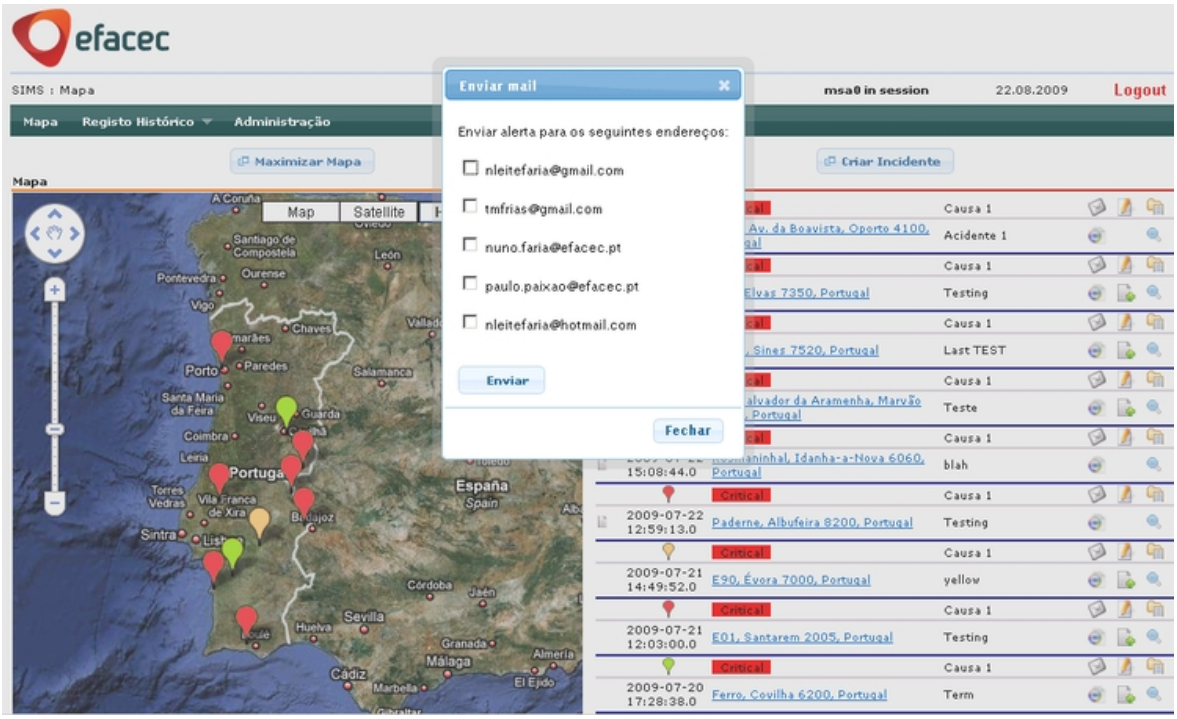


Figura 35 Enviar alerta (*email*)

Esta funcionalidade permite o envio de alertas por *email* com o conteúdo de um Incidente. É possível enviar um alerta apenas para um destinatário ou simultaneamente para vários. Os endereços estão definidos na parametrização do utilizador que é feita através do módulo de Administração.

Este envio de alertas foi desenvolvido usando a tecnologia JavaMail. Novamente, de forma a tornar a implementação da aplicação mais simples e sem necessidade de alteração de código fonte, a definição dos dados necessários ao uso desta tecnologia (como o servidor de mail) é feita no ficheiro de propriedades já referido em pontos anteriores.

Como pode ser visto na fig. 35, o *email* de alerta pode ser enviado para um ou mais endereços. O utilizador deve seleccionar o(s) endereço(s) para onde pretende enviar o *email*.

```
...


public class SendMailHelper
{

	private static SendMailHelper ivSendMailHelper = null;

...

	public static SendMailHelper getInstance()
	{
		if(ivSendMailHelper == null)
		{
			ivSendMailHelper = new SendMailHelper();
		}

		return ivSendMailHelper;
	}


	private SendMailHelper()
	{

		mailHostName = SimsProperties.getInstance().getMailHostname();
		mailHostPort = SimsProperties.getInstance().getMailPort();
		mailSendUser = SimsProperties.getInstance().getMailSendUser();
		useAuthentication =
SimsProperties.getInstance().getMailUseAuthentication();
		if(useAuthentication)
		{
			mailUser = SimsProperties.getInstance().getMailUser();
			mailPwd = SimsProperties.getInstance().getMailPassword();

		}

	}

	public int sendMail(String[] to,String subject, String content)
	{

		MailHelper m = new MailHelper(mailHostName,
Integer.parseInt(mailHostPort),mailSendUser ,to, subject, content, mailUser,
mailPwd, useAuthentication, null);


	  int i = m.sM();


	  return i;
	}

}
```

Os endereços seleccionados serão transformados num array de Strings (`String[] to`) que será um dos argumentos do método `public int sendMail(String[] to,String subject,`

`String content)`. Os outros argumentos são o subject e o content que serão preenchidos depois de serem obtidos os dados relativos a um Incidente da base de dados.

Este método será invocado no servidor depois de ser submetido o formulário "Enviar mail". O método retorna um inteiro que servirá para diferenciar os casos em que o *email* é enviado com sucesso, dos casos em que exista erro e consequentemente o *email* não seja enviado. Nesta altura e através de um mecanismo de AJAX similar ao apresentado nos casos anteriores, o cliente obterá uma resposta do servidor e exibirá a mensagem de sucesso ou erro conforme os casos.

```
...

public class MailHelper
{

...
       public MailHelper (String smtpHost, int smtpPort, String from, String[]
to,String subject, String content, String user, String pass, boolean
usarAutenticacao)
    {
...

       }
       public MailHelper (String smtpHost, int smtpPort, String from, String[]
to,String subject, String content, String user, String pass, boolean
usarAutenticacao, File attach)
    {
              ...
       }

       public int sM ()
       {
              java.util.Properties props = new java.util.Properties();

              props.put("mail.smtp.host", getSmtpHost());
              props.put("mail.smtp.port", getSmtpPort());
              if (ivUsarAutenticacao)
                     props.put("mail.smtp.auth","true");
              else
                     props.put("mail.smtp.auth","false");

              Session session = Session.getDefaultInstance(props,null);
              Message msg = new MimeMessage(session);

              try
              {
                     msg.setFrom(new InternetAddress(getFrom()));
                     InternetAddress[] addr=new InternetAddress [getTo().length];

                     for(int i=0; i < getTo().length; i++  )
                            addr[i] = new InternetAddress(getTo()[i]);

                     msg.setRecipients( Message.RecipientType.TO,addr);
                     msg.setSubject(getSubject());

                     MimeMultipart mp = new MimeMultipart();
                     MimeBodyPart text = new MimeBodyPart();
                     text.setDisposition(Part.INLINE);
```

```
                    text.setContent(getContent(), "text/html");
                    mp.addBodyPart(text);

                    if(getAttach() != null)
                    {
                           MimeBodyPart file_part = new MimeBodyPart();
                           File file = getAttach();
                           FileDataSource fds = new FileDataSource(file);
                           DataHandler dh = new DataHandler(fds);
                           file_part.setFileName(file.getName());
                           file_part.setDisposition(Part.ATTACHMENT);
                           file_part.setDescription("Attached file: " +
file.getName());
                           file_part.setDataHandler(dh);
                           mp.addBodyPart(file_part);
                    }
                    msg.setContent(mp);

                    Transport tr = session.getTransport("smtp");
                           if (ivUsarAutenticacao)
                                  tr.connect(getSmtpHost(), new
Integer(getSmtpPort()).intValue(), getUser(),
SimsProperties.getInstance().getMailPassword());
                           else
                                  tr.connect();

                    if(tr.isConnected())
                           tr.sendMessage(msg, msg.getAllRecipients());
             }
        catch (AddressException e)
        {
           return 0;
         }
        catch (MessagingException e)
        {
             e.printStackTrace();
              return 0;
         }

        return 1;

       }

       ...
}
```

O resultado do envio do alerta pode ser visualizado na fig. 36.

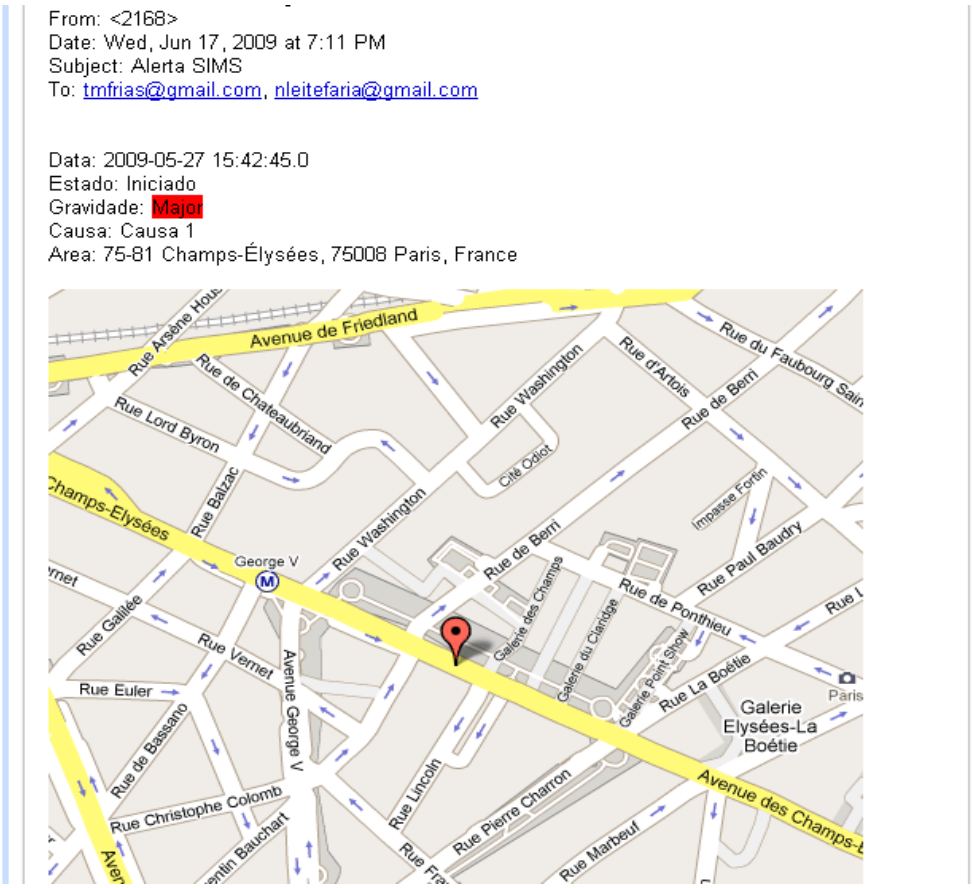


Figura 36 ***Email* de Alerta**

O campo subject é preenchido com o texto "Alerta SIMS". No corpo do *email* aparecem os campos que caracterizam um Incidente e também um mapa estático centrado no ponto de criação do Incidente com um marcador a sinalizar o último estado do Incidente em questão.

4.6.11 Registo Histórico de Incidentes

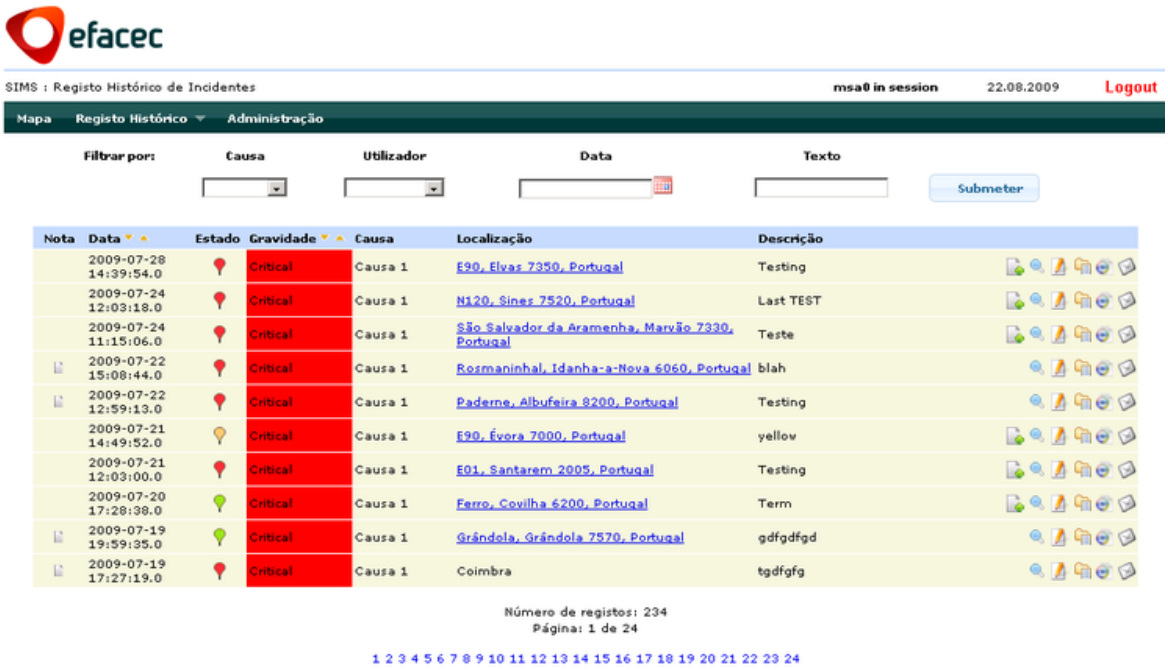


Figura 37 Registo Histórico de Incidentes

Nesta página é apresentada uma tabela que representa o Histórico dos Incidentes criados e um formulário que permite aplicar filtros à tabela.

A tabela é constituída pelas seguintes colunas:

- Data de criação
- Estado - - Iniciado, - Em execução, - Terminado
- Gravidade
- Causa
- Localização – Equivalente ao mapa inicial, no caso de o incidente ter sido criado a partir do mapa, o utilizador tem disponível um *link* a partir do qual é aberta uma janela com um mapa estático centrado na localização em questão.
- Descrição

Na secção inferior são disponibilizados ligações para as páginas de dados que constituem a tabela, permitindo ao utilizador um acesso mais imediato a um determinado conjunto de dados sem ter quer percorrer várias páginas. Este tipo de paginação tem ainda a particularidade de usar a funcionalidade de *paging* da tecnologia *AJAX*, isto é, de forma assíncrona, os dados de uma determinada página só são carregados quando o pedido é efectuado. Numa aplicação como esta, que trabalha com um número muito elevado de registos, este tipo de paginação tornou-se essencial.

É ainda possível ordenar a tabela (novamente, de forma assíncrona) por:

- Data (ascendente e descendente)
- Gravidade (ascendente e descendente)

Os filtros que podem ser aplicados são os seguintes:

- Causa (de Nota)
- Utilizador (que criou a Nota)
- Data (de criação da Nota)
- Texto (de Nota)

De igual forma os resultados da aplicação de filtros são carregados de forma assíncrona.

Como na tabela da página inicial, esta é actualizada (assincronamente) sempre que um Incindente é criado/alterado.

Os ícones do lado direito da tabela permitem aceder às funcionalidade já descritas no ponto 4.5.2.

4.6.12 Registo Histórico de Eventos

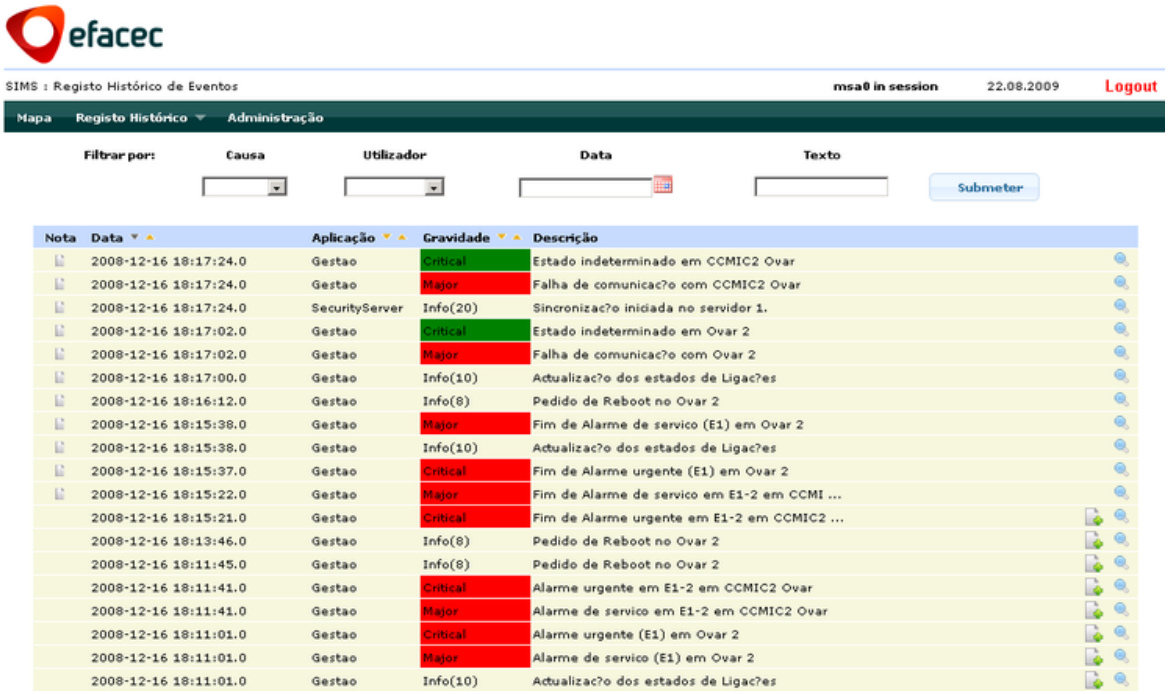


Figura 38 Registo Histórico de Eventos

Os Eventos não são criados a partir desta aplicação, sendo a base de dados alimentada por um processo de geração destes registos com origem no sistema INOSSv2 como descrito no primeiro capítulo.

Nesta página é apresentada uma tabela que representa o Histórico dos Eventos e um formulário que permite aplicar filtros à tabela.

A tabela é constituída pelos seguintes campos:

- Data
- Aplicação
- Gravidade
- Descrição

Usando a funcionalidade de paginação é possível percorrer de forma assíncrona as várias páginas que constituem a tabela.

É possível ordenar a tabela por:

- Data (ascendente e descendente)
- Aplicação (ascendente e descendente)
- Gravidade (ascendente e descendente)

Os filtros que podem ser aplicados são os seguintes:

- Causa (de Nota)
- Utilizador (que criou a Nota)
- Data (de criação da Nota)
- Texto (de Nota)

De igual forma os resultados da aplicação de filtros são carregados de forma assíncrona.

Neste caso as únicas funcionalidades disponíveis através de ícones são: Criar Nota, Gestor de Nota e Visualizar Evento.

O processo de criação e gestão de Nota é equivalente ao caso dos Incidentes mas agora aplicado aos Eventos.

4.6.13 Administração

Figura 39 Administração (Entrada)

Esta é a página de entrada no módulo de Administração da aplicação. É constituída por uma tabela onde estão listados os utilizadores do INOSSv2 e sobre os quais se podem definir parametrizações específicas para esta aplicação.

As acções disponíveis são:

- Visualizar dados
- Criar dados
- Editar dados
- Eliminar dados

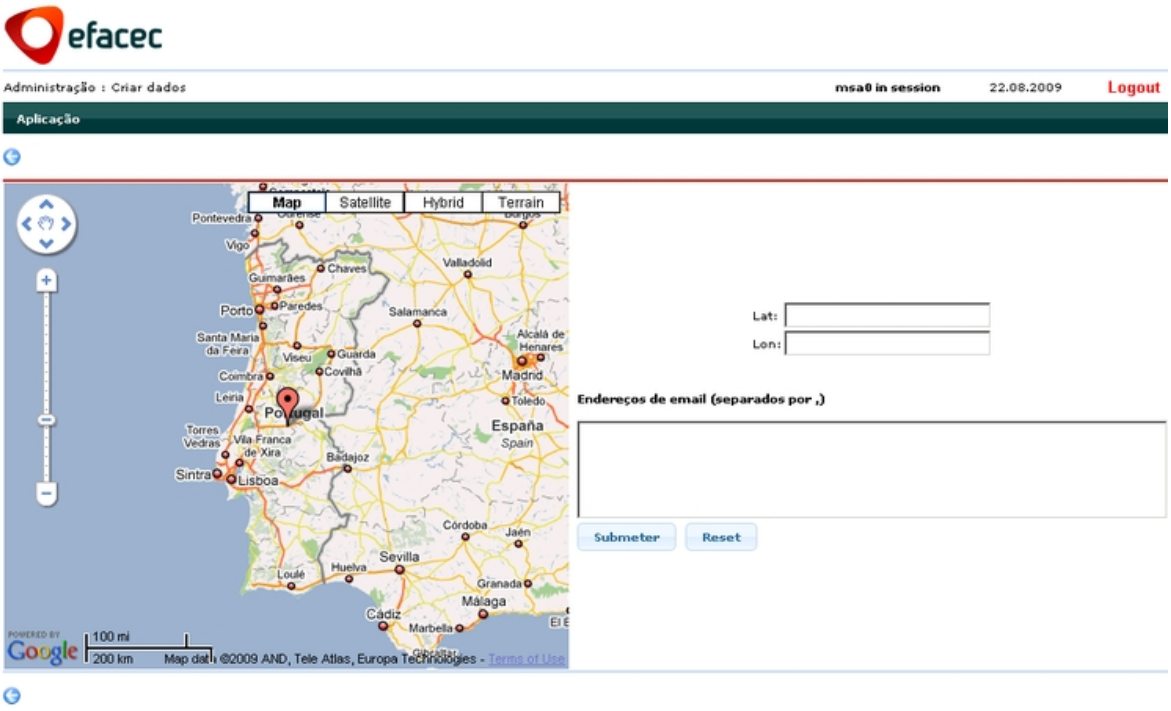


Figura 40 Criar Dados

Os dados que podem ser definidos para um utilizador são o Ponto Geográfico de Inicialização do Mapa (usado na acção Reset Map da página inicial da aplicação) – Lat e Lon - e Destinatários de Alertas – Endereços de *email*.

Para definir o Ponto de Inicialização o administrador deve marcar um ponto no Mapa clicando com o botão esquerdo do rato. As coordenada geográficas (latitude e longitude) ficam automaticamente preenchidas nas respectivas *input boxes*.

A parametrização dos destinatários é feita com a indicação dos vários endereços de email separados por vírgulas. Estes endereços serão usados para preencher de forma dinâmica as *check boxes* no pedido de envio de alertas pelo utilizador.

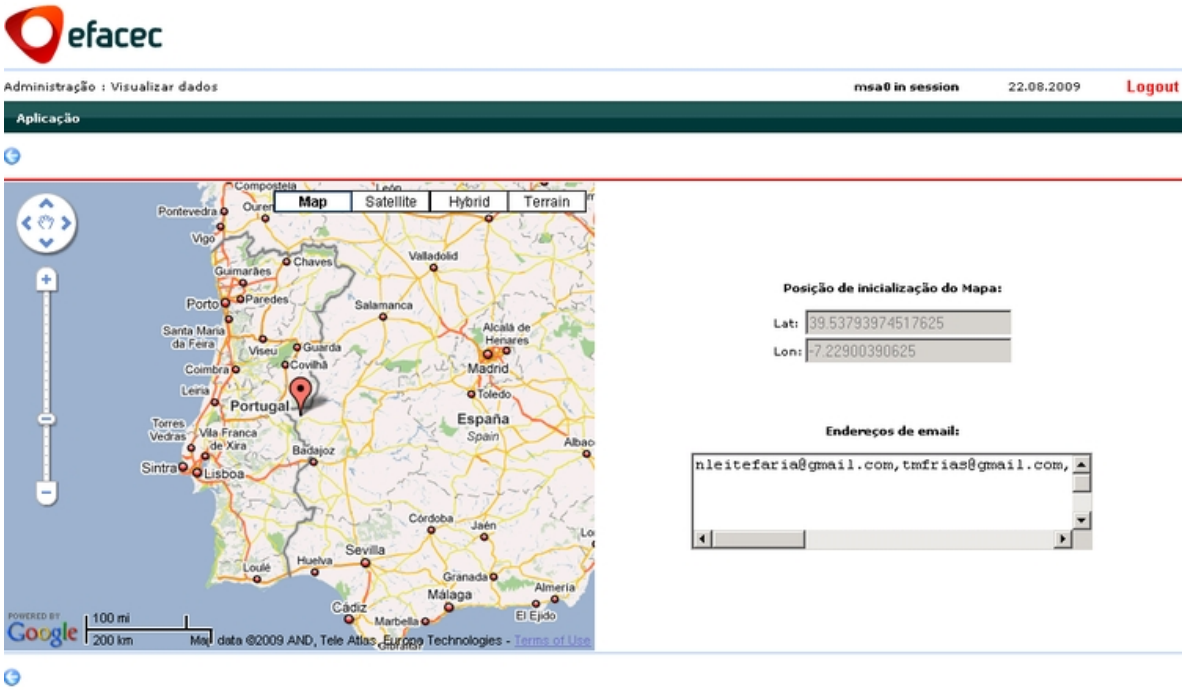


Figura 41 Visualizar dados

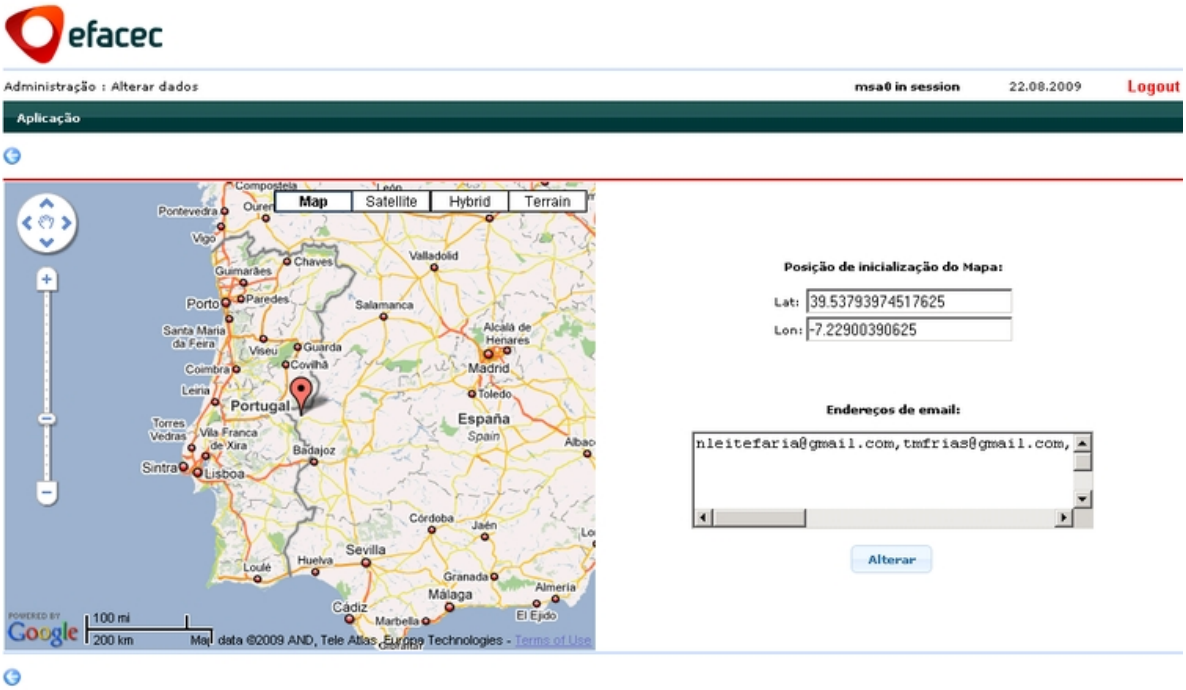


Figura 42 Editar dados

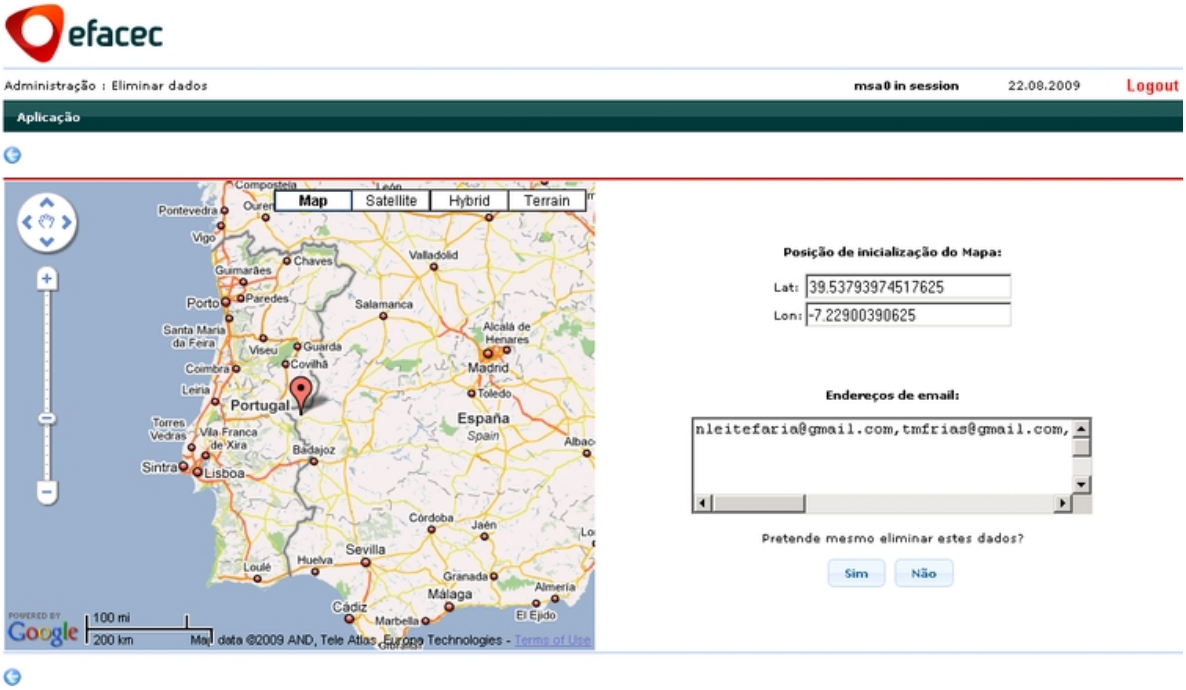


Figura 43 Eliminar dados

4.7 Testes

De forma a testar a fiabilidade e usabilidade da aplicação desenvolvida foram definidos quatro âmbitos de realização de testes:

a) Portabilidade da Aplicação com implementação de raiz numa nova máquina;

b) Funcionalidade da Aplicação (com base nos *Use Cases*);

c) Compatibilidade com os principais *browsers* existentes no mercado (Mozilla, Google Chrome e Internet Explorer 6, 7, e 8) ;

d) Testes de Carga.

De seguida descrevem-se as acções realizadas para cada uma das alíneas anteriores e respectivos resultados.

a) Portabilidade da Aplicação

Foi criado um ambiente de testes num servidor virtual instalado numa máquina da EFACEC. Neste servidor (Red Hat) existia um Tomcat, um servidor Apache e um servidor de base de dados. A aplicação foi instalada no Tomcat através da funcionalidade nativa de *deploy* existente no Tomcat Manager (ficheiro do tipo .war). A base de dados foi criada através da execução de *scripts* gerados a partir do servidor de desenvolvimento.

Depois de este ambiente estar implementado e terem sido realizadas as alterações necessárias ao ficheiro de propriedades (sims.properties), verificou-se a primeira instalação com sucesso da aplicação num novo ambiente. Deve ainda referir-se que o ambiente de desenvolvimento foi feito sobre Windows XP, tendo-se validado também a componente multiplataforma que se pretendia para a aplicação.

Foi com ligação a este servidor que foram realizados os restantes testes.

b) Funcionalidade da Aplicação

Como a aplicação é maioritariamente constituída por formulários, numa primeira fase a realização dos testes centrou-se em todos os *Use Cases* que usam formulários:

- Criar Incidente (Mapa)
- Criar Incidente (a partir de Botão)
- Alterar estado de Incidente
- Criar Nota
- Editar Nota

Os testes seguiram de uma forma normal e sem erros até ao momento que foram introduzidos caracteres aleatoriamente num campo de texto livre provocando o desaparecimento dos marcadores do Mapa e a não actualização das tabelas.

Rapidamente se percebeu que a falha ocorreu devido à introdução de determinados caracteres (>, <, &, %). Estes são caracteres reservados da linguagem XML, linguagem que é utilizada no Google Maps API para geração dos marcadores do Mapa. Concluiu-se

portanto que estes caracteres não poderiam ser inseridos no campos dos formulários, ou se fossem, a aplicação deveria ter a capacidade de os substituir pela sua *entity reference*:

Caracter	*Entity*
&	&amp
<	&lt
>	&gt
"	&quot
'	&#39

Tabela 8 ***Character entity references***

Como alguns dos caracteres anteriores são também inválidos em SQL, e são pouco relevantes para o texto dos campos dos formulários a preencher, optou-se pelo uso de expressões regulares para validação do texto. Desta forma fica assegurado que só os caracteres desejados são enviados para o servidor.

A validação dos campos é feita usando a seguinte função JavaScript:

```
function validaCampo(campo)
{

   var regex = /[^a-zA-Z\d_~!?|@-\^\.\s]/i;

   var campo2 = "";

   campo2 =campo2.concat(campo);

   while (campo2.match( regex ))
   {
          return false;
   }

   return true;
}
```

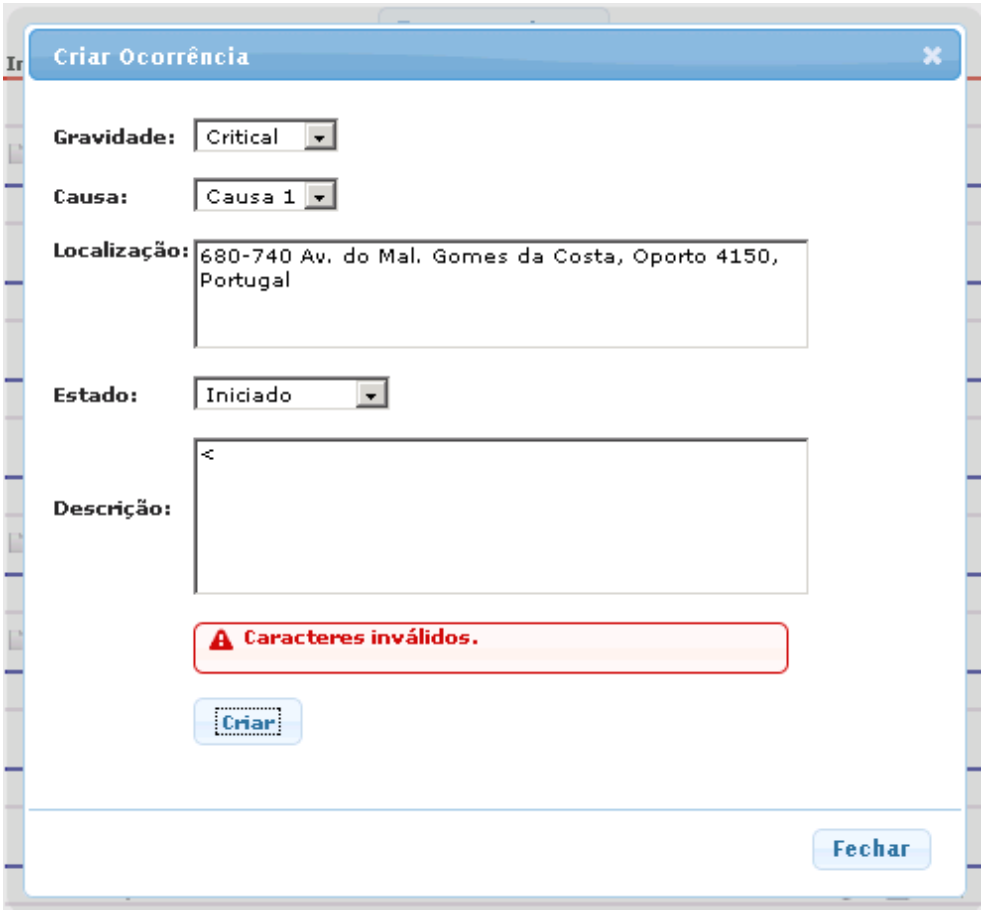


Figura 44 Validação de campos

Com esta alteração todos os formulários passaram a ter o comportamento esperado.

Numa segunda fase os testes foram feitos sobre os restantes *Use Cases* (p.ex. acções sobre tabelas, envios de alertas, etc). Nestes casos não foi detectada qualquer questão tendo-se concluído os testes funcionais com uma taxa de sucesso muito satisfatória.

c) Compatibilidade de *browsers*

O teste de compatibilidade de *browsers* foi realizado utilizando os *browsers*:

- Mozilla Firefox 3
- Google Chrome
- Internet Explorer 6, 7, 8

Estes testes revelaram falhas graves com todas as versões do Internet Explorer sendo que as falhas diminuíam à medida que a versão do Internet Explorer evoluía. Depois de algum estudo concluiu-se que estas falhas se devem ao facto de o Internet Explorer não cumprir as normas definidas pelo W3C (World Wide Web Consortium). Segundo as melhores práticas de desenvolvimento de aplicações *Web*, estas normas devem ser cumpridas e foi seguindo a orientação destas que a aplicação foi desenvolvida.

No Mozilla Firefox e no Google Chrome (que cumprem os standards W3C) os testes foram realizados sem que se detectasse qualquer tipo de problema ou falha que colocasse em questão as tecnologias escolhidas.

Foi necessário tomar uma decisão a partir dos resultados destes testes. As opções seriam refazer a aplicação sem garantia de que todos os problemas seriam resolvidos uniformemente para os três *browsers* (e levando a uma provável perda de funcionalidade) ou excluir o Internet Explorer como cliente válido para a aplicação.

Optou-se por restringir o uso da aplicação apenas aos *browsers* Firefox e Google Chrome tendo sido implementado um *script* em todas as páginas da aplicação que verifica qual o *browser* que está a ser utilizado, enviando uma mensagem no caso do cliente em questão tratar-se do Internet Explorer e redireccionando para o *download* de *browsers* compatíveis. Esta foi uma decisão tomada em conjunto com o Gestor de Projecto/Orientador.

d) Testes de Carga

Tendo em consideração que a aplicação deverá ser utilizada por um pequeno número de utilizadores não se sentiu a necessidade de realizar testes de carga muito intensos. Optou-se por colocar quatro utilizadores em máquinas clientes a utilizarem simultaneamente a aplicação (ao invés de geração de múltiplos processos que simulam a utilização da aplicação). Os resultados foram satisfatórios já que não se notou qualquer deficiência ou degradação de *performance* em qualquer das aplicações clientes.

Como garantia de sucesso total, a aplicação ficou instalada na EFACEC e será alvo de testes finais a serem realizados por programadores do departamento de ID da unidade de Transportes da EFACEC.

5 Conclusões e Desenvolvimentos futuros

5.1 Conclusões

A realização deste trabalho possibilitou a aquisição de novos conhecimentos em tecnologias emergentes (AJAX) e a consolidação de conhecimentos no que toca ao Desenvolvimento de Aplicações *Web*. Foi igualmente importante o estudo da tecnologia Oracle, dado que esta não foi abordada durante a componente curricular do presente Mestrado.

Os objectivos delineados no início do projecto foram os habituais para projectos de desenvolvimento de *software* e foram concluídos com sucesso:

- Estudo de tecnologias relevantes para o desenvolvimento de sistemas de gestão de incidentes: nesta fase fez-se uma escolha das tecnologias que à partida seriam mais eficazes para realizar tal tarefa, optando-se sempre por minimizar os custos. Dado que a principal solução da EFACEC (INOSS) é baseada na tecnologia Java e Oracle, ficou claro logo desde o início que estas seriam duas das tecnologias a adoptar. A escolha das outras tecnologias que requeriam inovação e um estudo mais aprofundado dado que são emergentes (*frameworks* de JavaScript e AJAX) foi feita baseada em artigos publicados em livros e sítios *Web* da especialidade.

- Levantamento de requisitos: neste ponto foi essencial a colaboração do Orientador/Gestor de Projecto. Em várias reuniões vocacionadas especificamente para esta tarefa conseguiu-se construir uma estrutura que englobasse todos os requisitos. É importante realçar que estas reuniões aconteceram com uma periodicidade quinzenal durante todo o projecto, e por vezes foi necessário ajustar as necessidades ao que era efectivamente possível realizar. Esta fase comprovou ser uma das mais importantes no ciclo de desenvolvimento de *software* pois acabou por servir de base para todas as outras que se lhe seguiram.

● Elaboração de especificação: neste ponto foi fundamental o uso de UML. O uso desta linguagem de modelação permitiu caracterizar toda a aplicação facilitando assim a fase de desenvolvimento. A tarefa de modelação revelou-se por vezes complexa e trabalhosa, mas os resultados finais forneceram uma base importante para a fase de desenvolvimento.

● Implementação de acordo com a especificação: depois de os pontos anteriores estarem terminados tornou-se bastante simples o desenvolvimento da aplicação. A excepção a este facto foi a tecnologia AJAX já que este foi o primeiro contacto com a tecnologia e como tal requereu um tempo de aprendizagem elevado.

● Testes: os testes realizados foram muito satisfatórios, já que a aplicação funcionou da forma esperada.

5.2 Desenvolvimentos futuros

Após teste e validação da aplicação desenvolvida foi possível definir alguns pontos que podiam ser melhorados ou funcionalidades que podiam ser acrescentadas:

● Sempre que é criado um Incidente deverá ser criado um Evento correspondente.

● A tabela anexa ao Mapa representa na aplicação os últimos dez Incidentes criados. Pretende-se que a partir de um mecanismo de paginação se possa trazer para essa tabela outros conjuntos de dez Incidentes. Quando isso acontecer, deverão aparecer dinamicamente no Mapa os marcadores correspondentes.

● Visualização no Mapa de Incidentes de acordo com o seu estado (mecanismo de filtro no mapa).

● Possibilidade de novas parametrizações na área de administração, por exemplo: as causas de um Incidente e as localizações (pré-definidas) de um Incidente.

● Aumento progressivo de funcionalidades do Mapa dado, que a API Google Maps de versão para versão acrescenta normalmente serviços que podem ter bastante interesse para este projecto.

Referências Documentais

[BAL01] BALES, Donald — *Java Programming with Oracle JDBC*. *O'Reilly* Media, Inc.. ISBN-13: 978-0-596-00088-2.

[BIB08] BIBEAULT, Bear; KATZ, Yeahuda — *jQuery in Action*. Manning Publications Co.. ISBN: 1933988355.

[BRO06] BROWN, Martin C. — *Hacking Google Maps and Google Earth.* John Wiley & Sons, Inc.. ISBN: 9780471790099.

[CHA07] CHAFFER, Jonathan; SWEDBERG, Karl — *Learning jQuery: Better Interaction Design and Web Development with Simple JavaScript Techniques*. Packt Publishing. ISBN-13: 978-1-84719-250-9.

[CRA07] CRANE, Dave; BIBEAULT, Bear; LOCKE, Tom — *Prototype and Scriptaculous in Action*. Manning Publications Co.. ISBN-13: 978-1-933988-03-0.

[CRD07] CRANE, Dave; BIBEAULT, Bear; LOCKE, Tom — *Ajax in Practice*. Manning Publications Co.. ISBN: 1-932394-99-0

[FAL03] FALKNER, Jayson; JONES, Kevin W. — *Servlets and JavaServer Pages™: The J2EE™ Technology Web Tier*. *Addison-Wesley* Professional. ISBN-13: 978-0-321-13649-7

[GAL02] GALLARDO, David — *Java™ Oracle® Database Development*. Prentice Hall. ISBN-13: 978-0-13-046218-3.

[GAR] GARRETT, Jesse James — *Ajax: A New Approach to Web Applications.* http://adaptivepath.com/ideas/essays/archives/000385.php

[GEA01] GEARY, David M. — *Advanced JavaServer Pages™*. Prentice Hall. ISBN-13: 978-0-13-030704-0.

[GIB06] GIBSON, Rich; ERLE Schuyler — *Google Maps Hacks*. Oreilly & Associates Inc.. ISBN-13: 978-0-596-10161-9.

[GMD] *Google Maps API Developer's Guide*. http://code.google.com/apis/maps/documentation/index.html

[GMR] *Google Maps API Reference*.

http://code.google.com/apis/maps/documentation/reference.html

[GOO01] GOODMAN, Danny — *JavaScript Bible*.

John Wiley & Sons, Inc.. ISBN-13: 978-0764531880.

[GRE04] GREENWALD, Rick; STACKOWIAK, Robert; STERN, Jonathan — *Oracle Essentials, Third Edition*.

O'Reilly Media. ISBN 10: 0-596-00585-7.

[HAL01] HALL, Marty — *More Servlets and JavaServer Pages™*.

Prentice Hall. ISBN-13: 978-0-13-067614-6.

[HAL03] HALL, Marty; BROWN, Larry — *Core Servlets and JavaServer Pages™: Volume 1: Core Technologies, 2nd Edition*.

Prentice Hall. ISBN-13: 978-0-13-009229-8.

[HAR04] Harold, Marty — Java Network Programming.

O'Reilly Media. ISBN 10: 0-596-00721-3.

[ITE] *iText by Example*.

http://itextdocs.lowagie.com/tutorial/

[JET] *The J2EE 1.4 Tutorial*.

http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

[LON04] LONEY, Kevin — *Oracle Database 10g: The Complete Reference*.

Oracle Press. Web ISBN: 0-07-225351-7.

[MAL02] MALHEIRO, Maria Benedita — *Desenvolvimento de Aplicações Distribuídas, Texto de Apoio*.

[MAL04] MALHEIRO, Maria Benedita — *Desenvolvimento de Aplicações Web, Acetatos*.

[PER04] PERRONE, Paul J.; Chaganti, Venkata S.R. "Krishna" R.; Schwenk ,Tom — *J2EE Developer's Handbook*.

Sams Publishing. ISBN-13: 978-0-672-32348-5.

[ROD05] RODRIGUES, António — *Oracle 10g e 9i Para Profissionais.*

FCA - Editora Informática. ISBN: 9789727223442.

[SIL07] SILVA, Robson Soares — *Oracle Database 10g Express Edition - Guia de Instalação, Configuração e Administração com Implementação PL/SQL Relacional e Objeto-Relacional*.

Erica. ISBN: 9788536501628.

[WEA04] WEAVER, James L.; MUKHAR, Kevin; CRUME, Jim — *Beginning J2EE 1.4: From Novice to Professional*.

Apress. ISBN13: 978-1-59059-341-7.

[WITEN] http://en.wikipedia.org/wiki/Apache_Tomcat

[WITPT] http://pt.wikipedia.org/wiki/Apache_Tomcat