Luís Filipe Lobo <ellobo@ipb.pt>

Plataforma de Sumários do Instituto Politécnico de Bragança

Escola Superior de Tecnologia e de Gestão

Outubro 2010

Plataforma de Sumários do Instituto Politécnico de Bragança

Dissertação apresentada ao Instituto Politécnico de Bragança para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Sistemas de Informação, sob a supervisão de Prof. Doutor Rui Pedro Lopes.

Luís Filipe Lobo <ellobo@ipb.pt>

Outubro 2010

Prefácio

O presente documento descreve os aspectos e decisões tomadas no decorrer do desenvolvimento da Plataforma de sumários do IPB. A plataforma enquadra-se num conjunto de ferramentas desenvolvidas e em desenvolvimento no IPB para apoio às actividades lectivas das várias unidades orgânicas.

A passagem de uma metodologia banal de registo de sumários para uma solução auxiliada electronicamente motivou o desenvolvimento de uma solução à medida das várias unidades orgânicas. Foram ouvidas as partes intervenientes, com o intuito de colmatar o maior número de necessidades possível.

A importância de uma plataforma deste tipo reside na vantagem das novas tecnologias em detrimento do registo tradicional em papel, quer pela normalização dos procedimentos de registo quer pelo aumento da sua disponibilidade.

Cada vez mais, procedimentos quotidianos são migrados para soluções interactivas de grande disponibilidade, permitindo aos utilizadores concentrar as suas energias na actividade lectiva. Este tipo de plataforma representa ainda uma mais valia em termos de gestão, na medida em que permite a intervenção no processo de forma activa, bem como a obtenção de dados estatísticos. Com o registo sumários é reunida informação crucial que suporta a instituição no dimensionamento de turmas, alocação de recursos e na elaboração de horários.

Agradecimentos

Agradeço a todos aqueles que de forma directa ou indirecta contribuíram para a realização desta dissertação. Com especial destaque para o meu orientador, Prof. Doutor Rui Pedro Lopes, pela sua ininterrupta disponibilidade, animosidade e pela sua capacidade inventiva que em muito contribuiu para o desenvolvimento da plataforma.

Agradeço também à a equipa do Centro de Desenvolvimento do IPB pelo suporte dado na construção da plataforma de sumários e à equipa do Centro de Comunicações pela disponibilidade e empenho na sua colocação em produção.

Dedico em especial às minhas três fontes de energia e inspiração incansáveis Andreia e Ana Luís e ao recentemente chegado Diogo.

A todos os que abriram a sua mente e partilharam comigo a sua opinião e experiência no campo, para eles o meu apreço e amizade.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

O registo de sumários e presenças é uma ferramenta importante no exercício das actividades lectivas em qualquer instituição de ensino. Este registo permite a organização ao nível curricular e ao nível do controlo de assiduidades.

Este tipo de aplicações requerem uma análise prévia dos requisitos, com o objectivo de tomar decisões acerca de tecnologia e ferramentas a utilizar no seu desenvolvimento.

A Web como canal de comunicação com os utilizadores, atrai cada vez mais a atenção dos *developers* pela facilidade de desenvolvimento e pela quantidade de ferramentas e suporte ao nível da comunidade. As aplicações vocacionadas para a Web apresentam um nível de disponibilidade que outras formas de desenvolvimento, tais como as aplicações *desktop*, não alcançam.

A disponibilização da aplicação dos sumários via web irá alcançar, com certeza, uma maior quantidade de utilizadores, pela facilidade actual de acesso à Internet. Garantindo ainda que actualizações futuras da aplicação serão transparentes para os utilizadores.

A integração da aplicação dos sumários com os serviços instalados é vital, uma vez que parte da informação necessária ao seu funcionamento é baseada nestes. O software de horários deverá implementar os mecanismos necessários à propagação de alterações e actualizações nas suas dependências.

A prototipagem de uma aplicação deste género deverá ter em mente a usabilidade e a capacidade de desenvolvimento futuro, pela utilização de tecnologias actuais e largamente suportadas. A informação deverá ser mantida de forma segura, garantido, no entanto, a rapidez de acesso.

O desenvolvimento multi-camada confere à aplicação capacidade de expansão e modularidade no desenvolvimento.

A constante reavaliação da aplicação no seu contexto, lavará ao seu crescimento em harmonia com os utilizadores e permitirá que a aplicação se torne cada vez mais intuitiva, eficaz e alternativa à prática banal do registo de sumários.

1.1 Enquadramento

O Instituto Politécnico de Bragança é uma instituição de ensino superior composta por cinco escolas:

- Escola Superior de Educação de Bragança
- Escola Superior de Tecnologia e Gestão de Bragança
- Escola Superior Agrária de Bragança
- Escola Superior de Saúde de Bragança
- Escola Superior de Comunicação, Administração e Turismo

A instituição tem um corpo docente de perto de 500 docentes e à volta de 140 funcionários. A contagem de alunos desde a abertura da instituição vai em perto de 27000. No ano de 2010/2011 encontram-se activas perto de 6800 inscrições.

São leccionadas no ano actual (2010) mais de 3000 disciplinas distribuídas por perto de 110 cursos.

A forma como o registo de sumários era feito à data do começo do projecto depende um pouco da unidade orgânica em questão. No entanto, uma grande parte delas regista os sumários e presenças de forma manual em papel. O processamento destes documentos é depois feito por cada uma das secretarias onde são validadas as assiduidades e registados os conteúdos leccionados.

1.2 Objectivos

Pretende-se, com este trabalho, implementar uma Plataforma de Sumários que vá de encontro às exigências em torno da elaboração de sumários do Instituto Politécnico de Bragança.

A plataforma deverá satisfazer as necessidades das várias unidades orgânicas da instituição, tendo em conta as especificidades de cada uma, ainda que mantendo um nível de abstracção suficiente que permita a sua utilização de um modo geral.

Devem ser criadas as estruturas necessárias de apoio à gestão e manutenção da plataforma, permitindo às várias unidades orgânicas intervir de forma contextualizada no processo.

Pretende-se uma aplicação que se adeque à tecnologia disponível na instituição, ainda que, aplicando técnicas e utilizando ferramentas actuais de forma a permitir a sua actualização.

1.3 Estrutura do Documento

O capítulo 2 do documento contextualiza o leitor no cenário das Aplicações Web. É feita uma introdução ao que representam e seu enquadramento no cenário actual de construção de aplicações. É feita uma descrição do protocolo `HTTP` em termos gerais, por forma a inteirar o utilizador dos mecanismos de comunicação de uma aplicação deste género. Ainda no capítulo 2, é feito um levantamento de tecnologia disponível para elaboração da plataforma de sumários.

No capítulo 3 é feito um levantamento dos requisitos da aplicação e as necessidades de integração com as aplicações e entidades existentes na instituição. Seguidamente é feita uma análise da tecnologia, métodos e protótipo a utilizar no desenvolvimento dos vários componentes da aplicação.

No capítulo 4 é detalhado o caminho seguido para o desenvolvimento da aplicação, seguido do cenário actual da aplicação de sumários e alguns dados estatísticos.

O último capítulo reflecte sobre a dissertação em termos gerais e propõe algumas possibilidades para desenvolvimento futuro da plataforma.

Capítulo 2

Aplicações Web

A Internet, como canal de comunicação, reuniu a unanimidade dos utilizadores de todos os tipos, devido essencialmente à sua facilidade de utilização e ao seu constante crescimento e expansão, de tal forma que o seu alcance toma hoje proporções globais.

Com este tipo de alcance é de esperar que fornecedores de serviços e informação se tenham voltado para este canal. Hoje em dia, grande parte da informação trocada por várias entidades/empresas como o estado, o seu próprio sistema financeiro ou até mesmo o serviço de saúde de um país inteiro é veiculada por canais estabelecidos, quer entre entidades ou entre estas e o utilizador final.

As Aplicações Web tal como as conhecemos, são normalmente o canal de comunicação entre quem presta os serviços e os utilizadores. Existe, no entanto, uma componente não visível ao utilizador final que é igualmente importante. Na maior parte dos casos a parte visível de uma Aplicação Web é apenas a montra de uma infraestrutura bastante maior.

Na maioria dos casos, por detrás da interface visível existe toda uma estrutura de gestão que, nos dias de hoje, funciona tendencialmente sobre a Internet. Estas estruturas de gestão desafiam o conceito de localização física de quem presta os serviços, permitindo que a mesma entidade esteja distribuída por vários locais, mantendo, ainda assim, sua gestão disponível às varias partes.

Apesar destas estarem disponíveis das mais variadas formas, a forma mais comum de utilizar uma aplicação web passa pela utilização de um browser apontando para um endereço web. Este endereço identifica o prestador de serviços de forma unívoca em toda a Internet.

A troca de conteúdos feita entre a aplicação web e o browser do utilizador é relativamente simples. Após o pedido para o endereço do servidor do prestador de serviços, é gerado um documento com o conteúdo solicitado, documento este que é descarregado e disposto no browser segundo um conjunto de especificações definidas por quem o concebeu.

Estas especificações são criadas recorrendo a um conjunto de linguagens de programação/marcação que definem o comportamento e o aspecto do conteúdo.

O conteúdo consiste de um conjunto de objectos que, definidos com recurso à linguagem de marcação `HTML` (HyperText Markup Language) compõem a árvore DOM (*Document Object Model*) do documento. Cada nó desta árvore consiste de uma *tag* que define o conteúdo do objecto (texto, imagem, vídeo, ...) bem como os atributos do seu comportamento e aspecto.

Apesar da capacidade de definir o aspecto de todo um documento em linha com o código `HTML`, uma estrutura adicional permite definir estilos para cada nó da árvore ou para conjuntos de nós. O conjunto destes estilos é chamado de Folha de Estilos do documento, ou `CSS` (Cascading Style Sheet). Cada elemento da árvore pode ser afectado por um conjunto de atributos de estilo.

Estas duas estruturas são geradas do lado do servidor e descarregadas para o cliente que posteriormente as interpreta e desenha o documento. A possibilidade de manipular o documento do lado do cliente é também possível, a fim de tornar a interface com o utilizador mais rica e interactiva. A árvore `DOM` do documento pode ser manipulada com recurso à linguagem de programação `JavaScript`. Esta permite a injecção e alteração de conteúdo da árvore, manipulação de eventos gerados pelo cliente (click, arrastar, ...) bem como das folhas de estilo associadas ao documento.

Mais recentemente (primeira década de 2000) surgiu uma técnica denominada de AJAX (Asynchronous JavaScript and XML) que permite obter dados do servidor de forma assíncrona. Esta técnica representa claramente uma mais valia quando se pretende desenvolver aplicações interactivas. Em vez recarregar o documento com novo conteúdo de cada vez que uma acção é desencadeada, é feito o carregamento selectivo de zonas (nós/ramos `DOM`) do documento.

A comunicação AJAX é feita utilizando código JavaScript com recurso a um objecto disponível na maioria dos browsers actuais - XMLHttpRequest [van Kesteren, 2009].

A troca de dados entre o servidor e o cliente nos pedidos AJAX é feita utilizando código XML (Extensible Markup Language) [Maler et al., 2004]. No entanto, este código é utilizado apenas pelo browser para encapsular os pedidos, de forma que a escolha da protocolo dos dados transferidos fica à inteira disposição de quem desenvolve a aplicação.

Na maioria das aplicações os dados transferidos estão revestidos de algum significado ou representam um conjunto de propriedades a alterar/adicionar. Em 1999 surgiu um formato de intercâmbio de dados, não dependente de qualquer linguagem de programação, que pela sua simplicidade, foi acolhido pela comunidade e é hoje em dia utilizado numa grande parte das Aplicações WEB que recorrem ao AJAX no seu funcionamento.

O JSON (JavaScript Object Notation - `http://www.json.org/`) [Crockford, 2006] é

um formato de intercâmbio de dados de fácil interpretação e *human-readable*. Consiste num conjunto de pares $nome : valor$ que representam normalmente um objecto ou estrutura de dados:

$$objecto = \{nome_0 : valor_0, \ldots, nome_n : valor_n\}$$

Esta forma de intercâmbio de dados é suportada de forma nativa no código JavaScript, o que torna as comunicações servidor $\leftrightarrow$ cliente transparentes para quem está a desenvolver a aplicação web.

Graças a algumas iniciativas *open-source*, são inúmeras as *frameworks* disponíveis para desenvolvimento de aplicações web. No entanto, algumas destacam-se pela sua aceitação junto da comunidade, pela sua facilidade de utilização e abstracção ao código HTML/CSS/JavaScript subjacente e pela quantidade de funcionalidade oferecida.

2.1 O protocolo HTTP

Nas aplicações web actuais (HTML4) a comunicação com o servidor é assíncrona[1], o que implica que cada pedido da aplicação cliente tem uma correspondência com uma resposta do lado do servidor sem que qualquer tipo de comunicação persistente seja estabelecida.

Os pedidos são normalmente destinados a um determinado recurso, identificado de forma única por uma URL (Uniform Resource Locator) [Berners-Lee et al., 1994]:

`https://apps.ipb.pt:8181/sumarios/`

Os principais componentes de uma URL podem ser identificados no texto em cima, sendo:

- `https`: O *scheme*, que neste caso representa a comunicação HTTPS (Hypertext Transfer Protocol Secure) [Rescorla, 2000]
- `apps.ipb.pt`: O nome do domínio Internet em que se encontra este recurso
- `8181`: O número do porto utilizado para o protocolo definido no *scheme*
- `/sumarios/`: O caminho (*path*) do recurso no servidor

Os pedidos HTTP podem ser formulados com recurso a vários métodos especificados no RFC2616 [Fielding et al., 1999], resultado em respostas diferentes com conteúdos diferentes. Interessa no entanto destacar aqueles com que um *developer* comum tem contacto:

- `GET` - Pede a apresentação de um determinado recurso

[1] Na versão 5 do HTML está prevista a comunicação bidireccional por intermédio de Web Sockets - [Hickson, 2009]

- `HEAD` - Semelhante ao pedido `GET` mas obtém apenas meta-informação acerca do recurso sem que este seja apresentado
- `POST` - Submete dados para processamento (ex.: formulário `HTML`) pelo recurso especificado. Este pedido pode resultar na apresentação de novos conteúdos/recursos

2.2 Frameworks de Desenvolvimento Web

A utilização de frameworks permite agilizar o desenvolvimento das Aplicações WEB ao nível do grafismo e interacção com o utilizador do lado do cliente (browser). Ao nível do servidor existe também um conjunto de tecnologias que interagem de formas eficazes com estas frameworks.

Uma das grandes vantagens, e a que motiva com certeza mais utilizadores para a utilização de frameworks do lado do cliente, tem a ver com a forma incoerente como os browsers interpretam o código HTML/JavaScript/CSS. Todos os *web developers* se enfrentam com a necessidade de desenvolver código diferente orientado para cada browser. De facto, este tipo de frameworks lidam com estas discrepâncias de forma transparente para o *developer*. Esta faceta é normalmente denominada de *cross-browser support*.

jQuery

A quase totalidade das frameworks JavaScript apresenta características muito semelhantes, pelo que não faria sentido enumerar cada uma delas. De entre as que fizeram parte do processo de análise da aplicação de sumários, a framework `jQuery`[1] destacou-se pela facilidade de utilização e funcionalidades oferecidas.

O jQuery é uma Biblioteca de funcionalidade JavaScript que simplifica a manipulação de documentos HTML, tratamento de eventos, animação e interacções AJAX.

O acesso aos objectos disponíveis no documento (`DOM`) é feito com recurso a um engenho criado para o efeito. O Sizzle[2] é um sub-projecto do `jQuery` que permite, com base numa linguagem de consulta (filtros), obter referências para um ou mais elementos da árvore `DOM`.

Um dos pontos fortes da iniciativa `jQuery` reside na sua abertura à comunidade e às suas contribuições, que tornaram a framework rica de funcionalidade e de melhoramentos.

Apesar de todas as facilidades e melhoramentos que o `jQuery` introduz ao desenvolvimento deste tipo de aplicações do lado do cliente, uma parte importante de qualquer aplicação web reside do lado do servidor. O `jQuery` não prevê qualquer tipo de facilidade para o desenvolvimento de código do lado do servidor, no que respeita à comunicação

[1] http://jquery.com/
[2] http://sizzlejs.com/

com o cliente, pelo que toda a troca de informação tinha que ser feita com protocolos implementados, o que representa custos de desenvolvimento adicionais para a aplicação.

Servidores de Aplicações e *server-side scripting*

A decisão da forma como o conteúdo é criado do lado servidor, está revestida de uma importância vital para a aplicação web. São várias as plataformas para a criação do repositório da aplicação, no entanto, poucas são as que contêm o nível de integração e funcionalidades que apresentam as plataformas baseadas na especificação Java EE (*Java Enterprise Edition*).

O Java EE inclui um conjunto de especificações orientadas para o desenvolvimento *enterprise* abordando várias camadas. Uma das camadas, e aquela que merece destaque no âmbito das aplicações web, é a camada web (*web tier*). Nesta destaca-se a especificação das *Java Servlets* (actualmente na versão 2.5).

Uma `servlet` consiste de um elemento do lado do servidor que permite disponibilizar conteúdo com recurso à especificação do protocolo `HTTP`. No caso das *Java Servlets* todo o contexto do servidor *enterprise* é disponibilizado ao *developer* sob a forma de uma `API` (Application Programming Interface).

Uma `servlet` do lado do servidor responde, normalmente, a pedidos `HTTP`. A sua anatomia consiste realmente na implementação dos vários métodos disponíveis na especificação do protocolo `HTTP`. Na implementação de cada um dos métodos, o *developer* tem acesso à informação relacionada com o pedido, por exemplo um pedido `GET`, bem como ao canal que permite responder ao cliente que o efectuou. Reunindo assim os requisitos básicos e essenciais ao desenvolvimento dos componentes do lado do servidor.

Existem várias implementações da especificação *Java Enterprise*, quer comerciais quer gratuitas.

O servidor *GlassFish* implementa a maior parte das funcionalidades da especificação e encontra-se disponível para utilização em produção de forma gratuita. Este servidor de aplicações é desenvolvido e suportado pela Sun (actualmente Oracle).

Em volta da tecnologia Java EE, foi criado um conjunto de `APIs` que cobrem várias áreas desde a visualização, camada funcional, de dados e segurança. Esta quantidade de `APIs` e capacidade de integração é o que torna os servidores de aplicações tão atractivos.

Ao nível da visualização o `GlassFish` suporta diversas possibilidades, que vão desde a geração de páginas Web à integração de interfaces nativas, com objectivo de execução em ambientes de *desktop*. No que toca ao primeiro caso, elaboração de aplicações para execução em browser web, a norma Java EE define, como standard, a utilização de JSF (Java Server Faces[1]). Este tecnologia permite construir uma solução web utilizando, integralmente,

[1] http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

markup languages (XML e XHTML). Adicionalmente, suporta vários mecanismos de geração de código em tempo real, tornando-a a tecnologia ideal para cenários com clientes heterogéneos (dispositivos móveis, browsers, sistemas vocais, etc.). De notar que, para ambientes web, a norma prevê a utilização de `AJAX`, dispensando o carregamento integral da página.

Dado que recorre a linguagens de marcação, é a tecnologia ideal para a elaboração de páginas web por parte de designers, dado que não requer conhecimento de qualquer outra linguagem de programação ou de scripting, como Java, PHP, Python ou outras, separando, perfeitamente, a visualização do processamento.

Dada a sua maturidade, é possível encontrar bibliotecas, para livre utilização, que complementam o standard com um conjunto de componentes ricos, dando a possibilidade de criar interfaces complexas com relativamente pouco esforço.

Por outro lado, dadas as características do sistema que se pretende desenvolver e dadas as dificuldades em abarcar linguagens distintas, optou-se por tirar proveito da capacidade de integração patente nos servidores de aplicações Java EE e recorrer a APIs que suportam a elaboração de interfaces web em Java, dispensando o conhecimento de outras linguagens de marcação.

GWT - Google Web Toolkit

O Google Web Toolkit (GWT[1]) é um conjunto de ferramentas para a construção e optimização de aplicações vocacionadas para os web browsers. A framework `GWT` é gratuita e oferece transparência no desenvolvimento de aplicações para os `browsers` mais conhecidos.

A `API` permite o desenvolvimento ao nível do cliente e do servidor, ainda assim, todo o código é desenvolvido em `Java`. Do lado do cliente estão disponíveis um conjunto de componentes gráficos para a construção de interfaces. Do lado do servidor, o `GWT` oferece integração com *Java Servlets*, permitindo assim utilizar a tecnologia *Java EE* com todas as vantagens que isso pressupõe.

A comunicação entre o cliente (*browser*) o servidor (servidor *enterprise*) é feita com recurso a `RPCs` (*Remote Procedure Calls*). A tecnologia `GWT RPC` fornece ao *developer* a possibilidade de aceder a recursos localizados no servidor (`servlets`) de forma assíncrona utilizando pedidos `AJAX`.

Uma interface Java (`RemoteService`) descreve os procedimentos disponíveis do lado do servidor. Do lado do servidor uma `servlet` especial (`RemoteServiceServlet`) implementa a funcionalidade destes procedimentos.

Os principais componentes de uma aplicação `GWT` são:

[1] http://code.google.com/intl/en/webtoolkit/

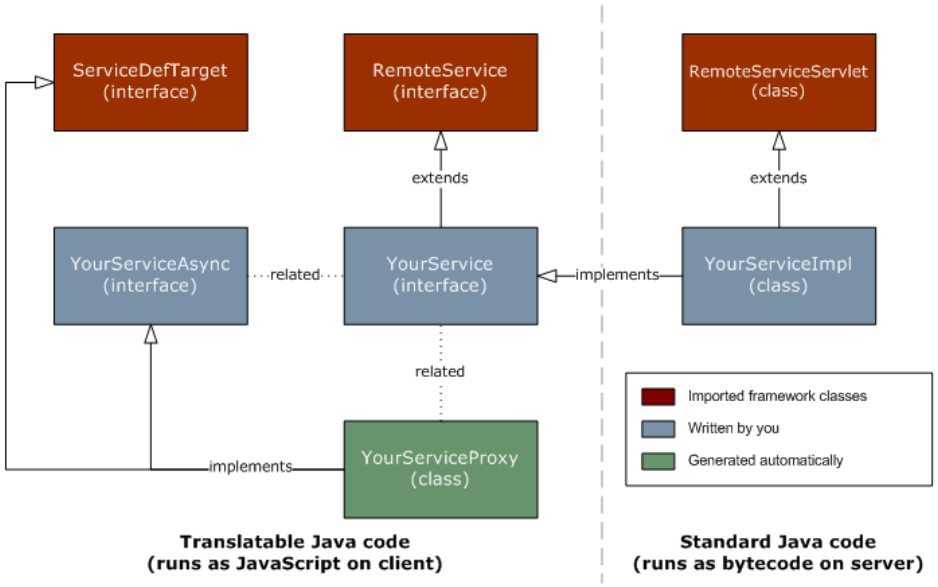


Figura 2.1: GWT RPC - *Remote Procedure Calls*

- **Ficheiro HTML**: Este ficheiro representa o ponto de entrada onde são injectadas todas as dependências do `GWT` bem como o código resultante da aplicação. Neste ficheiro pode ser ainda definido o *layout* principal da página e carregadas as folhas de estilos (`CSS`) que definem o seu aspecto.
- **Código Java Cliente**: O código do lado do cliente contém a interface com o utilizador, bem como as definições dos `RPCs` a implementar do lado do servidor.
- **Código Java Servidor**: No código do servidor são implementadas as especificações dos `RPCs` definidos do lado do cliente.

Antes de ser publicado, o projecto é compilado num pacote pronto a ser publicado no servidor de aplicações sob a forma de um recurso web. A compilação consiste na criação de código `HTML`, `JavaScript` e `CSS` correspondente ao código Java do cliente e classes Java do lado do servidor, entre elas estão as `servlets` que implementam os `RPCs` previamente especificados.

Para a construção da aplicação é utilizado código Java, com a excepção da definição do aspecto da página que é feito em `CSS`. Os pedidos `RPC` são feitos de forma transparente para o programador, utilizando os mecanismos de comunicação disponíveis no `GWT` (descritos com mais detalhe na secção 4.4).

Capítulo 3

Requisitos e Análise

A aplicação de sumários providencia, ao docente, uma forma de criar, alterar e eliminar sumários, bem como manter um registo de presenças nas aulas. Adicionalmente, prevê-se dar a possibilidade de o docente solicitar alterações pontuais de aula, bem como a criação de aulas adicionais ou de compensação.

Na perspectiva dos serviços, a plataforma dos sumários permite manter um registo exacto e actualizado de aulas, pedidos e autorizações de alterações, estatísticas de presenças, bem como um histórico de alterações de horários ao longo do semestre.

Na perspectiva da gestão, a plataforma permite obter dados úteis para alocação de espaços, elaboração e optimização de horários e acompanhamento, em tempo real, de toda a actividade lectiva.

Para dar resposta a todos estes requisitos, é necessário proceder a uma análise rigorosa, prevendo que, ao longo da sua vida, poderá haver alterações e complementos no sentido de optimizar e melhorar o seu funcionamento.

3.1 Requisitos

A aplicação de sumários providencia, na sua essência, uma forma de criar sumários tendo, como base, outros serviços em produção no IPB e funcionando em articulação com estes.

A alocação de aulas aos docentes, operação prévia à possibilidade de elaboração de sumários por parte dos docentes, é efectuada na ferramenta de elaboração de horários, em produção já há alguns anos em várias Escolas do IPB. Esta ferramenta permite, com base em restrições temporais e espaciais, elaborar os horários a serem distribuídos aos alunos, aos docentes e a serem afixados nas salas de aula.

Horários

A inserção de sumários depende do horário do docente. Estes horários não são gerados ou geridos no Software de Sumários, mas sim no Sistema de Gestão de Actividades Lectivas (descrito com mais pormenor na secção 3.2.2).

No entanto, devem ser criadas as estruturas necessárias à importação destes horários para a plataforma de sumários por forma a que a última versão dos horários esteja sempre disponível. Pretende-se que, com esta sincronização *on-demand*, no final de cada exercício (semestre), os sumários contenham armazenado o somatório das diferentes versões de horários.

A sincronização dos horários deve ser solicitada explicitamente pela comissão de horários, aquando da submissão de uma nova versão. Esta solicitação colocará um pedido de sincronização no software de sumários e notificará os responsáveis pela plataforma do referido pedido.

Juntamente com o pedido de sincronização será criado um relatório que resume o processo de geração dos eventos por parte do software de horários (GAL). O pedido será aceite ou rejeitado de forma explícita com base neste relatório.

O software de horários cria os horários sob a forma de um conjunto de eventos. Cada um destes eventos representa uma aula que pode ser leccionada por um ou mais docentes, a um ou mais cursos/disciplinas em simultâneo. Assim cada aula tem associadas as variáveis:

- **Data**: Dia em que a aula tem lugar
- **Inicio e Fim**: Hora de inicio e fim da aula para o dia designado
- **Sala**: A sala em que a aula tem lugar
- **Docentes**: Lista de docentes que leccionam a aula
- **Disciplinas**: As disciplinas associadas a esta aula (no caso de vários cursos haverá várias disciplinas)

A informação acerca das disciplinas e cursos está disponível na base de dados dos Serviços Académicos (mais detalhes na secção 3.2.1). Os cursos estão organizados por escola e plano. Cada curso pode ainda conter um conjunto de disciplinas. Assim cada item da lista de disciplinas da aula contém as variáveis:

- **Curso**: O curso a que esta disciplina pertence
- **Plano**: O plano a que o curso pertence
- **Escola**: A escola que ministra o referido curso

- **Opção**: No caso de esta disciplina fazer parte de uma opção, este campo contém a informação dessa opção
- **Turma**: Letra de A a Z que define a turma associada à disciplina
- **Tipo de Aula**: O tipo de aula associada ao evento para esta disciplina em particular segundo a norma ECTS[1]

Inserção de Sumários

A criação e edição de sumários deve ser intuitiva, abstraindo o utilizador das variáveis associadas a cada evento, como a data e hora, os dados da disciplina, curso e tipo de aula. Numa primeira instância, será mostrado apenas um campo para a inserção do texto do sumário, podendo o utilizador no entanto, alterar as demais variáveis com recurso a um formulário avançado onde estarão disponíveis um conjunto de campos adicionais, tais como:

- **Bibliografia**: A bibliografia associada ao sumário inserido[2]
- **Tipo de Aula**: É comum que o tipo de aula do sumário não corresponda ao tipo de aula original no evento, pelo que é pertinente poder alterá-lo

Os sumários devem ser inseridos por disciplina (consequentemente por curso). O utilizador deve, no entanto, pode inserir apenas sumários para uma parte das disciplinas, podendo preencher os sumários em falta mais tarde.

Adicionalmente o utilizador deve ainda poder sub-dividir o tempo do evento em vários sumários[3].

Ainda no formulário avançado, deve existir um campo para inserir o número do sumário. Na medida do possível este deverá ser sugerido com base na numeração e aulas anteriores. O número do sumário será de preenchimento facultativo, no entanto, quando preenchido deverá aparecer nas folhas de sumário.

Os sumários deverão ser editáveis durante um período de tempo antes e depois da data do evento visado. Neste período o utilizador poderá alterar todos os campos do sumário ou ainda anulá-lo. O período para edição de sumários deverá ser definido por cada entidade de acordo com as suas normas internas.

[1] European Credit Transfer and Accumulation System - http://ec.europa.eu/education/lifelong-learning-policy/doc48_en.htm

[2] Na maioria dos sumários este campo não é relevante pelo que passa para segundo plano

[3] Ex.: para a mesma aula entre as 10:00 e o 12:00 o utilizador deverá poder inserir 2 sumários, um das 10:00 às 11:00 e outro das 11:00 ao 12:00

Marcação de Presenças

Associadas a cada sumário deverá haver uma lista de presenças de alunos. O formulário de edição da lista de presenças deve permitir a pesquisa de alunos inscritos por nome de aluno ou número mecanográfico. A pesquisa deverá ser, na medida do possível, contextualizada com a escola e curso que definem o evento. A fim de prever casos particulares, este formulário deve ainda permitir a inserção de alunos não inscritos à disciplina.

Para cada evento deve haver a possibilidade de criar uma folha de presenças. Nesta folha constará uma lista dos alunos previstos para a aula. Esta lista poderá ser alterada pelo utilizador na altura de impressão, com recurso a formulário de pesquisa semelhante ao utilizado para a marcação de presenças. Deve haver ainda a possibilidade de definir a quantidade de linhas em branco na referida lista para o caso de alunos que não constam da base de dados dos serviços académicos.

A folha de presenças deverá conter um cabeçalho com informação relevante da disciplina em questão. O aspecto deve ser semelhante ao apresentado no Apêndice A.1.

Impressão

Depois de inserido o sumário, deve haver a opção de impressão, juntamente com a folha de presenças, sendo a impressão desta opcional. Deve ainda ser possível imprimir sumário e folha de presenças em folha separada. No caso de a folha de presenças não ser impressa, deve existir a possibilidade de colocar dois sumários por página, no caso de haver mais do que um sumário para o mesmo evento. A folha de sumário terá um aspecto semelhante ao apresentado no Apêndice A.2.

Pedidos de Alteração de Aulas

Para o caso das aulas ainda não sumariadas, deve existir a possibilidade de criar um pedido de alteração. Este pedido de alteração incide sobre a data e localização (sala) da aula. Para justificar o pedido de alteração, este deve fazer-se acompanhar por uma breve descrição.

Os pedidos de alteração devem ser moderados (sujeitos a aprovação) à *posteriori*, por alguém indicado para o efeito. O moderador poderá ainda alterar os valores do pedido, para ir de encontro às disponibilidades da instituição. A decisão final da moderação deve ser acompanhada por um texto justificativo.

O utilizador deve poder consultar o estado dos seus pedidos de alteração pendentes e ser informado (via e-mail) da decisão final dos moderadores.

Supervisão

Com o objectivo de permitir o acompanhamento do preenchimento dos sumários ao longo do semestre, deve ser possível a cada instituição supervisionar a inserção de sumários para cada ano/semestre, curso e disciplina. Os sumários devem ser mostrados numa lista por ordem cronológica, com a informação da data, hora e sala, bem como o docente que criou/alterou o sumário.

Criação de Aulas Adicionais

Em certas circunstâncias, é útil a criação de aulas adicionais às importadas na altura da sincronização com o GAL. Estima-se que as sincronizações com o software de horários implicam a importação de milhares de eventos, pelo que não será funcional uma nova importação de cada vez que um destes casos se verifique. Assim, o software de deverá suportar a criação de aulas adicionais.

O ecrã para a criação de aulas adicionais deve permitir a visualização das aulas actuais por sala, docente e curso, afim de evitar colisões com eventos existentes e garantir a disponibilidade de recursos e docentes. Depois da criação da aula, os responsáveis pelo horário na instituição bem como o docente associado à aula, devem ser notificados via e-mail.

Quando uma aula é criada com recurso a esta facilidade, deve ser identificada de forma diferente no horário (por exemplo, com uma cor diferente).

Para além da criação de aulas, deve ser ainda possível alterar ou remover aulas existentes.

Validação de Aulas

Deve ser criada uma funcionalidade para permitir a validação dos sumários e presenças inseridos. Esta validação permitirá verificar irregularidades no preenchimento dos sumários.

O ecrã de validação dos sumários deve permitir a visualização dos horários por sala, docente e curso.

A validação consiste de uma análise prévia do preenchimento do sumário e presenças para cada aula, seguido de uma validação que marca a aula como sumariada. Após validação, a aula fica bloqueada a alterações.

Afim de prever situações excepcionais, a anulação de uma validação previamente inserida deve ser possível. Neste caso a aula volta ao estado inicial e pode ser alterada pelo docente.

Deve existir ainda a possibilidade de notificar o docente quando um sumário de uma aula está em falta. A notificação será feita por e-mail, contendo informação que permita ao docente identificar a aula em questão.

Para o caso de uma justificação de falta ser apresentada para um sumário em falta, deve ser possível marcar a aula como válida, no entanto, a validação deve fazer-se acompanhar da informação da justificação. A aula justificada deve ainda aparecer identificada de forma diferente das aulas validadas de forma normal.

Painel de Administração

Com o objectivo de facilitar algumas tarefas de manutenção da plataforma deverá ser criado um painel de administração, onde as seguintes tarefas podem ser executadas:

- **Sincronização dos Horários**: Nesta opção o administrador poderá consultar os pedidos de sincronização de horários e proceder à sincronização.
- **Gestão de utilizadores**: A gestão de utilizadores deve ser realizada nesta opção. Deve ser possível atribuir ou revogar aos utilizadores capacidades de gestão nas funcionalidades supra-citas, bem como a definição da instituição a que o utilizador pertence.
- **Gestão de Sessões**: Deve ser possível ao administrador verificar as sessões actualmente activas. Adicionalmente deve ser possível consultar os passos que o utilizador realizou durante a sessão. Esta funcionalidade permitirá ao administrador diagnosticar problemas que possam surgir com a utilização da plataforma.
- **Transmutação de Sessão**: Para fins de depuração de erros e assistência na utilização da plataforma, o administrador deve poder transmutar-se num utilizador específico. Esta funcionalidade será apenas visível aos administradores da plataforma pelas suas implicações a nível ético, de segurança e privacidade.

A aplicação deverá comportar vários níveis de acesso, cada um deles representando os vários participantes em todo o processo. Esta modularidade permitirá que os utilizadores possam desempenhar vários papéis dentro da plataforma com base nas suas credenciais de acesso. Desta forma, podem ser definidos vários perfis de utilizador, nomeadamente:

- **Docente**: Utilizadores com acesso à funcionalidade de inserção de sumários e presenças, pedidos de alteração de aulas e impressão de documentos relacionados com os sumários.
- **Aluno**: Os alunos poderão consultar os sumários inseridos pelos docentes às unidades curriculares em que se encontram inscritos.
- **Supervisor**: Os utilizadores com este perfil poderão supervisionar o progresso da inserção do sumários.

- **Comissão de Horários**: Utilizadores com capacidade de criação, alteração e anulação de aulas.
- **Secretariado**: Utilizadores com capacidade de validação e justificação de aulas e notificação de sumários em falta.
- **Pedidos de Alteração**: Neste perfil os utilizadores podem aceitar ou rejeitar os pedidos de alteração de aulas. Adicionalmente, estes utilizadores devem poder criar pedidos de alteração para aulas nos horários de todos os docentes.
- **Administrador**: Utilizadores com acesso às funcionalidade no painel de administração.

A gestão das credenciais dos utilizadores será levada cabo pelos administradores da plataforma na funcionalidade de gestão de utilizadores.

Alunos

Os alunos devem poder consultar os sumários inseridos para as unidades curriculares a que se encontram inscritos. Deve ser possível ao aluno verificar as aulas em que lhe foi marcada presença.

A aplicação deve ser desenvolvida tendo em mente a utilização pelas várias instituições do campus. É de toda a conveniência que os utilizadores tenham acesso apenas à informação da instituição a que pertencem.

A interface com o utilizador deve ser intuitiva e de fácil utilização, por forma a tornar a informatização do processo o mais transparente possível.

3.2 Interacção e Dependências

A aplicação de gestão de sumários tem enquadramento no IPB, onde têm surgido várias aplicações de apoio à actividade lectiva e nas quais se baseia. Consegue-se, desta forma, minimizar a duplicação de esforço, evitando replicar funcionalidades que se encontram previstas em outras aplicações.

As mais importantes, pela natureza da informação e pela ampla divulgação entre as Escolas, são as aplicações dos Serviços Académicos e a aplicação de Gestão de Actividades Lectivas.

3.2.1 Serviços Académicos

Os Serviços Académicos concentram a gestão de todos os assuntos relacionados com os actos académicos do Instituto. Como tal, enquadram-se, em termos hierárquicos, supra-escola,

providenciando serviços comuns a todas as unidades orgânicas do IPB.

Para suportar toda a informação académica dos alunos do IPB, baseia-se num sistema de informação, organizado sob a forma de uma hierarquia de entidades. A aplicação de sumários utilizará esta informação em conjunto com a informação obtida a partir do software de horários para obter os dados necessária à utilização da plataforma.

A hierarquia da informação dos serviços académicos não é, no entanto, necessária no seu todo para o funcionamento da aplicação de sumários. Da informação disponível nos Serviços Académicos, os sumários necessitam apenas de um subconjunto, nomeadamente:

- Informação sobre as **Escolas**
- Informação da **Inscrições dos Alunos** às Unidades Curriculares utilizada na obtenção da lista de alunos para a marcação das presenças.
- Dados dos **Alunos** (nome, e-mail, etc. . .)
- Informação acerca de **Cursos, Planos e Disciplinas** (Unidades Curriculares) para mostrar no horário do docente.

O acesso à base de dados é fornecida sob a forma de um conjunto de vistas SQL (Figura 3.1).

Em baixo descreve-se a informação contida em cada uma das vistas, bem como a relação entre elas.

- **IPB_ESCOLAS** - Nesta vista pode ser consultada a informação relativa às escolas do campus:
  - `cod_escola`: Código da Escola
  - `escola`: Nome da Escola
  - `abreviatura`: Nome abreviado da escola
  - `activa`: Se a escola deve ser considerada para utilização

  IPB_CURSOS - Nesta vista pode ser consultada a informação relativa aos cursos disponíveis para uma determinada escola:
  - `cod_escola`: Código da Escola em que o Curso está a ser ministrado
  - `cod_curso`: Código do Curso
  - `curso`: Nome do Curso
  - `abrev`: Nome abreviado do Curso
  - `cod_tipo`: Tipo de curso (Licenciatura, Bacharelato, . . .)

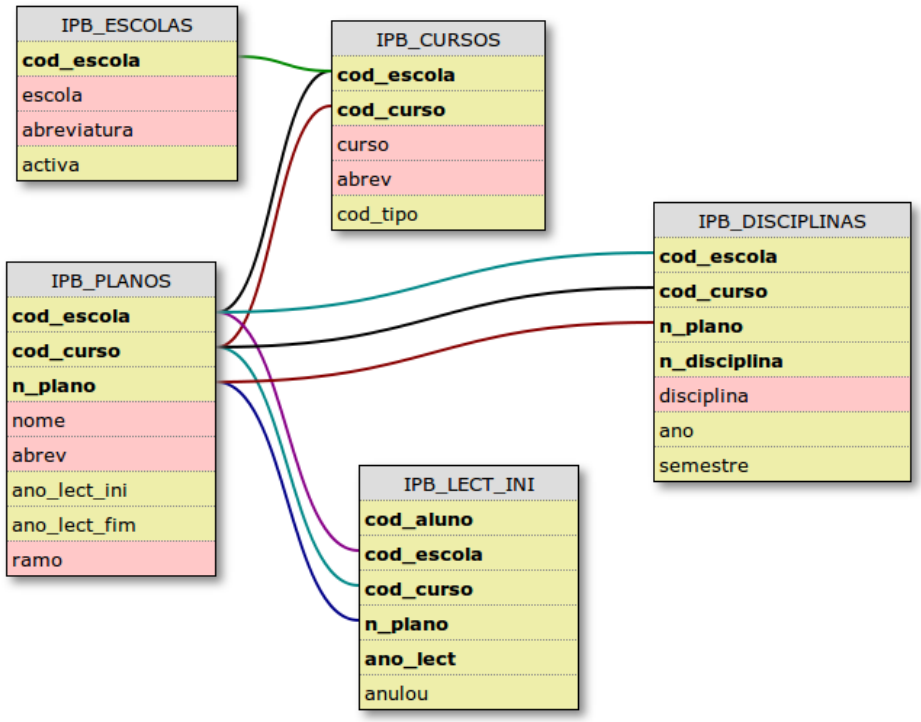


Figura 3.1: Vistas SQL dos Serviços Académicos

IPB_PLANOS - Nesta vista pode ser consultada a informação relativa aos planos existentes para cada curso:

- `cod_escola`: Código da Escola em que o Curso está a ser ministrado
- `cod_curso`: Código do Curso a que este Plano pertence
- `n_plano`: Número do Plano
- `nome`: Nome do Plano
- `abrev`: Nome abreviado do Plano
- `ano_lect_ini`: Ano lectivo em que o plano teve inicio
- `ano_lect_fim`: Ano lectivo em que o plano terminou
- `ramo`: Nome do ramo. Normalmente refere-se a uma ramificação do mesmo curso.

IPB_DISCIPLINAS - Nesta vista pode ser consultada a informação relativa às disciplinas existentes em cada plano:

- `cod_escola`: Código da Escola em que o Curso está a ser ministrado
- `cod_curso`: Código do Curso da Disciplina

- `n_plano`: Número do Plano a que a Disciplina pertence
- `n_disciplina`: Número da Disciplina
- `disciplina`: Nome da Disciplina
- `ano`: Ano curricular em que a disciplina é ministrada (1º, 2º, ...)
- `semestre`: Semestre em que a disciplina é ministrada

IPB_LECT_INI - Vista que contém as inscrições dos alunos nos planos:

- `cod_aluno`: Código do Aluno
- `cod_escola`: Código da Escola
- `cod_curso`: Código do Curso
- `n_plano`: Número do Plano da Inscrição
- `ano_lect`: Ano lectivo da Inscrição
- `anulou`: Quando este campo está activo significa que houve anulação da inscrição.

Os dados de cada aluno podem ser consultados na vista `IPB_ALUNOS` (Figura 3.2). Esta vista contém os seguintes campos:

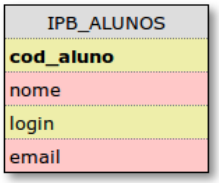


Figura 3.2: Vista dos Alunos

- `cod_aluno`: Código do Aluno
- `nome`: Nome do Aluno
- `login`: Login do Aluno
- `email`: E-mail do aluno

3.2.2 Sistema de Gestão de Actividades Lectivas (GAL)

O Sistema de Gestão de Actividades Lectivas facilita a elaboração de horários e marcação de exames. O `GAL` é uma aplicação *desktop* cujo funcionamento se baseia num conjunto de *plugins*. Tem vindo a ser expandida desde 2004 através da adição de funcionalidades de acordo com a adesão das unidades orgânicas à sua utilização.

De entre as suas funcionalidades (*plugins*), destacam-se com especial ênfase:

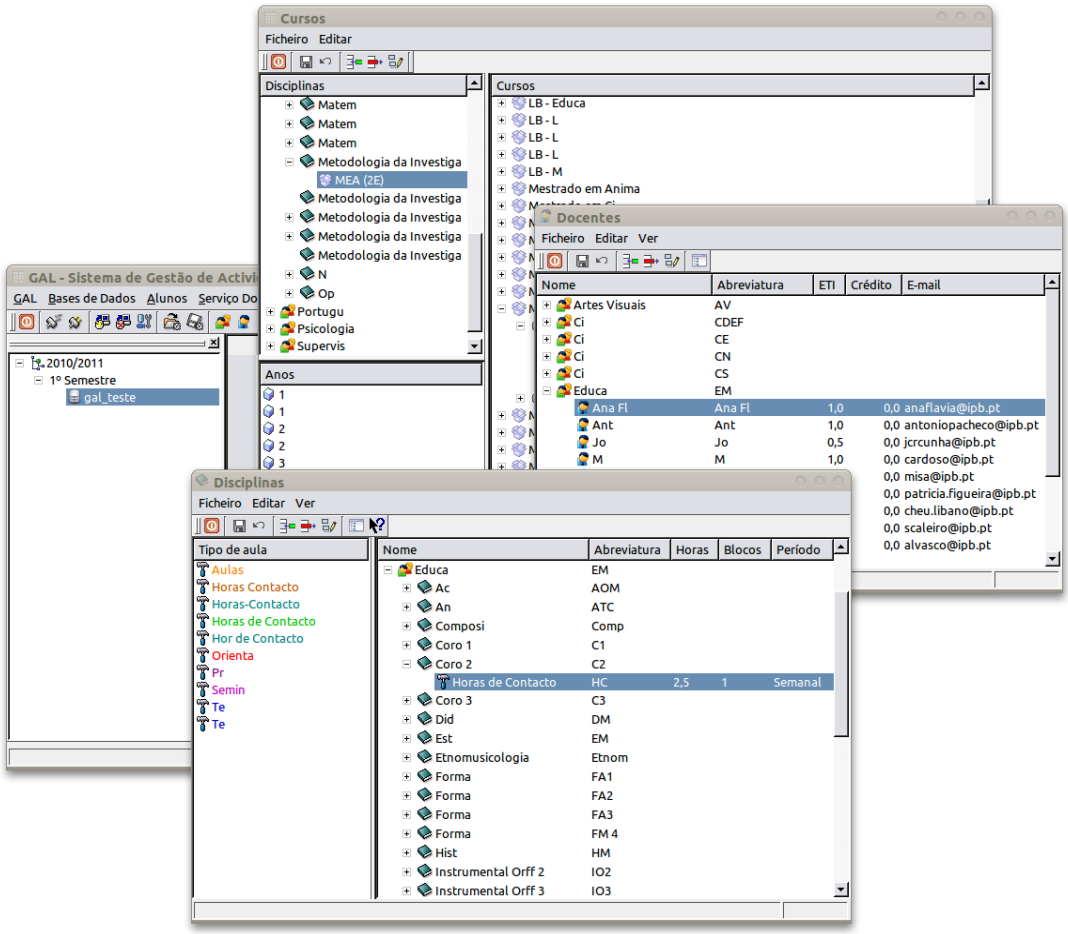


Figura 3.3: Sistema de Gestão de Actividades Lectivas

- Registo manual de docentes, cursos, planos e unidades curriculares, com possibilidade de importação a partir dos Serviços Académicos.
- Registo de Salas e Gestão de Recursos.
- Elaboração de Horários.
- Marcação de Exames.
- Impressão de diversos Documentos relacionados com as actividades lectivas.

Para além dos *plugins* desenvolvidos, existe um conjunto de funcionalidades adicionais que serão objecto de integração futura na aplicação, tais como a Geração Automática de Horários.

Um esforço conjunto da equipa de desenvolvimento da plataforma dos sumários em conjunto com a equipa de desenvolvimento do `GAL`, permitiu a integração das duas aplicações pela adição de uma opção de exportação dos horários construídos para a aplicação de sumários.

A exportação dos horários das várias escolas é centralizada numa base de dados intermédia de controlo de versões. Esta base de dados serve como a fonte de novas versões para a plataforma de sumários.

3.3 Análise

Com o objectivo de garantir a possibilidade de migração e actualização, o funcionamento da aplicação de sumários assenta em três camadas - **Interface com o Utilizador**, **Camada Funcional** e **Base de Dados** (Figura 3.4).

As camadas estão interligadas de forma a que alterações ou mesmo a substituição de uma delas não implique alterações nas restantes. Esta topologia permite que o desenvolvimento e funcionamento da aplicação sejam modulares.

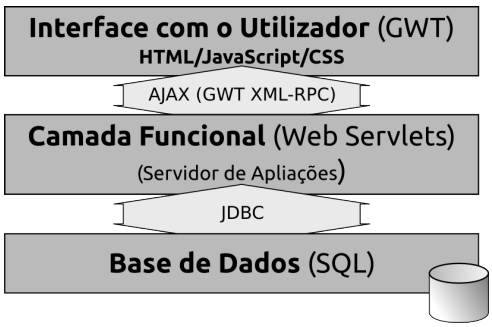


Figura 3.4: Plataforma de Sumários

A camada **Interface com o Utilizador** contém os componentes gráficos de interacção com o utilizador final. O desenvolvimento dos ecrãs é feito com recurso à framework GWT (Google Web Toolkit). Esta camada contém os ficheiros HTML, JavaScript e CSS que definem a Aplicação Web dos sumários.

A comunicação feita entre a Aplicação Web dos sumários e o servidor é feita de forma assíncrona com recurso à tecnologia `AJAX` e ao protocolo `XML-RPC` disponível na `API` do `GWT`. Esta comunicação define o canal de comunicação com a camada abaixo.

A **Camada Funcional** contém toda a lógica da aplicação. Esta camada medeia os pedidos feitos na Aplicação Web e executa as operações necessárias nas camadas subsequentes. Toda esta lógica encontra-se no servidor de aplicações sob a forma de *Web Servlets* que processam os pedidos `XML-RPC` provenientes da interface gráfica, executam as operações solicitadas e respondem com a informação resultante.

Esta camada é ainda responsável pela autenticação e validação do nível de acesso dos utilizadores, bem como a sincronização com o software de horários (GAL).

A interacção com a camada seguinte é feita com recurso à tecnologia `JDBC`[1] (Java Database Connectivity [Andersen, 2006]). O `JDBC` fornece uma `API` standard para acesso a diversos motores de bases de dados de diversos fabricantes.

Finalmente a camada **Base de Dados** alberga o sistema de informação dos sumários onde serão armazenados os dados gerados na aplicação de sumários. Nesta camada estão implícitas ainda as ligações aos sistemas de informação dos Serviços Académicos e Sistema de Gestão de Actividades Lectivas (GAL).

3.3.1 Sistema de Informação dos Sumários

O esquema da Figura 3.5 resume as entidades relacionadas com a criação de sumários, marcação de presenças, validação de sumários e justificação de faltas.

As aulas (entidade `AULA`) são organizadas ao longo de períodos lectivos (`PERIODO`) que representam normalmente um semestre. Cada aula pode ser leccionada a mais do que um curso/disciplina (`DETALHES_AULA`), por vários docentes (`DOCENTE`) e tem uma localização definida (`SALA`).

Cada disciplina da aula pode ter mais do que um sumário (`SUMARIO`) associado. Os sumários podem conter várias referências bibliográficas (`BIBLIOGRAFIA`). As presenças do sumário (`PRESENCA`) são marcadas com base no código do aluno (`ALUNO`).

A validação de sumários (`VALIDACAO_SUMARIO`) é feita por um utilizador (`USERS`) com autorização (`MODULE_USER`) para tal. Da mesma forma, um conjunto de utilizadores podem efectuar a justificação de faltas (`JUSTIFICACAO_FALTA`).

[1] http://www.oracle.com/technetwork/java/overview-141217.html

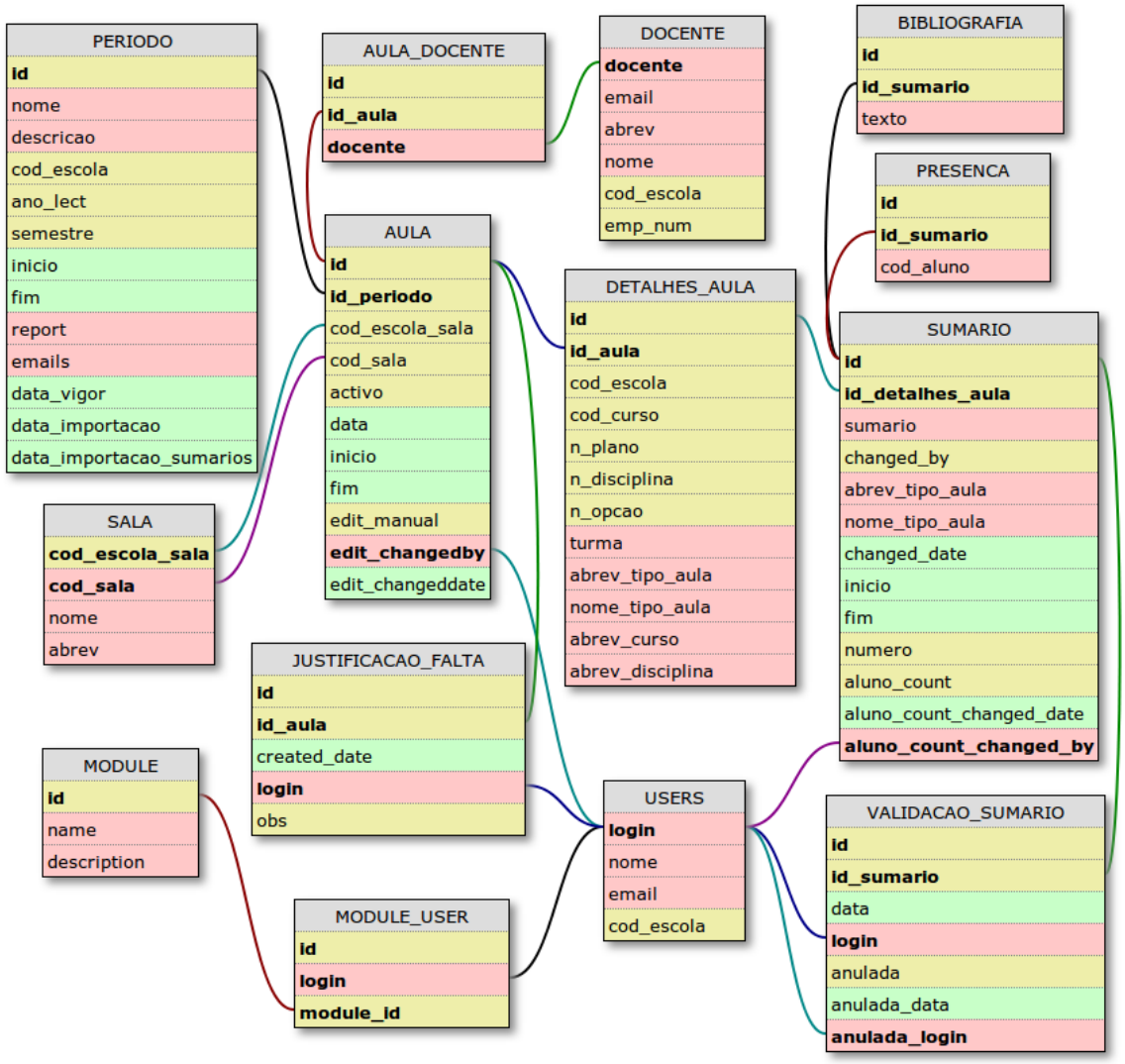


Figura 3.5: Sistema de Informação dos Sumários (A)

A entidade PERIODO tem a função de agrupar as aulas que constam no horário. Não representam necessariamente um período lectivo. Devido à natureza da sincronização dos horários (ver seccção 4.2), o mesmo período lectivo pode ser representado por várias entradas nesta entidade. O campo report contém um relatório com informação acerca do processo de geração feita no software de horários. O campo emails contém os contactos dos membros da comissão de horários que deverão ser notificados de todo processo de sincronização. A data de entrada em vigor deste período pode ser consultada no campo data_vigor. Os campos data_importacao e data_importacao_sumarios referem-se respectivamente, à data de submissão da versão de horários no software de horários e à data em que essa versão foi sincronizada com o software de sumários.

A entidade AULA representa uma aula no horário do docente. A aula tem associada uma data e uma hora de inicio e fim. O campo id_periodo contém o identificador do período a que esta aula pertence. Os campos edit_manual, edit_changedby e edit_changeddate são respectivamente, *flag* indicativa de que a aula foi criada ou alterada de forma manual, o nome de utilizador e a data da alteração. Uma *flag* adicional activo define se esta aula deve aparecer no horário. Os campos cod_escola_sala e cod_sala identificam o local em que a aula vai ter lugar.

A entidade SALA armazena o nome e abreviatura (abrev) de uma sala. Actualmente as salas são apenas identificadas pelo cod_sala, no entanto, a introdução do campo cod_escola_sala vai no sentido de uniformizar a identificação de recursos ao nível de todo o campus.

A associação entre docentes e aulas é feita com recurso à entidade AULA_DOCENTE, podendo neste caso a mesma aula ser dada por vários docentes. A entidade DOCENTE contém a informação do docente, nomeadamente, o nome de utilizador (docente), nome, email, abreviatura (abrev) o número de funcionário (emp_num) e a escola a que pertence (cod_escola).

Os cursos e disciplinas que a aula representa estão descritos na entidade DETALHES_AULA. Os campos cod_escola, cod_curso e n_plano contém os identificadores dos serviços académicos relativamente à escola, curso e plano. A disciplina é identificada pelo número de disciplina (n_disciplina) e no caso desta ser uma opção, o número da opção pode ser consultado no campo n_opcao. No caso desta aula ter uma turma associada para o respectivo curso, esta poderá ser consultada no campo turma. Com o objectivo de manter a coerência com os nomes definidos no software de horários são armazenados (na altura da sincronização) as abreviaturas do curso e da disciplina (abrev_curso, abrev_disciplina). Os nomes completos de cursos, planos, disciplinas e opções são obtidos a partir dos serviços académicos. O nome e abreviatura do tipo de aula ECTS podem ser consultados nos campos nome_tipo_aula e abrev_tipo_aula.

Existem casos particulares de cursos que se encontram desfasados em um semestre relativamente aos outros cursos (Figura 3.6). Os códigos de tais cursos são listados na entidade `DESFASADOS`, contendo o ano lectivo a que se refere o desfasamento, os códigos de escola (`cod_escola`), curso (`cod_curso`) e o número do plano (`n_plano`) e um factor que determina o desfasamento em semestres. A titulo de exemplo, a um curso leccionado no ano lectivo de 2010/2011 no 1º semestre mas que realmente se encontra no ano 2009/2010, 2º semestre deverá ser aplicado um factor -1.

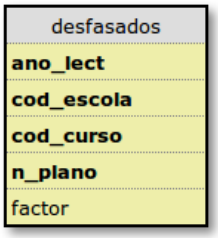


Figura 3.6: Entidade `DESFASADOS`

Os vários sumários de cada unidade curricular são armazenados na entidade `SUMARIOS`. O texto do sumário pode ser consultado no campo `sumario`. Por omissão os sumários terão o mesmo tipo de aula definido na entidade `DETALHES_AULA`, no entanto, o tipo do sumário pode ser redefinido nos campos `abrev_tipo_aula` e `nome_tipo_aula`. O utilizador e data de criação ou alteração de um determinado sumário podem ser consultados nos campos `changed_by` e `changed_date` respectivamente. O `inicio` e `fim` do sumário podem ou não coincidir com o inicio e fim da aula. Este artifício foi inserido com a finalidade de ser possível inserir mais do que um sumário para a mesma aula. Para facilitar a numeração dos sumários foi criado um campo `numero` que irá armazenar a numeração inserida pelo utilizador. Para além da marcação de presenças é útil a marcação sob a forma de uma contagem global (`aluno_count`), armazenando a data (`aluno_count_changed_date`) e o utilizador responsável pela contagem (`aluno_count_changed_by`).

O sumários poderá conter várias referência bibliográficas, definidas na entidade `BIBLIOGRAFIA` no campo `texto`.

A marcação de presenças é feita na entidade `PRESENCA`. Cada entrada nesta entidade representa a presença de um aluno (`cod_aluno`) à aula associada a um determinado sumário (`id_sumario`).

Os utilizadores que têm acesso à plataforma dos sumários são definidos na tabela `USERS`. Cada entrada nesta entidade contém o `nome`, `email` a escola a que pertence (`cod_escola`). A opção de distinguir utilizadores e docentes tem a ver com o facto de alguns utilizadores, tais como supervisores, administradores e comissão de horários não terem serviço docente.

As várias funcionalidades nos sumários estão organizadas sob a forma de módulos definidos na entidade `MODULE`. Cada entrada nesta entidade contém o nome (`name`) e descrição (`description`) do módulo. A atribuição de módulos a utilizadores é feita na entidade `MODULE_USER`. A cada utilizador é dado o acesso a um conjunto de módulos definindo assim o papel do utilizador na plataforma.

Os utilizadores com acesso ao módulo de secretariado têm a capacidade de validar a inserção dos sumários. Para cada validação é criada uma entrada na entidade `VALIDACAO_SUMARIO`. Cada entrada desta entidade contém o identificador do sumário (`id_sumario`), o utilizador que efectuou a validação (`login`) e a `data` em que foi feita a validação. Para o caso de haver uma anulação de uma validação previamente inserida, a *flag* `anulada` será activa e a data (`anulada_data`) e `login` do utilizador que efectuaram a anulação serão armazenados.

Para o caso de falta justificada a uma aula, uma entrada é criada na entidade `JUSTIFICACAO_FALTA`. A entrada contém o identificador da aula a que se refere (`id_aula`), a data de entrega da justificação (`created_date`), o utilizador que recebeu a justificação e criou a entrada (`login`) e um campo de observações onde deve ser inserida a justificação da falta ou informação relevante.

A funcionalide de pedidos de alteração de aulas pode ser implementada recorrendo a um conjunto de entidades adicionais, descritas na Figura 3.7.

Cada pedido de alteração efectuado corresponde a uma entrada na entidade `RESCHEDULE`. A entrada contém informação relativa à aula anterior, a informação da modificação a efectuar no caso de o pedido ser aceite e alguns campos de controlo para determinar a data e identificação do utilizador do pedido. A informação da localização, data, inicio e fim da nova aula é definida nos campos `cod_escola_sala` e `cod_sala`, `data`, `inicio` e `fim` respectivamente. A informação da localização, data, inicio e fim da aula antiga encontra-se nos campos do mesmo nome mas prefixados com `old_`. O campo `docente` contém o nome de utilizador que solicitou o pedido de alteração. Cada registo contém ainda a data em que o pedido foi criado (`data_pedido`) e uma justificação (`obs_pedido`).

A decisão relativamente à aprovação do pedido é armazenada no campo `resultado`, acompanhado das observações (`obs_resultado`) relativas à decisão e da data em que o estado foi alterado (`data_resultado`). O estado de cada pedido é descrito por um valor numérico, cujo significado pode ser:

- `2`: Aceite, mas com modificações feitas ao pedido inicial.
- `1`: Aceite.
- `0`: Pendente (Estado inicial do pedido).
- `-1`: O pedido foi rejeitado (não foi aceite).

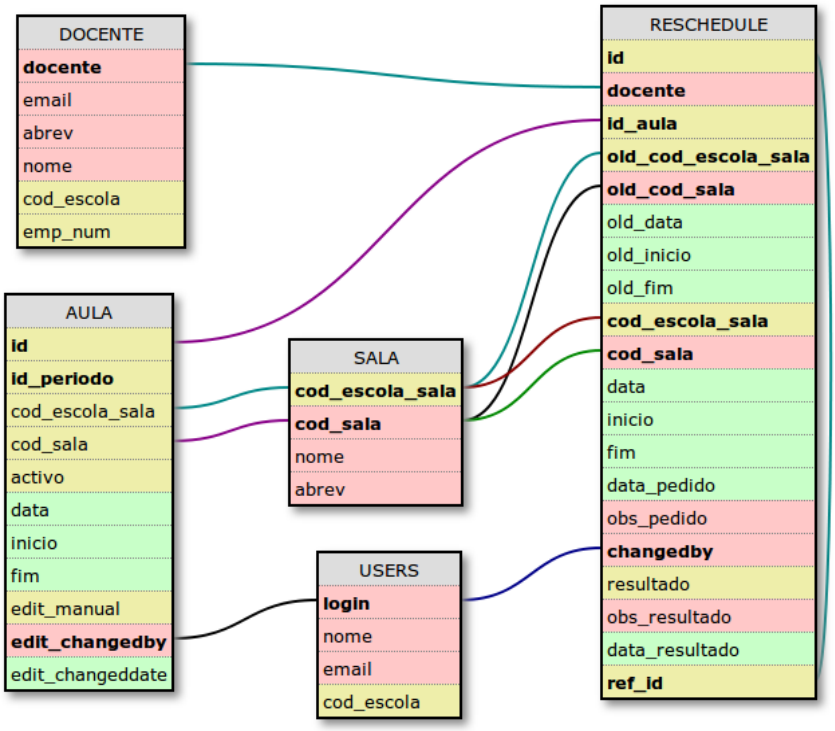


Figura 3.7: Sistema de Informação dos Sumários (B)

- `-2`: O pedido foi cancelado pelo utilizador que o criou.

No caso de o pedido ser aceite mas com modificações, o pedido original é mantido. Para isso o pedido original é marcado com o *resultado* = 2 e um novo pedido é criado com as modificações. Este novo pedido contém uma referência para o original, esta referência pode ser consultada no campo `ref_id` e contém o valor do identificador (`id`) do pedido original.

Na maioria das entidades foi adicionado o campo `id`, que permite acelerar a referência e identificação dos campos para operações de pesquisa, edição e remoção.

A fim de facilitar a depuração de erros duas entidades adicionais foram criadas para armazenar as operações efectuadas durante as sessões dos utilizadores na aplicação (Figura 3.8). Todos os pedidos `RPC` são armazenados e associados a um identificador sessão.

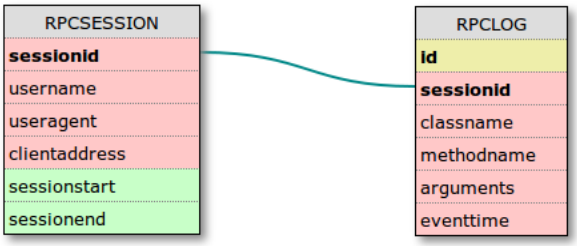


Figura 3.8: Sistema de *Log*

A entidade `RPCSESSION` armazena os dados relativos à sessão do utilizador. O campo `sessionid` armazena o valor interno gerado pelo servidor de aplicações para identificar a sessão. Cada entrada contém ainda o nome de utilizador (`username`), informação acerca do browser utilizado (`useragent`) e o endereço IP da máquina a partir da qual a sessão foi criada (`clientaddress`). O inicio e fim da sessão estão armazenados nos campos `sessionstart` e `sessionend` respectivamente. As sessões abertas no servidor serão aquelas em que o campo `sessionend` é nulo.

A cada pedido `RPC` corresponde uma entrada na entidade `RPCLOG`. Cada entrada contém informação da sessão em que o pedido foi efectuado (`sessionid`), o nome do objecto em que o pedido foi processado (`classname`), o procedimento que foi invocado para processar o pedido (`methodname`), os argumentos utilizados na invocação desse procedimento (`arguments`) e o momento em que o pedido teve lugar (`eventtime`).

Depois de definidas as entidades do sistema de informação dos sumários, estão reunidas as condições par a especificação dos objectos e operações disponíveis na camada funcional.

3.3.2 Camada Funcional

A camada funcional é responsável pela mediação entre os pedidos da interface com o utilizador e a sua concretização na base de dados. Ainda nesta camada estão as estruturas de autenticação e autorização dos utilizadores.

A funcionalidade disponível nesta camada é declarada com recurso a interfaces Java do tipo `RemoteService` do `GWT`. Cada interface define um conjunto de procedimentos abstractos aos quais corresponderá uma implementação do lado do servidor. Ao utilizar o tipo `RemoteService`, os métodos da interface podem ser invocados de uma forma assíncrona, dando origem a chamadas aos procedimentos correspondentes do lado do servidor (via `XML-RPC`).

Os serviços (ou interfaces) disponíveis para invocação remota são:

- `SumariosService`: Operações relacionadas com a utilização do software de sumários. Esta interface concentra uma parte importante dos procedimentos disponíveis.
- `HorarioService`: Operações relacionadas com a criação, edição e remoção de aulas adicionais (Comissão de Horários).
- `ImportService`: Operações relacionadas com a sincronização com o software de horários (Administração).
- `SAService`: Consulta de informação acerca das entidades dos Serviços Académicos.
- `SessionService`: Autenticação e gestão de sessões.
- `StatsService`: Consulta e geração de dados estatísticos.
- `UserService`: Operações relacionadas com a gestão, pesquisa e sincronização de utilizadores e docentes.

Em baixo são enumerados e descritos os métodos de maior relevo disponíveis em cada um dos serviços, bem como o significado dos seus argumentos e resultado. Informação omissa acerca de interfaces, classes e atributos pode ser consultada no código fonte, devidamente comentado, no Apêndice B.2. Em baixo são apenas apresentados os métodos que representam acções ou executam algum tipo de funcionalidade.

`SumariosService`

A interface `SumariosService` agrupa a maior parte da funcionalidade necessária à aplicação dos sumários, no que respeita ao exercício da docência. O código fonte completo desta interface pode ser consultado no Apêndice B.1.5.

```
Integer sugereNum(Integer idDetalhesAula);
```

O método permite obter uma sugestão do número do sumário com base na disciplina. A disciplina é identificada com base no argumento `idDetalhesAula`.

```
Aula validaAula(Integer idAula, Boolean justificacao, String justificacaoObs);
```

Ao invocar este método, os sumários da aula com o identificador `idAula` são marcados como válidos. A observação é fornecida no argumento `justificacaoObs`. No caso desta validação corresponder a uma justificação, o valor `justificacao` deverá ser `true`.

```
void validaSumario(Integer id, Boolean justificacao, String justificacaoObs);
```

O resultado obtido na invocação deste método é semelhante ao anterior. No entanto, a validação afecta apenas um sumário.

```
Aula invalidaAula(Integer idAula, String motivo);
```

Ao invocar este método, a aula com o identificador `idAula` é marcada como inválida. As observações para a invalidação são fornecidas no argumento `motivo`.

```
void enviarSugestao(String moduleId, String moduleName, String info, String
    subject, String message);
```

Ao invocar este método, um e-mail é enviado à lista de suporte com a mensagem descrita nos argumentos `subject` e `message`. O identificador do módulo actual (`moduleId`), o nome do módulo (`moduleName`) e informação de contexto adicional (`info`) são anexados à sugestão enviada.

```
void fastCreateSumario(Integer idAula, String sumario);
```

Este método permite a criação do sumário para uma determinada aula (`idAula`), apenas com a introdução do seu texto (`sumario`). No caso de a aula ser leccionada mais do que uma disciplina, será criado um sumário com o mesmo texto para cada uma delas.

```
Sumario createSumario(Integer idDetalhesAula, String sumario, StringTimestamp
    inicio, StringTimestamp fim, String abrevTipoAula, String nomeTipoAula, List
    <String> bibliografia, Integer numero);
```

Este método permite especificar um sumário (`sumario`) para cada disciplina (`idDetalhesAula`), bem como o seu `inicio`, `fim`, tipo de aula (`abrevTipoAula` e `nomeTipoAula`), as referências bibliográficas (`bibliografia`) e o número do sumário. O valor devolvido contém a informação do sumário criado num objecto do tipo `Sumario` (ver B.2.5).

```
void anulaSumarios(Integer idAula);
```

Este método permite anular os sumários previamente inseridos para uma determinada aula (`idAula`).

```
List<Sumario> replaceSumarios(Integer idDetalhesAula, List<Sumario> replacement)
    ;
```

Este método possibilita a alteração do(s) sumário(s) previamente inseridos para uma determinada disciplina (`idDetalhesAula`). Os novos sumários são fornecidos sob a forma de uma lista de objectos do tipo `Sumario`. O valor devolvido por este método contém a lista de sumários criada nesta invocação.

```
void addPresenca(Integer idSumario, Integer codAluno);
```

Este método permite marcar a presença de um aluno (`codAluno`) num sumário (`idSumario`).

```
void removePresenca(Integer idSumario, Integer codAluno);
```

As presenças inseridas com recurso ao método anterior podem ser removidas invocando este método com os mesmos argumentos.

```
void setPresencas(Integer idSumario, List<Integer> codAlunos, Integer alunoCount
    );
```

Este método permite a alteração da lista de alunos presentes (`codAlunos`) para um determinado sumário (`idSumario`). Em vez da lista de alunos presentes, pode ser definida a contagem de alunos (`alunoCount`).

```
void setModuleEnabled(String user, String moduleId, boolean enabled);
```

Este método permite dar ou remover, dependendo do valor do argumento `enabled`, autorização de acesso a um módulo (`moduleId`) a um determinado utilizador (`user`).

```
ReScheduleRequest requestReschedule(Integer idAula, StringDate newDate,
    StringTimestamp inicio, StringTimestamp fim, SalaId newIdSala, String obs);
```

Com este método é possível efectuar um pedido de alteração de uma determinada aula (`idAula`). O método recebe ainda como argumentos, a nova data da aula (`newDate`), a nova hora de `incio` e `fim` e a nova localização (`SalaId` - ver B.2.4). As observações relativas ao pedido são fornecidas no argumento `obs`. O método devolve um objecto do tipo `ReScheduleRequest` (ver B.2.3), que contém toda a informação relativa ao pedido.

```
void processReschedule(Integer id, String obs, ReScheduleStatus result);
```

O processamento do pedido submetido com o método anterior é feito com recurso a este método. Como argumentos são fornecidos, o identificador do pedido de alteração (`id`), as observações que descrevem a decisão (`obs`) e o resultado da decisão. O resultado da decisão é do tipo `ReScheduleStatus` (ver B.2.6) que determina se o pedido foi ou não aceite.

```
void approveWithChanged(Integer id, StringDate newDate, StringTimestamp inicio,
    StringTimestamp fim, SalaId newIdSala, String obs);
```

Este método aprova o pedido de alteração solicitado, no entanto especifica valores alternativos à alteração solicitada. Para além do identificador do pedido de alteração (`id`),

o método recebe como argumentos, a nova data da aula (`newDate`), a nova hora de `incio` e `fim` e a nova localização. Uma observação é também fornecida (`obs`) como justificação da alteração.

```
void cancelReScheduleRequest(Integer id);
```

Este método permite cancelar um pedido de alteração previamente criado. O argumento `id` representa o número do pedido de alteração.

HorarioService

Na interface `HorariosService` estão os métodos necessários à criação e edição de aulas adicionais. A funcionalidade contida nesta interface é utilizada no módulo de Comissão de Horários. O código fonte completo desta interface pode ser consultado no Apêndice B.1.2.

```
Aula createAula(Aula a);
```

Este método permite criar uma aula adicional com base nos parâmetros do argumento `aula`. Este argumento é do tipo `Aula` (ver B.2.7). O método devolve um objecto do mesmo tipo contendo a informação da aula criada.

```
Aula updateAula(Aula a);
```

Este método permite alterar os valores de uma determinada aula (argumento `aula`). O valor devolvido é a nova aula com os valores especificados no argumento.

```
void dropAula(Integer idAula);
```

Este método permite remover uma aula com base no seu identificador (`idAula`).

```
void notificarAula(Integer idAula);
```

Este método permite aos utilizadores do módulo de secretariado notificar um docente de sumários em falta para uma determinada aula (`idAula`). A notificação é feita via e-mail e contém os dados da aula para a qual o sumário está em falta.

ImportService

A interface `ImportService` contém os métodos necessários à sincronização com o software de horários. O código fonte completo desta interface pode ser consultado no Apêndice B.1.1.

Em resumo a sincronização é feita a partir de uma base de dados intermédia, que contém uma estrutura apropriada à importação para os sumários. Essa estrutura contém um ou mais períodos exportados a partir do software de sumários, correspondendo às várias versões dos horários das unidades orgânicas do IPB. A cada exportação do software de sumários corresponderá um pedido de sincronização visível no painel de administração (ver detalhes na Secção 4.2).

```
void importPeriodo(int idPeriodo);
```

Este método desencadeia a sincronização dos horários com a base de dados intermédia para o período com o identificador `idPeriodo`.

```
String getReport(Integer idPeriodo);
```

Este método permite obter o relatório de geração criado no software de sumários para um determinado período (`idPeriodo`) na base de dados intermédia.

```
void sendImportSucccededNotification(Integer idPeriodo);
```

Este método permite notificar a comissão de horários responsável pela elaboração de uma determinada versão dos horários de que a sincronização foi bem sucedida. Os contactos da comissão de horários para cada período pode ser obtido a partir do campo `emails` da entidade `PERIODO` (Figura 3.5).

```
void descartarPeriodo(Integer idPeriodo, String motivo);
```

Por vezes, após análise do relatório de geração, ou por qualquer outra irregularidade é necessário rejeitar um pedido de importação. Isso pode ser conseguido invocando este método fornecendo como argumentos, o identificador do período da base de dados intermédia (`idPeriodo`) e uma descrição textual da razão que motivou a rejeição do pedido de sincronização (`motivo`).

SessionService

A interface `SessionService` define os métodos para a gestão das sessões e autenticação. O código fonte completo desta interface pode ser consultado no Apêndice B.1.6.

```
User login(String login, String passwd);
```

Este método permite iniciar uma sessão no servidor para o utilizador com as credenciais definidas pelos argumentos `login` (nome de utilizador) e `passwd` (password do utilizador). O valor devolvido contém a informação do utilizador correspondente ao `login` fornecido, ou será nulo no caso de a autenticação ter falhado.

```
User remoteUser();
```

Este método permite obter a informação do utilizador para a sessão actual. O valor devolvido é um objecto do tipo `User` (ver B.2.1) contento os dados do utilizador.

```
void logout();
```

Este método termina a sessão actual.

```
User becomeUser(String login);
```

Com o objectivo de prestar assistência aos utilizadores dos sumários, por vezes é necessário seguir os seus passos, permitindo uma depuração do problema de forma mais eficaz. Este método permite a um utilizador (com privilégios para tal) transformar a sua sessão na de um utilizador em específico.

`UserService`

A interface `UserService` define os métodos necessários a gestão e sincronização de utilizadores. O código fonte completo desta interface pode ser consultado no Apêndice B.1.4.

```
User addUser(User user, Integer codEscola, String[] modules) throws
    SumariosException;
```

Este método permite criar um utilizador (`user`) na entidade `USERS` (Figura 3.5), pertencente a uma determinada escola (`codEscola`) e com autorização para usar um conjunto de módulos (`modules`). O valor devolvido pelo método é um objecto do tipo `User` contendo a informação do utilizador criado.

```
void removeUser(String login) throws SumariosException;
```

Este método permite remover um utilizador previamente criado. O utilizador a remover é identificado pelo seu nome de utilizador (`login`).

```
List<Docente> syncDocentes(List<String> modules) throws SumariosException;
```

Este método permite sincronizar a lista de docentes obtidos a partir do software de horários (entidade `DOCENTE`) com a lista de utilizadores (entidade `USERS`). O argumento `modules` define a lista de módulos atribuídos por omissão aos utilizadores sincronizados. O método devolve a lista de docentes que foram sincronizados.

A funcionalidade da aplicação de horários foi dividida em módulos afim de ser possível aos utilizadores assumir papéis diferentes com base nos módulos que lhe foram atribuídos. O acesso aos métodos descritos nas interfaces em cima depende dos módulos a que o utilizador tem acesso. Desta forma o controlo de acesso é feito com base nos papéis que o utilizador desempenha. Esta forma de controlo de acesso é denominada de *Role Based Access Control* [NIST, 1995].

Definidas as interfaces e a funcionalidade, é possível definir alguns critérios de interacção com o utilizador no que respeita à interface gráfica.

3.3.3 Prototipagem

Nas aplicações web convencionais, cada acção implica o refrescamento de toda a página, no entanto, devido à quantidade de informação que está disponível normalmente nos

ecrãs do software de sumários, o refrescamento constante da página seria um golpe na funcionalidade da aplicação. Assim a aplicação foi desenvolvida de forma a que cada acção obtenha apenas os dados necessários ao refrescamento selectivo de partes da página.

Os módulos para os quais o utilizador actual tem permissões, estão disponíveis sob a forma de um menu em árvore. Cada item na árvore corresponde a um ecrã que contém os componentes necessários às tarefas associadas ao módulo.

O módulo de **Inserção de Horários** contém uma forma componente temporal. Esta componente é susceptível a erros quando a introdução de variáveis está a cargo do utilizador. De forma a evitar este tipo de erros é de toda a conveniência que sejam utilizados componentes gráficos que permitam ao utilizador abstrair-se das questões temporais, ainda que, mantendo o utilizador consciente do seu estado actual.

Os horários são compostos por eventos distribuídos de forma mais ou menos periódica ao longo de cada exercício. A periodicidade de uma grande parte dos eventos é semanal, pelo que uma vista semanal do horário reflecte de forma mais ou menos natural a actividade lectiva.

Na vista semanal (Figura 3.9) os blocos de aula são dispostos por dia. Cada coluna desta vista representa um dia da semana e cada linha representa um intervalo de tempo em minutos (30 minutos). Desta forma as aulas estão ordenadas de forma cronológica. A vista semanal permite aos utilizadores navegar para uma semana específica, de forma a que a visualização de eventos anteriores ou posteriores à data actual seja possível.

Quando seleccionada, cada aula apresenta um conjunto de opções sob a forma de menu. Para o caso das aulas não sumariadas, o utilizador terá disponíveis as opções para criação de sumários, pedido de alteração de aula (Figura 3.10) e criação de folha de presenças.

Depois da aula sumariada, as opções permitirão ao utilizador editar, imprimir ou anular o sumário actual, criar, alterar ou imprimir a lista de presenças (Figura 3.11) ou ainda imprimir a folha de presenças da aula.

Para além do menu de opções, é disponibilizada ao utilizador informação de contexto acerca da aula, disciplinas e localização do elemento seleccionado.

Quando accionada a opção de criação de sumário para uma determinada aula, é apresentada uma caixa de diálogo contento apenas o campo para inserção do texto do sumário (Figura 3.12). Esta caixa permite a criação rápida de um sumário para a aula.

Uma outra caixa de diálogo é utilizada para a edição de parâmetros adicionais, tais como o tipo de aula, a hora de inicio e fim do sumário e a referência bibliográfica (Figura 3.13).

A mesma caixa de diálogo é utilizada na opção de edição de um sumário previamente inserido.

O módulo de **Presenças** permitirá aos docentes obter estatísticas acerca das presenças

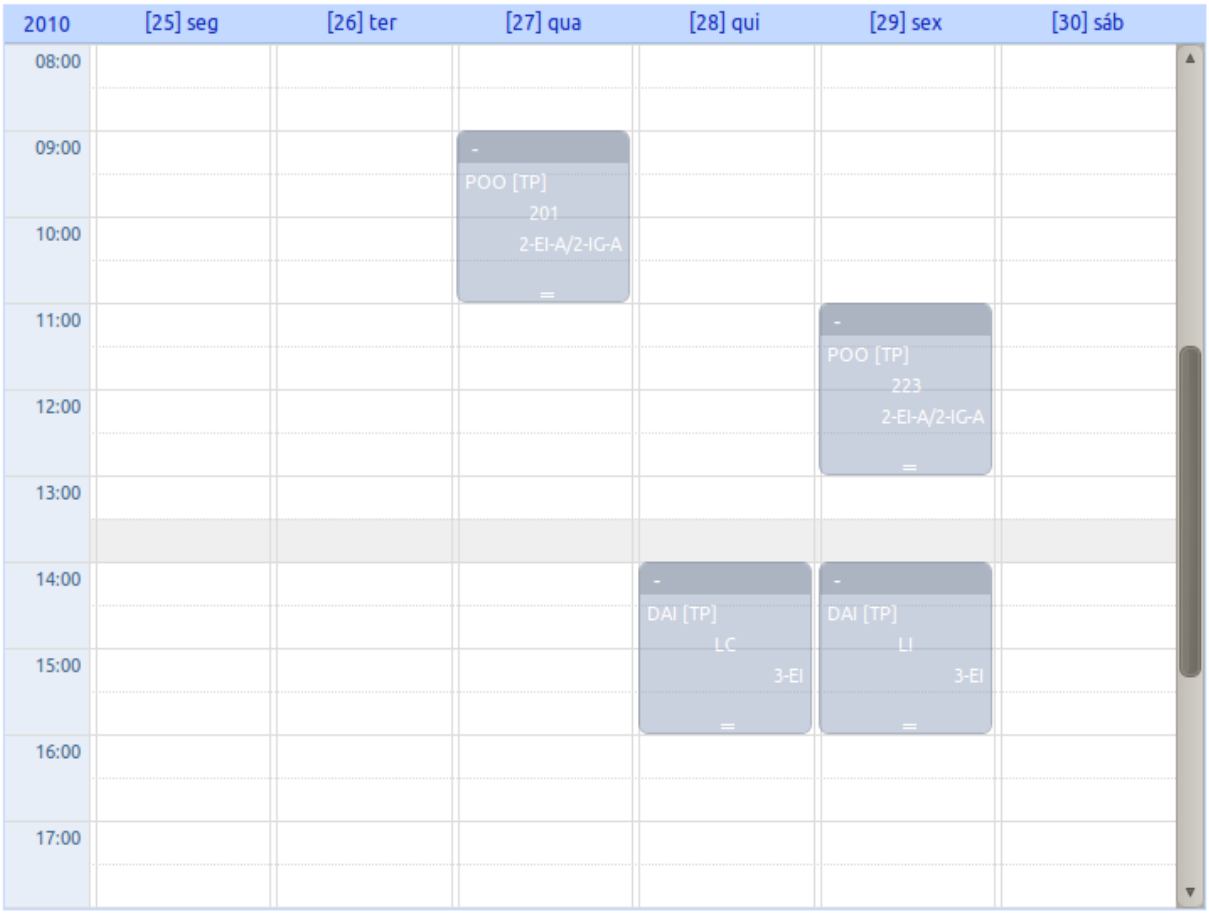


Figura 3.9: Vista Semanal do Horário

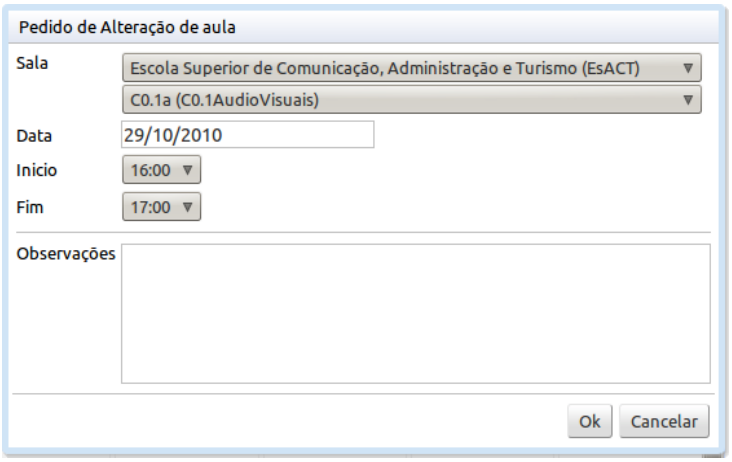


Figura 3.10: Pedido de Alteração de Aula

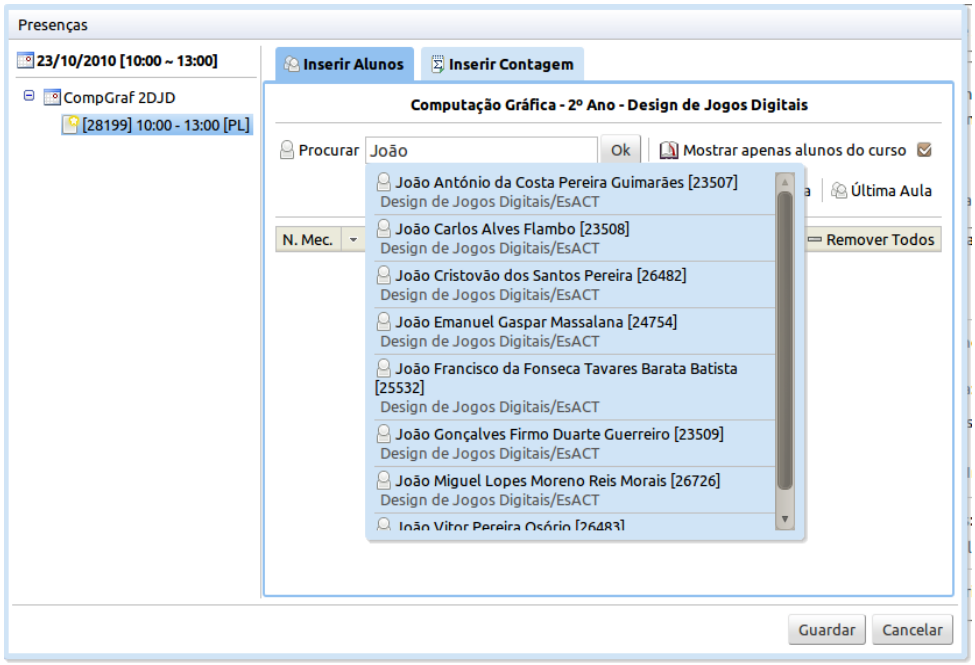


Figura 3.11: Marcação de Presenças

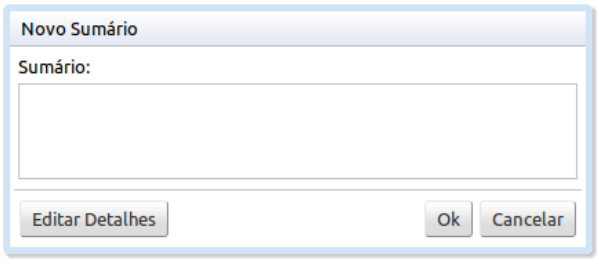


Figura 3.12: Criação rápida de sumários

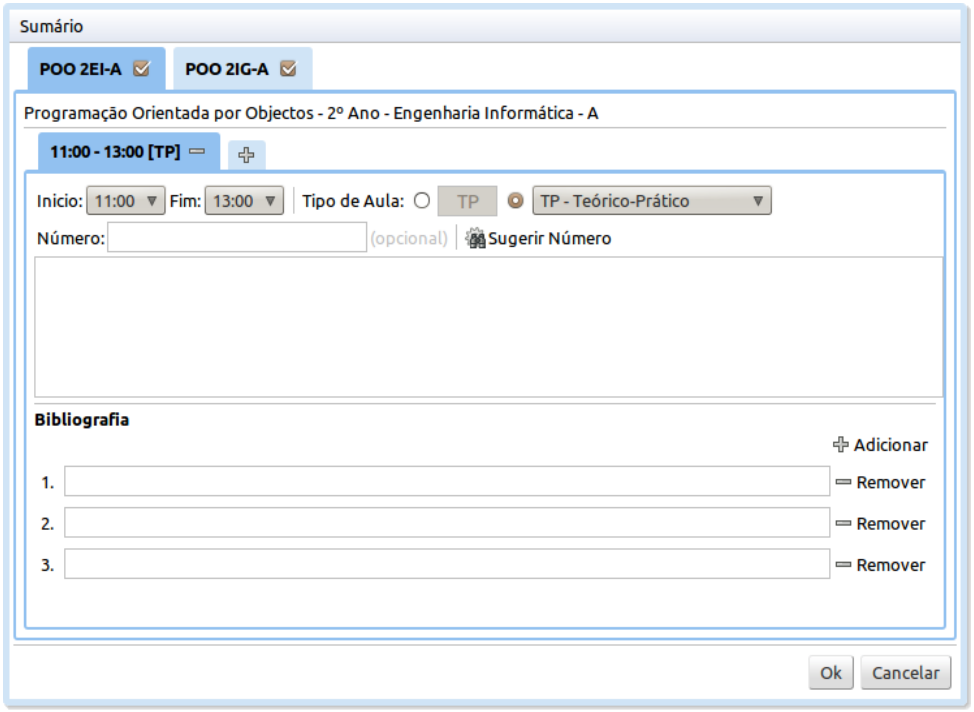


Figura 3.13: Caixa de Diálogo de Criação de Sumários

dos alunos. Para cada tipo de estatística é adicionado um tabulador com filtros apropriados e listagem em forma tabelar (Figura 3.14).

Esta forma de apresentação serve de modelo para a apresentação de estatísticas em geral na aplicação de sumários. É de prever que, com o desenvolvimento futuro da aplicação, mais tipos de estatística em volta das presenças sejam necessárias.

O módulo da **Comissão de Horários** funciona com base na vista semanal do calendário, à semelhança do módulo de Inserção de Sumários. No entanto, existe a necessidade de filtrar as aulas com base no docente, curso e localização da aula. Para o efeito, foi associado um conjunto de filtros à vista semanal (Figura 3.16). Os filtros podem ser associados por forma a permitir refinar a lista de aulas na vista semanal.

A criação de aulas é feita seleccionando uma quadricula no horário. Após essa selecção, uma caixa de diálogo é mostrada ao utilizador para com os campos necessários à criação da aula (Figura 3.15), nomeadamente a localização da aula, as disciplinas desta aula e os docentes que a leccionam. A data e hora utilizadas para a criação da aula tomam o valor lido a partir da quadricula seleccionada.

O duplo clique sobre uma aula existente permite alterar a aula, recorrendo à mesma caixa de diálogo utilizada para a criação. Premindo a tecla `DELETE` do teclado quando uma aula está seleccionada permite ao utilizador remover uma determinada aula.

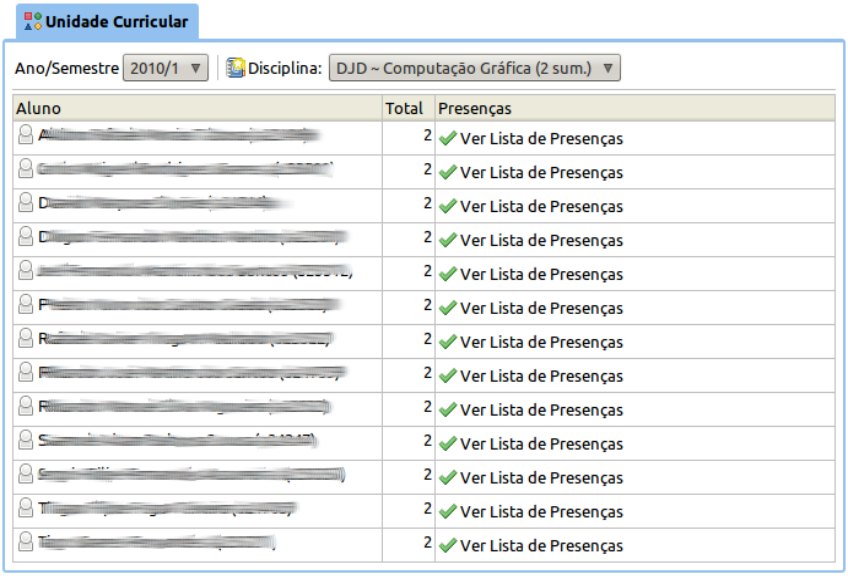


Figura 3.14: Estatística de Presenças (por Unidade Curricular)

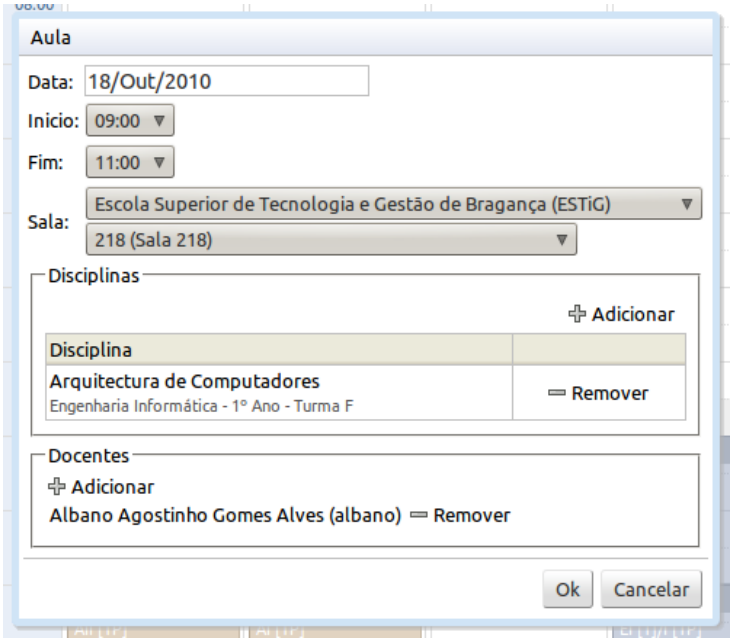


Figura 3.15: Criação de Aula Adicional

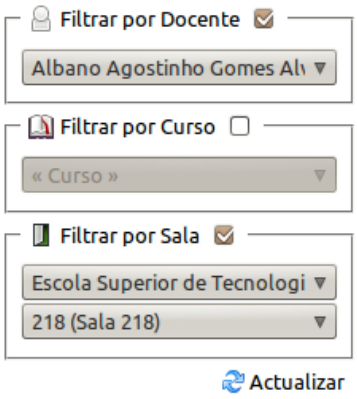


Figura 3.16: Filtro - Comissão de Horários

O módulo de **Secretariado** permite aos utilizadores validar os sumários das aulas e notificar os docentes acerca de sumários em falta. O mesmo filtro usado na comissão de horários permite ao secretariado refinar a lista de aulas numa vista semanal do horário. Quando uma aula é seleccionada, o utilizador tem a opção de validar o sumário, anular uma validação anterior, notificar o docente de um sumário em falta ou então obter informação acerca da aula seleccionada. A validação do sumário é feita com recurso a um *wizard* (Figura 3.17).

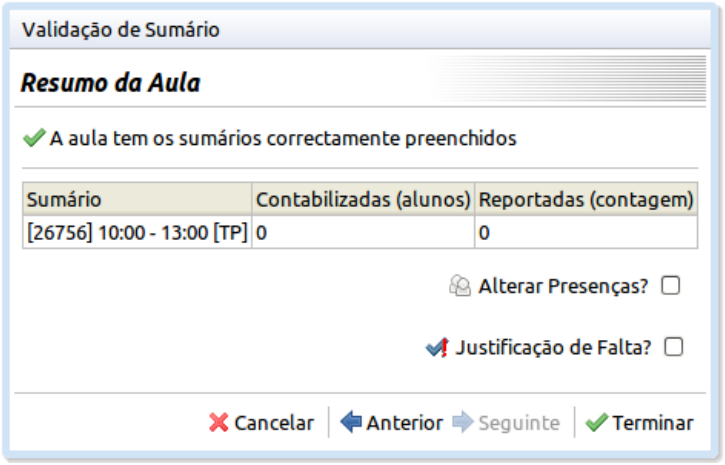


Figura 3.17: Wizard de Validação de Sumários

Neste wizard o utilizador poderá verificar se os sumários foram preenchidos correctamente e alterar as presenças do sumário (para o caso de terem sido entregues em papel). Para o caso de se tratar de uma justificação de falta, a opção de justificação de falta deverá ser activada.

O módulo de **Aprovação de Pedidos** permitirá aos seus utilizadores aprovar pedidos

de alteração de aula submetidos no módulo de Inserção de Sumários. Os pedidos de alteração pendentes são listados de forma tabelar (Figura 3.18) contendo a informação do pedido, nomeadamente, a informação da aula actual, os dados da alteração, as observações do utilizador para o pedido e os dados do utilizador que solicitou a alteração.

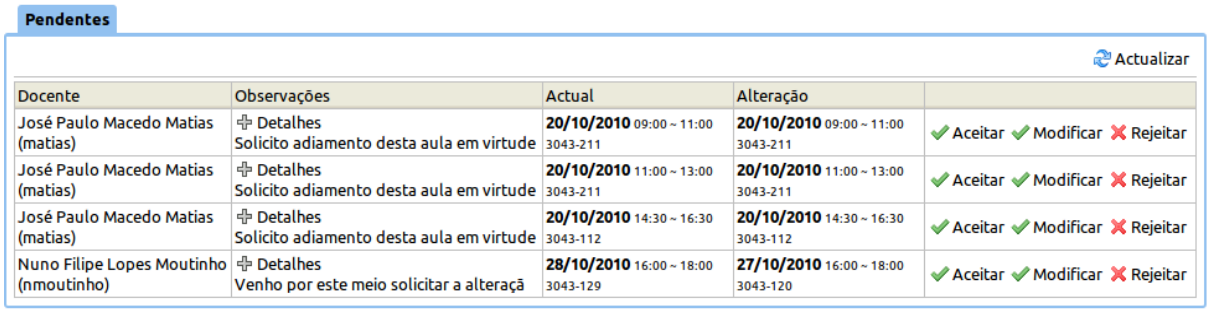


Figura 3.18: Lista de Pedidos Pendentes

Para cada pedido o utilizador tem a opção de aceitar, modificar ou rejeitar. No caso de qualquer uma das opções ser seleccionada, uma caixa de diálogo mostra um formulário para preenchimento dos dados necessários ao processamento do pedido de alteração.

O módulo de **Alteração de Aulas** permite a um funcionário (não docente), fazer pedidos de alteração de aula de forma administrativa. Para conseguir esta funcionalidade foi utilizada a vista semanal na qual as aulas são filtradas com recurso a uma lista de docentes. Depois de seleccionado o docente desejado, a vista é actualizada com a lista de aulas para o docente. Ao seleccionar uma aula é mostrada a opção de criar um pedido de alteração de aula. Os dados do pedido são preenchidos na caixa de diálogo utilizada para os pedidos de alteração (Figura 3.10).

No painel de administração do módulo de **Administração**, estão disponíveis sob a forma de tabuladores as opções de sincronização de docentes, importação de períodos (sincronização de horários) e gestão de utilizadores.

O tabulador de gestão de utilizadores (Figura 3.19) permite a gestão de utilizadores de cada unidade orgânica do IPB. Os utilizadores são mostrados de forma tabular com opção de pesquisa.

Em cada item da lista existe a opção de remover e alterar a lista de módulos a que o utilizador pertence.

O tabulador de sincronização de docentes, permite acrescentar à lista de utilizadores os docentes que foram importados a partir do software de horários. Quando a sincronização é iniciada, é mostrada uma caixa de diálogo com a lista de docentes (Figura 3.20), pedindo a confirmação do utilizador.

O tabulador que permite a sincronização dos horários a partir da base de dados

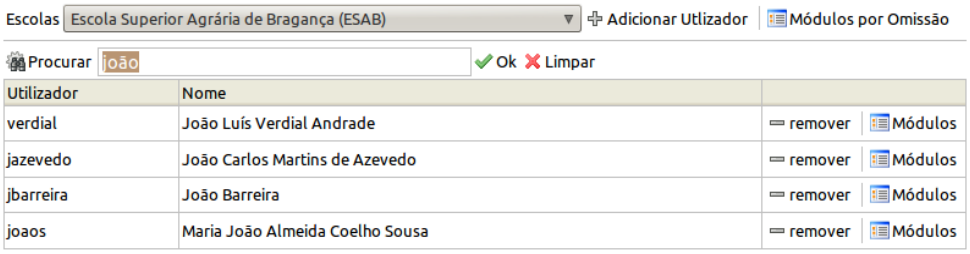


Figura 3.19: Gestão de Utilizadores

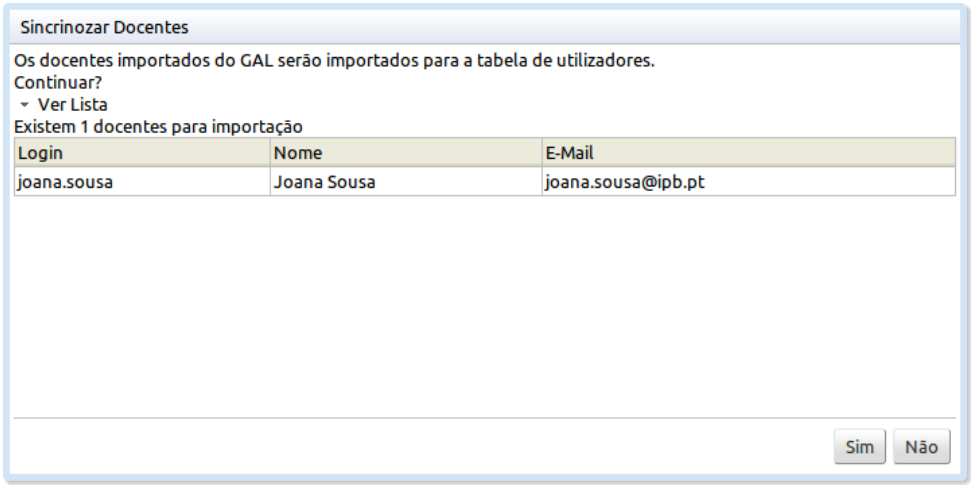


Figura 3.20: Sincronização de Docentes

intermédia, tem o nome de importação de períodos. Neste tabulador é mostrada uma lista das novas versões de horários exportadas a partir do software de horários para a base de dados intermédia (Figura 3.21).

#	Nome	Descriçao	Inicio/Fim	COD. Escola	Ano/Semestre	Data Vigor	Data Import.	
114	Escola Superior de Saúde de Bragança	25/10/2010	20/09/2010 - 15/01/2011	7015	2010/1	25/10/2010	19/10/2010 12:40	Importar Rejeitar

Figura 3.21: Sincronização de Horários

A fim de permitir o *feedback* por parte dos utilizadores da plataforma em qualquer momento, existe uma opção sempre visível no topo da página para a submissão de erros e sugestões. Uma caixa de diálogo permite aos utilizadores introduzir um assunto e uma sugestão (Figura 3.22).

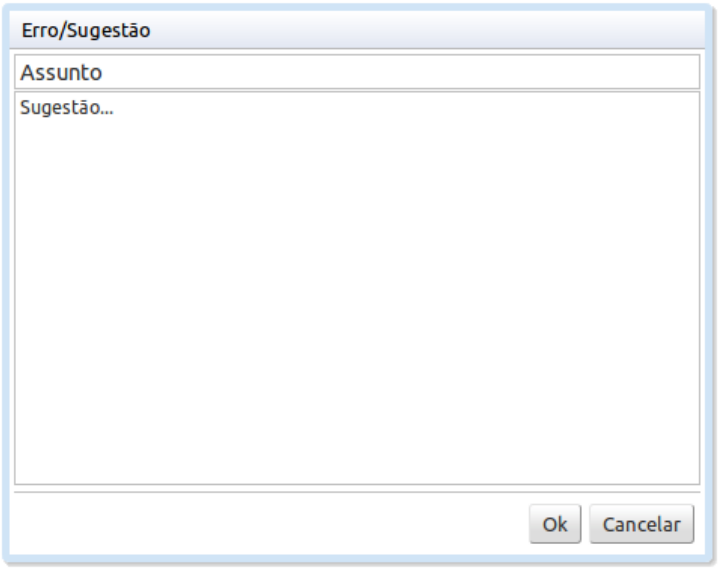


Figura 3.22: Caixa de Diálogo de Erro/Sugestão

Cada item da lista contém a informação de cada versão de horários, nomeadamente, o nome da escola, a data, inicio e fim do exercício, o ano e semestre, a data de entrada em vigor e a data em que a sua exportação foi feita. O administrador poderá importar ou rejeitar cada uma das versões disponíveis na lista. No caso de rejeição é pedida ao utilizador uma justificação. O relatório da geração produzido no software de horários pode ser consultado clicando na linha de cada versão.

O ecrã de de Gestão de Sessões do módulo de Administração permite aos administradores verificar as sessões em curso no servidor. As sessões são filtradas com base no seu estado (abertas, fechadas ou ambas), a data em que foram iniciadas e o login do utilizador (Figura

3.23). As sessões são apresentadas de forma tabelar. Cada linha apresenta a informação de uma sessão, onde constam, o nome de utilizador, informação da aplicação cliente (*browser*), o inicio e fim da sessão e o endereço de rede a partir do qual a sessão foi iniciada.

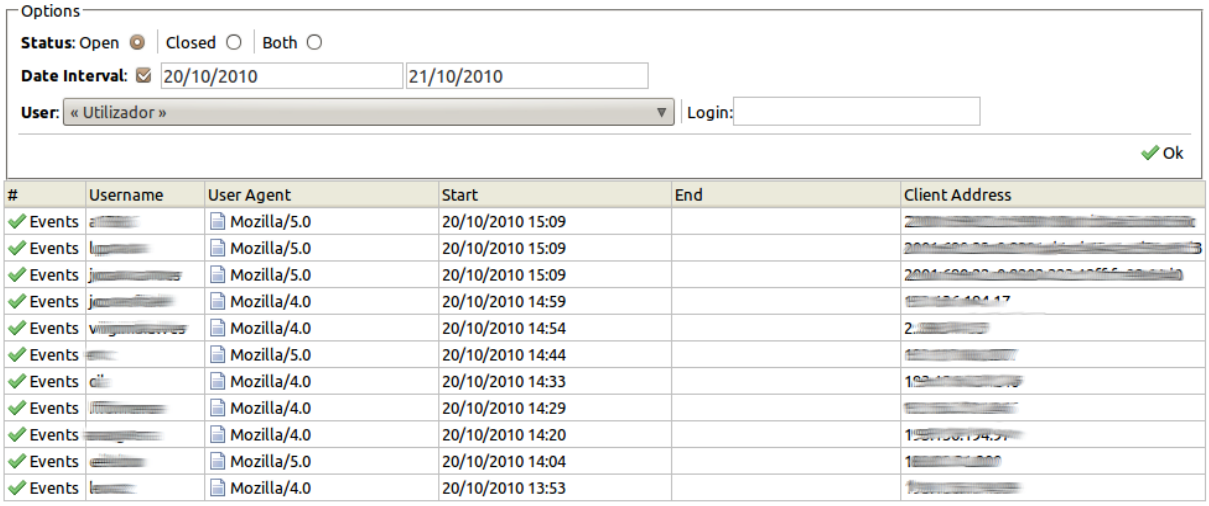


Figura 3.23: Gestão de Sessões

Para cada linha da lista de sessões, existe ainda a opção de consultar a lista de procedimentos invocados e os argumentos da invocação. Para tal uma caixa de diálogo é apresentada com a lista de invocações (Figura 3.24).

O prototipo da aplicação tem em mente a facilidade de utilização das várias funcionalidades da aplicação pela disposição dos componentes gráficos de forma intuitiva e simples de identificar. É feito um esforço no sentido do preenchimento automático de alguns dos campos nas várias caixas de diálogo com informação de contexto, por forma a acelerar e facilitar a utilização da aplicação.

Após levantamento dos requisitos da aplicação e análise do funcionamento, estão reunidas as condições para a fase de implementação.

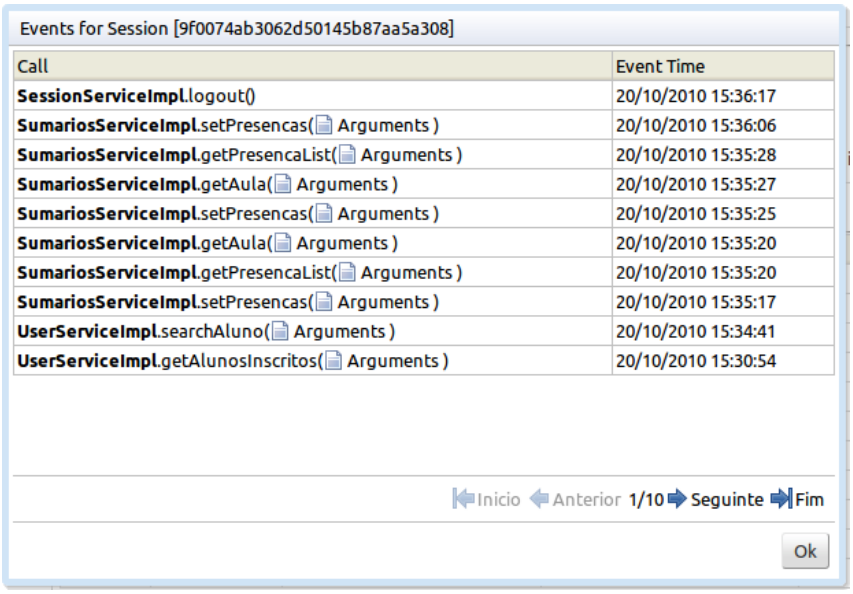


Figura 3.24: Gestão de Sessões

Capítulo 4

Desenvolvimento e Resultados

A plataforma de sumários teve um desenvolvimento faseado no qual foi ouvido um conjunto de intervenientes das várias unidades orgânicas do IPB. Houve a preocupação de utilizar, na medida do possível recursos e software disponíveis por forma a minimizar os custos de desenvolvimento.

Após levantamento dos recursos ao nível de software foram tomadas as opções para Armazenamento de Dados e alojamento da camada lógica da aplicação (Camada Intermédia).

Foram apurados também os recursos ao nível dos utilizadores, no que respeita à forma de acesso a Aplicações Web, por forma a desenvolver uma interface compatível e adequada. Deste modo a aplicação deverá chegar a uma maior quantidade de público alvo.

4.1 Persistência de Informação

Uma parte importante de todos os sistemas de informação do IPB estar a usar o mesmo sistema de base de dados. Este facto teve um peso importante na decisão do sistema de Armazenamento de Dados para a plataforma dos sumários.

A equipa multidisciplinar do Centro de Desenvolvimento do IPB integra, entre outros, um `DBA` (Database Administrator[1]) que lida com os assuntos relacionados com a gestão, manutenção e *backup* da base de dados, centralizando assim, o conjunto de bases de dados de uma parte significativa das aplicações do IPB.

O sistema de informação do IPB, nomeadamente o dos Serviços Académicos, foi desenvolvido com recurso à tecnologia *Oracle Database 11g Enterprise Edition*[2]. A edição enterprise deste sistema de base de dados, oferece melhoramentos ao nível do desempenho e segurança em relação às edições standard. A instância do servidor Oracle está situado num servidor dedicado que aloja os vários *schemas* de base de dados.

[1] Administrador de Bases de Dados
[2] http://www.oracle.com/us/products/database/enterprise-edition-066483.html

Ao optar pela utilização deste sistema, a plataforma de sumários pode usufruir de processos já implementados relacionados com os actos académicos. A não utilização deste sistema, implicaria sincronizações adicionais para o sistema de informação escolhido dos dados necessários à plataforma de sumários, bem como a mimetização dos processos já implementados.

Para o desenvolvimento com recurso à linguagem Java, a Oracle disponibiliza a sua própria implementação da especificação JDBC (Java Database Connectivity[1]). A API JDBC fornece uma abstracção relativamente à implementação das bases de dados de vários fabricantes. A implementação dos fabricantes é fornecida sob a forma de um *Driver* JDBC que implementa as rotinas de acesso à base de dados (Figura 4.1).

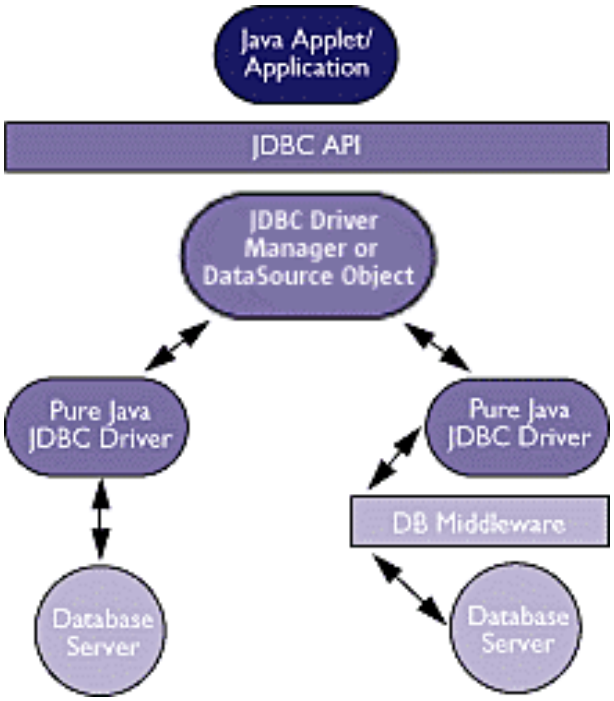


Figura 4.1: Arquitectura JDBC

O acesso à base de dados, a nível aplicacional, da plataforma de sumários é feito com recurso à tecnologia JDBC. Desta forma a decisão de migrar para outro sistema de base de dados não implica mexidas significativas no código.

O servidor de aplicações *GlassFish* permite a configuração de ligações JDBC que mais tarde podem ser utilizadas de forma transparente nas classes Java publicadas com as aplicações. Cada ligação JDBC corresponde a um recurso que é identificado com um nome único. Esse nome único é usado nas classes para obter uma referência ao recurso. A obtenção de referências para recursos nos servidores Java EE é feita com recurso à API

[1] http://www.oracle.com/technetwork/java/overview-141217.html

JNDI (Java Naming and Directory Interface[1]). Desta forma, as ligações são configuradas no servidor de aplicações, tornando o desenvolvimento da aplicação independente da base de dados e dos pormenores de ligação.

Na medida do possível, foi utilizado o SQL standard[2] para a criação da estrutura e procedimentos da base de dados. Ao utilizar código SQL standard, a migração para outro sistema de base de dados de toda a solução implicará mexidas menores no código.

4.2 Sincronização com os Horários

A construção dos horários das várias unidades orgânicas do IPB é conseguida com a ajuda do Sistema de Gestão de Actividades Lectivas (secção 3.2.2). Depois de consolidada uma nova versão dos horários, um pedido de sincronização com os sumários pode ser feito utilizando a funcionalidade de exportação para os sumários.

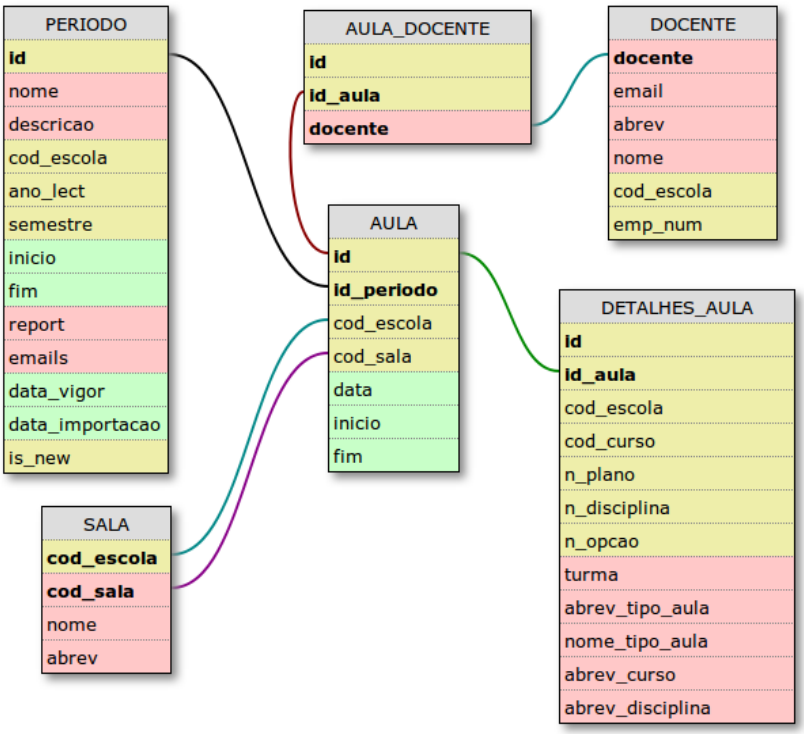


Figura 4.2: Base de Dados Intermédia (Sincronização)

Esta exportação consiste na criação de todos os eventos criados no horário, para uma base de dados intermédia (Figura 4.2). Esta base de dados apresenta uma estrutura

[1] http://www.oracle.com/technetwork/java/index-jsp-137536.html
[2] A Oracle suporta pelo menos o standard SQL-92 - http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt

semelhante à dos sumários no que respeita à actividade lectiva, tendo os campos das tabelas precisamente os mesmos significados (secção 3.3.1).

A construção dos horários no GAL é feita normalmente segundo uma perspectiva semanal (em alguns casos quinzenal). No entanto, para o funcionamento dos sumários são necessários todos os eventos do semestre *a priori*. Na exportação, o GAL replica o horário semanal/quinzenal para a duração de todo o exercício.

A exportação do software de horários cria na base de dados intermédia um novo período (entidade `PERIODO`) com informação acerca da versão exportada. Adicionalmente, um relatório (`report`) é gerado acerca de possíveis irregularidades ou inconsistências na exportação das várias entidades. Ainda no período, são armazenados os contactos da comissão de horários (`emails`).

As aulas exportadas em cada versão dos horários, à semelhança do que acontece na base de dados dos sumários, são agrupadas por período. Os códigos dos cursos, planos, disciplinas, opções, o tipo de aula e a turma associados a cada aula são armazenados na entidade `DETALHES_AULA` e associadas à respectiva aula. Por uma de coerência com os nomes dos cursos, disciplinas e tipos de aula especificados no `GAL`, estes são também armazenados nesta entidade e posteriormente sincronizados para a base de dados dos sumários.

A *flag* `is_new` na entidade `PERIODO` tem o objectivo de identificar os períodos que ainda não foram importados para a plataforma dos sumários ($is\_new = 1$).

Depois da criação da nova versão dos horários na base de dados intermédia, um pedido é enviado via e-mail para a lista de suporte, solicitando a substituição do horário actual com a nova versão a partir da data de entrada em vigor. Um novo item é colocado no painel de administração, no tabulador de sincronização de horários, na lista de pedidos de importação pendentes.

Após a importação ser bem sucedida para os sumários, um e-mail é enviado para os membros da comissão de horários a notificar a disponibilidade da nova versão.

A *flag* `is_new` na entidade `PERIODO` da base de dados intermédia é actualizada com o valor 0, com o objectivo de evitar a importação de períodos repetidos.

As salas, docentes e atribuição dos docentes às aulas (`AULA_DOCENTE`) criados no `GAL` são exportados também para a base de dados intermédia.

A lista de docentes na base de dados dos sumários tem apenas utilidade para o exercício dos sumários, sem qualquer implicação a nível da autenticação. No entanto, quando a lista de docentes é sincronizada (de forma administrativa) com a lista de utilizadores dos sumários, estes passam a ter acesso à plataforma.

Como a introdução de docentes no software de horários é livre[1], poderia levantar uma

[1]Podem ser introduzidos nomes de utilizador de forma arbitrária

questão de segurança, se a sincronização fosse automática, pelo que esta é feita de forma administrativa.

Em cada versão dos horários exportada é definida uma data de entrada em vigor (`data_vigor`). Esta data define o ponto a partir do qual a versão de horários entra em vigor.

A mesma unidade orgânica pode, no entanto, exportar várias versões do mesmo horário durante o mesmo exercício. Cada exportação contém os eventos para todo o semestre, pelo que é necessário evitar a duplicação de eventos. A forma encontrada para evitar esta duplicação passa pela utilização dos vários períodos de forma acumulativa. O ponto que define onde um período acaba coincide com a data de entrada em vigor do próximo. Como resultante, o mesmo horário nos sumários conterá um somatório dos horários parciais correspondentes às versões exportadas a partir do software de horários.

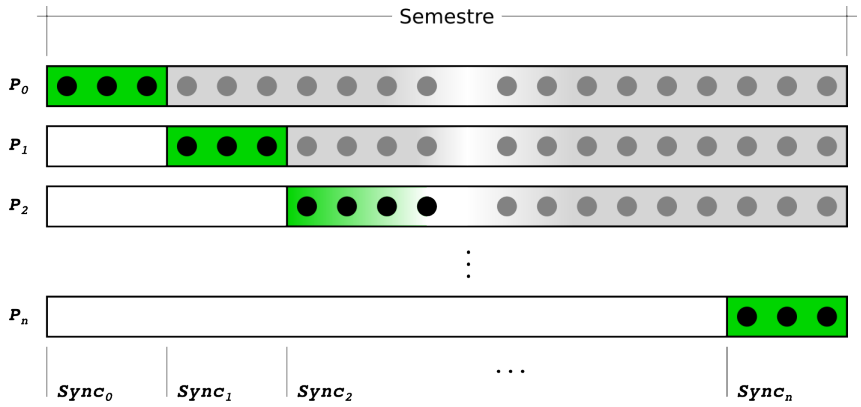


Figura 4.3: Sincronização de Períodos

No esquema da Figura 4.3 está descrito o caso genérico da sincronização de várias versões dos horários. A sincronização $Sync_0$ corresponde ao momento inicial do horário. Neste ponto apenas as aulas do período P_0 estão em vigor (círculos a negro). Quando é iniciada uma nova sincronização, as aulas anteriormente em vigor do período P_0 terão que ser anuladas (círculos a cinza), dando lugar às aulas no novo período P_1.

Todas as aulas a partir da data de entrada em vigor do período P_1 serão anuladas no período P_0, que coincide com o momento da sua sincronização $Sync_1$.

A anulação das aulas é conseguida colocando a *flag* `activo` com o valor 0 (ver secção 3.3.1). O procedimento é repetido para sucessivas sincronizações, resultando numa espécie de *meta-período* que consiste no somatório temporal dos períodos parciais.

4.3 Camada Intermédia

O desenvolvimento da camada intermédia é feita com recurso a *Web Servlets*. Estas *servlets* irão fornecer a implementação das interfaces do tipo `RemoteService` definidas na Camada Funcional (secção 3.3.2).

Implementação dos Serviços com recurso ao `GWT`

A implementação destas interfaces sob a forma de *servlets* é conseguida estendendo a funcionalidade da classe `RemoteServiceServlet` disponível na `API` do `GWT`. Os *servlets* deste tipo permitem ao utilizador usufruir da comunicação `AJAX` utilizando o protocolo `GWT XML-RPC`, sem que para isso seja necessária a criação de estruturas de código adicionais.

A titulo de exemplo, consideremos a seguinte interface de um serviço remoto `GWT`:

```
public interface Servico extends RemoteService {
        User criaUtilizador(String nome);
}
```

A cada interface de serviço `GWT` corresponde uma segunda interface, chamada assíncrona, que servirá de *proxy* para a invocação dos métodos do serviço na *servlet*:

```
public interface ServicoAsync {
        void criaUtilizador(String nome, AsyncCallback<User> callback);
}
```

Como a invocação é assíncrona, o seu resultado tem que ser obtido de forma assíncrona, ou seja, com recurso a um *callback*. A invocação assíncrona do serviço no código `GWT` do lado do cliente pode ser efectuado da seguinte forma:

```
ServicoAsync svc = GWT.create(Servico.class);
svc.criaUtilizador("luislobo", new AsyncCallback<User>() {
        public void onSuccess(User result) {
                // Fazer coisas com o utilizador
        }
        public void onFailure(Throwable caught) {
                // O pedido falhou, tratar do erro (caught)
        }
});
```

A *web servlet* correspondente do lado do servidor apenas necessitaria do seguinte código:

```
public class ServicoImpl extends RemoteServiceServlet implements Servico {
        public User criaUtilizador(String nome) {
                User u = ...;
                // Criar o utilizador
                return u;
        }
}
```

Autenticação e Autorização

Existe actualmente na instituição um directório centralizado de contas de utilizador baseado no protocolo LDAP (Lightweight Directory Access Protocol[Zeilenga, 2006]). A entidade responsável pela manutenção e gestão dos utilizadores neste directório é o CCOM (Centro de Comunicações).

A autenticação dos sumários utiliza em primeira instância a lista de utilizadores na entidade USERS para verificar se o utilizador tem acesso à plataforma e quais os módulos a que tem acesso, seguidamente consulta o servidor LDAP do CCOM para verificar se as credenciais fornecidas (nome de utilizador e password) estão correctas.

Para a comunicação com o servidor de LDAP foi utilizada a API JLDAP (LDAP Class Libraries for Java[1]). Esta API permite efectuar pedidos de autenticação (*binding*) a uma determinada conta de utilizador e pesquisas no directório LDAP.

A integração com os serviços de autenticação do CCOM, permite à plataforma de sumários independência na gestão dos utilizadores e níveis de acesso, usufruindo das credênciais definidas no LDAP.

Desta forma, para que os utilizadores de cada unidade orgância tenham acesso à plataforma dos sumários, basta serem adicionados no GAL (em conformidade com os dados no directório) e posteriormente sincronizados para a plataforma.

4.4 Interface Web

A Interface Web da aplicação dos sumários foi desenvolvida de forma a ir de encontro aos standards Web, com o objectivo de ser suportado nos principais browsers. Uma grande parte deste objectivo foi conseguida graças à framework de desenvolvimento GWT.

O GWT oferece um conjunto de componentes gráficos (*widgets*[2]) standard para o desenvolvimento de aplicações vocacionadas para a Web. A técnica de desenvolvimento de aplicações GWT, é semelhante à utilizada no desenvolvimento de aplicações com recurso à tecnologia Java Swing[3].

Os componentes gráficos são dispostos na página utilizando *layouts*, de forma que a página mantém um aspecto coerente, independentemente do tamanho.

A framework GWT permite a utilização de design patterns[4] (padrões de desenho) de aplicações gráficas. O desenvolvimento da interface web dos sumários, assentou na medida do possível, nestes padrões de desenho, nomeadamente:

[1] http://www.openldap.org/jldap/
[2] http://code.google.com/webtoolkit/doc/latest/RefWidgetGallery.html
[3] http://download.oracle.com/javase/tutorial/uiswing/
[4] http://www.javacamp.org/designPattern/

- `Observer Pattern`: Comunicação entre componentes gráficos e propagação de eventos em geral.
- `Proxy Pattern`: Na medida do possível, funcionalidade semelhante foi agregada pela utilização de interfaces e classes abstractas, permitindo várias implementações do mesmo conceito.
- `Singleton Pattern`: Funcionalidade repetida em vários pontos da aplicação foi agregada em `singletons` de forma a manter a coerência para procedimentos semelhantes e permitir a reutilização de código.

Apesar do desenvolvimento ser feito com base no Java, o resultado final da interface gráfica é uma página web suportada pela maioria dos browsers. O aspecto dos componentes gráficos é definido com recurso a folhas de estilos (`CSS`). A estilização pode ser feita na configuração dos componentes *inline* (no código Java) ou então pela injecção de folhas de estilo na aplicação.

A Interface Web dos Sumários

A página inicial da Interface Web dos sumários, apresenta uma caixa de diálogo de autenticação (Figura 4.4). Quando submetidas as credenciais, o servidor verifica a sua validade.

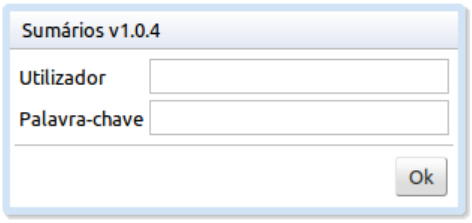


Figura 4.4: Caixa de Diálogo de Autenticação

Após autenticação do utilizador, é apresentado o ecrã principal da Interface Web. Este ecrã principal apresenta, numa fase inicial, um menu em árvore contendo os módulos a que o utilizador tem acesso, organizados por contexto (Figura 4.5).

Ao seleccionar um dos módulos, a zona central da página é substituída com os componentes gráficos do módulo solicitado. O carregamento do código correspondente dos módulos é feito de forma assíncrona utilizando uma facilidade do `GWT` chamada *code splitting*[1]. Desta forma a página não é carregada de uma forma monolítica, mas sim à medida que o utilizador os vai necessitando cada módulo.

[1] http://code.google.com/p/google-web-toolkit/wiki/CodeSplitting

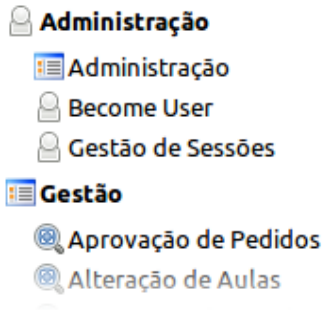


Figura 4.5: Parte do Menu do Utilizador

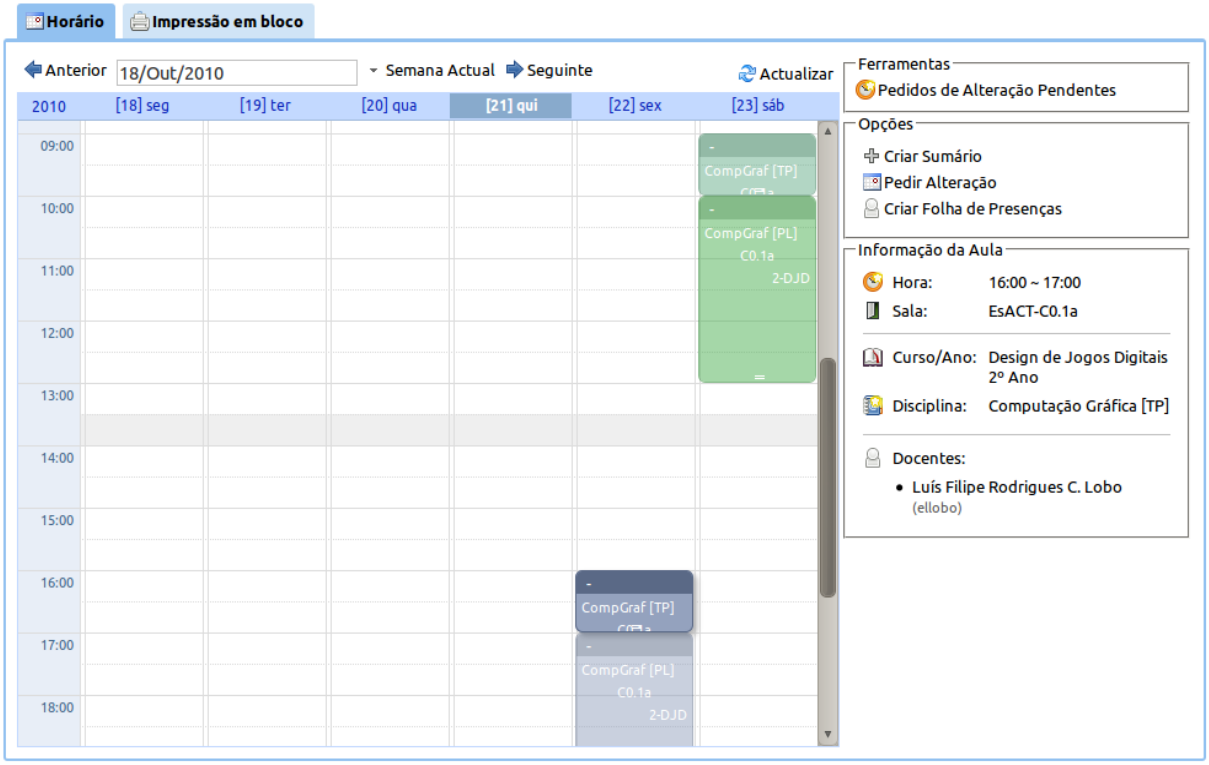
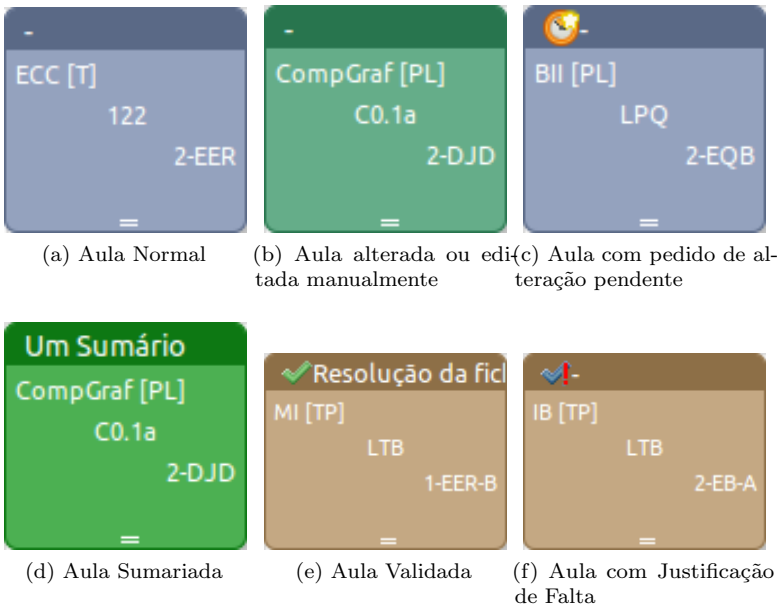


Figura 4.6: Horário do Docente



(a) Aula Normal (b) Aula alterada ou editada manualmente (c) Aula com pedido de alteração pendente

(d) Aula Sumariada (e) Aula Validada (f) Aula com Justificação de Falta

Figura 4.7: Blocos de Aula

O módulo de inserção de sumários tem o nome de "Sumários (docentes)" na Interface Web. Quando seleccionado, são carregados dois tabuladores correspondentes ao Horário e Impressão em Bloco. O Horário do docente apresenta a lista de aulas para a semana actual (Figura 4.6). A cor e o ícone de cada aula dependem do tipo e estado da aula, nomeadamente:

Do lado direito estão localizados um painel de Ferramentas, contendo opções variadas para o docente, um painel de Opções, contendo as operações disponíveis para a aula seleccionada no horário, e um painel de Informação da Aula, contendo a informação acerca da aula e sumário seleccionados. As opções do menu variam conforme a aula tem ou não sumário. Para o caso de a aula não estar sumariada, as opções são as mostradas na figura 4.6. As opções variam conforme o tipo de aula e o seu estado.

Na medida do possível foram utilizadas caixas de diálogo para introdução de dados, de forma a permitir ao utilizador manter o foco na aplicação principal (Figura 4.8)

Houve uma preocupação em manter a coerência do layout e o aspecto dos componentes em todos os módulos, apropriando apenas o conjunto de opções disponíveis à funcionalidade pretendida. As operações que implicam obtenção de dados a partir do servidor, foram sinalizadas com marcadores visuais de forma que o utilizador se aperceba do significado dos tempos de espera.

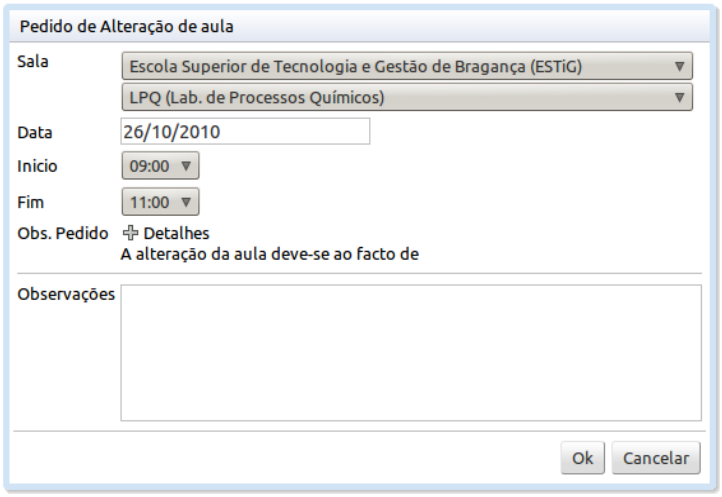


Figura 4.8: Caixa de Diálogo

4.5 Avaliação da Aplicação

À data de elaboração desta dissertação, os sumários encontram se em fase de produção no seu primeiro exercício, correspondendo ao primeiro semestre do ano lectivo de 2010/2011.

Antes da colocação em produção, foi designado um período de testes que decorreu durante o segundo semestre do ano lectivo de 2009/2010. Este período de testes contou com a colaboração de um conjunto de utilizadores com vários perfis das várias unidades orgânicas.

Foram ouvidas várias entidades com experiência e poder de decisão nas regras em volta do exercício dos sumários na instituição de forma a ir de encontro às necessidades de todos em geral, procurando colmatar as diferenças de cada unidade orgânica em particular.

Durante o período de testes, foram lançados novos desafios à plataforma, alguns dos quais foram implementados e resolvidos problemas de funcionamento. Quando a plataforma foi considerada estável e adequada às exigências iniciais, foi colocada em produção e disponibilizada à comunidade em geral.

Actualmente a aplicação encontra-se alojada no *Data Center* do Centro de Comunicações do IPB e está acessível aos utilizadores em geral na `URL`:

http://sumarios.ipb.pt/

4.5.1 Estatísticas de Utilização

Desde a sua colocação em produção, a plataforma de sumários registou um total de 16491 acessos (sessões), de entre os quais 3979 foram efectuados por alunos, resultando num

Escola	Ano Lectivo	Semestre	Aulas	Sumários
ESAB	2009	2	4472	2409
ESAB	2010	1	7189	988
ESEB	2009	2	9827	7542
ESEB	2010	1	9340	869
ESTiG	2009	2	7307	4969
ESTiG	2010	1	10024	2840
EsACT	2009	2	3239	3075
EsACT	2010	1	5038	1140
ESSB	2009	2	5557	1485
ESSB	2010	1	6404	1170

Tabela 4.1: Numero de Aulas e Sumários por semestre para cada escola

total de 428248 eventos (pedidos `AJAX`).

O número actual de utilizadores registados na plataforma é de 772[1], de entre os quais 12 são não-docentes.

O número total de aulas registadas na plataforma é de 68397, com um total de 26487 sumários inseridos e 110624 presenças marcadas.

A Tabela 4.1 lista o número de aulas e sumários por cada escola no segundo semestre de 2009 e no primeiro semestre de 2010.

Estes dados estatísticos permitem concluir que a adesão aos sumários está a ser favorável. Até ao momento, o numero de erros/sugestões enviadas utilizando a opção para o efeito na aplicação é de 118, no entanto, alguns utilizadores fazem este tipo de contacto utilizando a lista de suporte do Centro de Desenvolvimento. Por este motivo não é possível precisar a quantidade de pedidos de suporte.

[1] Os alunos não carecem de registo na plataforma

Capítulo 5

Conclusões e Trabalho Futuro

O registo de sumários e presenças representa uma ferramenta vital para as escolas. Os sumários representam o contrato celebrado entre os alunos e o docente para a transmissão do saber. A implementação de uma ferramenta informatizada à medida para o registo de sumários e presenças representa um passo importante na sua optimização.

A passagem de uma metodologia banal para o registo electrónico de sumários e presenças permite estabelecer uma interface comum e normalizada. O suporte digital para o armazenamento de sumários e presenças oferece um nível de acesso à informação nunca possível com a utilização do método convencional em papel.

Com a migração para uma solução electrónica obtém-se um nível elevado de disponibilidade, permitindo aos utilizadores uma experiência mais personalizada sem a burocracia inerente a este tipo de processos.

A abordagem ao desafio em desenvolver a plataforma de sumários foi feita de forma faseada.

Numa fase inicial foram recolhidos os requisitos necessários juntos das entidades com experiência na matéria. Depois de obtidos estes requisitos foram equacionados os vários caminhos possíveis para a implementação da solução, tendo em mente estruturas e solução já implementadas na instituição.

A decisão recaiu sobre a implementação de uma solução baseada na web, de forma a permitir aos utilizadores a sua utilização de forma ininterrupta. Foram escolhidas ferramentas standard para o desenvolvimento de forma a garantir um elevado nível de suporte.

Durante todo o processo de desenvolvimento, manteve-se a preocupação em proporcionar uma utilização intuitiva aos utilizadores pela introdução de controlos gráficos visualmente atractivos. Os procedimentos em volta do registo de sumários e presenças estão subjacentes na interface gráfica ainda que o utilizador não necessite de estar consciente deles, aumentando assim o nível de abstracção da aplicação.

Os sumários foram desenvolvidos com uma topologia multi-camada, permitindo a intervenção independente em cada uma delas sem prejuízo às demais. Esta topologia permite ainda que o desenvolvimento seja distribuído por uma equipa multidisciplinar.

A tecnologia utilizada é baseada na linguagem de programação Java, pelo suporte que esta tem junto da comunidade e pela quantidade disponível de facilidades vocacionadas para este tipo de plataforma. A interface gráfica foi desenvolvida com recurso ao Google Web Toolkit (`GWT`), por permitir o desenvolvimento em Java utilizando um conjunto de standards de desenvolvimento ainda que o resultado final seja uma página web. O sistema de base de dados escolhido foi o *Oracle Database 11g* por uma questão de integração com as restantes aplicações na instituição, das quais a aplicação de sumários depende.

Os sumários estão actualmente a ser utilizados por todas as escolas do IPB. A adesão dos docentes foi maciça. As escolas decidiram adoptar a plataforma como ferramenta para registo de sumários e presenças.

5.1 Trabalho Futuro

A plataforma de sumários encontra-se em franco desenvolvimento, com novas funcionalidades a serem adicionadas a cada momento. Este facto deve-se em parte à contribuição dos utilizadores com sugestões e notificações de erros.

O desenvolvimento futuro da plataforma segue no sentido de optimizar alguns processos e fornecer integração com outras aplicações, nomeadamente:

- Preenchimento automático de sumários com base na aplicação do Guia ECTS[1].
- Disponibilização dos sumários nas áreas das unidades curriculares do VIRTUAL.ipb[2]
- Criação conjunta com o VIRTUAL.ipb de turmas de alunos por forma a facilitar a marcação de presenças

Na altura da elaboração deste documento decorre um projecto que vai representar uma mais valia para a plataforma de sumários - O Registo Automático de Presenças.

Este projecto encontra-se actualmente em fase de implementação, tendo como escola piloto a Escola Superior de Tecnologia e de Gestão de Bragança.

O sistema consiste de um dispositivo de leitura RFID (Radio Frequency Identification[3]) colocado nas salas. Aos docentes e alunos serão fornecidos cartões de identificação providos da mesma tecnologia. A hora de entrada na sala de aula e os dados do docente/aluno serão automaticamente registados numa base de dados intermédia.

[1] http://guiaects.ipb.pt/
[2] http://virtual.ipb.pt/
[3] http://rfid.net/basics/190-what-is-rfid

O registo de presenças na plataforma de sumários poderá ser depois feito com base nos registos inseridos nessa base de dados.

Bibliografia

[Andersen, 2006] Andersen, L. (2006). JDBC 4.0 API Specification. JSR 221.

[Berners-Lee et al., 1994] Berners-Lee, T., Masinter, L., and McCahill, M. (1994). Uniform Resource Locators (URL). RFC 1738 (Proposed Standard). Obsoleted by RFCs 4248, 4266, updated by RFCs 1808, 2368, 2396, 3986.

[Crockford, 2006] Crockford, D. (2006). The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational).

[Fielding et al., 1999] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard). Updated by RFCs 2817, 5785.

[Hickson, 2009] Hickson, I. (2009). The web sockets API. W3C working draft, W3C. http://www.w3.org/TR/2009/WD-websockets-20091222/.

[Maler et al., 2004] Maler, E., Paoli, J., Sperberg-McQueen, C. M., Yergeau, F., and Bray, T. (2004). Extensible markup language (XML) 1.0 (third edition). first edition of a recommendation, W3C. http://www.w3.org/TR/2004/REC-xml-20040204.

[NIST, 1995] NIST (1995). An introduction do role-based access control. Technical report, NIST.

[Rescorla, 2000] Rescorla, E. (2000). HTTP Over TLS. RFC 2818 (Informational). Updated by RFC 5785.

[van Kesteren, 2009] van Kesteren, A. (2009). XMLHttpRequest. Last call WD, W3C. http://www.w3.org/TR/2009/WD-XMLHttpRequest-20091119/.

[Zeilenga, 2006] Zeilenga, K. (2006). Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map. RFC 4510 (Proposed Standard).

Apêndice A

Exemplos de Documentos

A.1 Folha de Presenças

INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Comunicação, Administração e Turismo

Licenciatura	Design de Jogos Digitais	Ano Curricular	2º	Semestre	1
Unidade Curricular	Computação Gráfica	Tipo de Aula	TP	Turma	

Aula N.º	0	Duração	16:00 ~ 17:00	Sala	C0.1a	N. Alunos Presentes		Ano Lectivo	2010/2011

Data: 08/10/2010

Assinatura do Docente:

(Luís Filipe Rodrigues C. Lobo)

PRESENÇAS

N.º Aluno	Nome	Assinatura
24572	Luís António Gonçalves Ribeiro	
23581	Miguel dos Santos Reis	
17120	Nuno André Guerra Pinhel	
24087	Paulo Jorge Gomes de Pinho	
24633	Pedro Manuel Eugénio Pinto	
23519	Pedro Miguel Vieira Fidalgo	
23520	Pedro Nuno dos Santos Geada	
23522	Rafael Xavier Vinagre Machado	
24759	Ricardo José Martins dos Santos	
23523	Ricardo Manuel Silva Nogueira	
24347	Samuel Adam Barbosa Sousa	
23525	Sergio Filipe Fernandes Assureira	
24763	Tiago Filipe Fraga Teixeira	
23527	Tiago Soares Fernandes	

A.2 Folha de Sumário

INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Comunicação, Administração e Turismo

Licenciatura	Design de Jogos Digitais	Ano Curricular	2º	Semestre	1
Unidade Curricular	Computação Gráfica	Tipo de Aula	TP	Turma	

Aula N.º		Duração	16:00 ~ 17:00	Sala	C0.1a	N. Alunos Presentes		Ano Lectivo	2010/2011

SUMÁRIO

- Apresentação da Unidade Curricular
- Introdução Histórica à Computação Gráfica

BIBLIOGRAFIA

Data: 08/10/2010

Assinatura do Docente: _____

(Luís Filipe Rodrigues C. Lobo)

Apêndice B

Código Fonte

B.1 Código Fonte dos Serviços

B.1.1 ImportService

```
package pt.ipb.sumarios.web.client.svc;

import java.util.List;

import pt.ipb.sumarios.entity.client.CursoPatchItem;
import pt.ipb.sumarios.entity.client.PeriodoImport;
import pt.ipb.sumarios.entity.client.SumariosException;

import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;

@RemoteServiceRelativePath("../svc/ImportServiceImpl")
public interface ImportService extends RemoteService {

        public static final class Util {
                public static ImportServiceAsync getInstance() {
                        return GWT.create(ImportService.class);
                }
        }

        void switchCodEscolaPeriodo(Integer idPeriodo, Integer oldCodEscola,
            Integer newCodEscola) throws SumariosException;

        void patchCodCurso(Integer idPeriodo, List<CursoPatchItem> items) throws
             SumariosException;

        List<PeriodoImport> getPeriodos() throws SumariosException;
```

```
        List<PeriodoImport> getPeriodos(boolean isNew) throws SumariosException;

        PeriodoImport getPeriodo(int idPeriodo) throws SumariosException;

        void importPeriodo(int idPeriodo) throws SumariosException;

        String getReport(Integer id);

        void sendImportSucccededNotification(Integer idPeriodo);

        void descartarPeriodo(Integer idPeriodo, String motivo) throws
            SumariosException;

}
```

B.1.2 HorarioService

```
package pt.ipb.sumarios.web.client.svc;

import java.util.List;

import pt.ipb.sumarios.entity.client.Aula;
import pt.ipb.sumarios.entity.client.Docente;
import pt.ipb.sumarios.entity.client.Sala;
import pt.ipb.sumarios.entity.client.SumariosException;
import pt.ipb.sumarios.entity.client.sa.SACurso;
import pt.lobo.gwt.user.client.date.StringDate;

import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;

@RemoteServiceRelativePath("../svc/HorarioServiceImpl")
public interface HorarioService extends RemoteService {

        public static final class Util {
                public static HorarioServiceAsync getInstance() {
                        return GWT.create(HorarioService.class);
                }
        }

        List<Docente> getDocenteList(Integer codEscola) throws SumariosException
            ;

        List<Aula> getAulaList(SACurso c, Docente d, Sala s, StringDate weekDate
            ) throws SumariosException;
```

```
        Aula createAula(Aula a) throws SumariosException;

        Aula updateAula(Aula a) throws SumariosException;

        void dropAula(Integer idAula) throws SumariosException;

        void notificarAula(Integer idAula) throws SumariosException;

}
```

B.1.3 StatsService

```
package pt.ipb.sumarios.web.client.svc;

import java.util.List;

import pt.ipb.sumarios.entity.client.AnoSemestre;
import pt.ipb.sumarios.entity.client.Presenca;
import pt.ipb.sumarios.entity.client.SumariosException;
import pt.ipb.sumarios.entity.client.sa.SAUnidadeCurricular;
import pt.ipb.sumarios.entity.client.stats.Stats0Item;
import pt.ipb.sumarios.web.client.PagedList;

import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;

@RemoteServiceRelativePath("../svc/StatsServiceImpl")
public interface StatsService extends RemoteService {

        public class Util {
                public static StatsServiceAsync getInstance() {
                        return GWT.create(StatsService.class);
                }
        }

        List<Presenca> getPresencaList(SAUnidadeCurricular uc, Integer anoLect,
            Integer semestre) throws SumariosException;

        PagedList<Stats0Item> stats0(Integer codEscola, Integer codCurso,
            Integer nPlano,
                        Integer nDisciplina, Integer nOpcao, String
                            abrevTipoAula,
                        AnoSemestre as, String docente, String orderBy, Integer
                            offset, Integer limit) throws SumariosException;

}
```

B.1.4 UserService

```
package pt.ipb.sumarios.web.client.svc;

import java.util.List;

import pt.ipb.sumarios.entity.client.Docente;
import pt.ipb.sumarios.entity.client.SumariosException;
import pt.ipb.sumarios.entity.client.sa.SAAluno;
import pt.ipb.sumarios.entity.client.session.SessionInfo;
import pt.ipb.sumarios.entity.client.session.SessionLogEntry;
import pt.ipb.sumarios.entity.client.session.SessionInfo.SessionStatus;
import pt.ipb.sumarios.web.client.PagedList;
import pt.ipb.sumarios.web.client.User;
import pt.lobo.gwt.user.client.date.StringTimestamp;

import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;

@RemoteServiceRelativePath("../svc/UserServiceImpl")
public interface UserService extends RemoteService {

        public static final class Util {
                public static UserServiceAsync getInstance() {
                        return GWT.create(UserService.class);
                }
        }

        List<User> getUserList(String filter, Integer codEscola) throws
            SumariosException;

        User addUser(User user, Integer codEscola, String[] modules) throws
            SumariosException;

        void removeUser(String login) throws SumariosException;

        List<User> searchUser(String filter) throws SumariosException;

        List<Docente> searchDocente(String filter) throws SumariosException;

        List<SAAluno> searchAluno(String filter, Integer codEscola, Integer
            codCurso, Integer nPlano, int maxResult) throws SumariosException;

        List<SAAluno> searchAluno(String filter, int maxResult) throws
            SumariosException;
```

```
        List<SAAluno> getAlunosInscritos(Integer idDetalhesAula) throws
            SumariosException;

        /**
         * Procura alunos que ja estiveram presentes na turma a que se refere o
             sumario
         * no mesmo ano
         * @param idSumarioRef Sumario tomado como referencia para comparacao
         * @return A lista de alunos com pelo menos uma presenca no sumario
         * @throws SumariosException
         */
        List<SAAluno> getAlunosComPresenca(Integer idSumarioRef) throws
            SumariosException;

        List<SAAluno> getAlunosComPresencaDA(Integer idDetalhesAula) throws
            SumariosException;

        /**
         * Obtem a lista de alunos presentes na ultima aula
         * @param idSumarioRef Sumario de
         * @return
         * @throws SumariosException
         */
        List<SAAluno> getAlunosUltimaAula(Integer idSumarioRef) throws
            SumariosException;

        List<SAAluno> getAlunosUltimaAulaDA(Integer idDetalhesAula) throws
            SumariosException;

        /**
         * Soncroniza a tabela docentes/users para que os docentes que vem do
             gal possam entrar na plataforma
         * @param modules Os modulos atribuidos por omissao aos docentes
         * @return Lista de docentes que foram sincronizados
         * @throws SumariosException
         */
        List<Docente> syncDocentes(List<String> modules) throws
            SumariosException;

        /**
         * @return A contagem de docentes que existem na tabela docentes e nao
             existem na tabela users
         * @throws SumariosException
         */
        Integer getSyncDocentesCount() throws SumariosException;

        List<Docente> getSyncDocentesList() throws SumariosException;
```

```
        // Session

        List<User> getSessionUsers() throws SumariosException;

        List<SessionInfo> getSessionList(String username, StringTimestamp from,
            StringTimestamp to, SessionStatus status) throws SumariosException;

        PagedList<SessionLogEntry> getSessionEvents(String sessionId, int offset
            , int limit) throws SumariosException;

}
```

B.1.5 SumariosService

```
package pt.ipb.sumarios.web.client.svc;

import java.util.List;
import java.util.Map;

import pt.ipb.sumarios.entity.client.AnoSemestre;
import pt.ipb.sumarios.entity.client.Aula;
import pt.ipb.sumarios.entity.client.DetalhesAula;
import pt.ipb.sumarios.entity.client.Docente;
import pt.ipb.sumarios.entity.client.JustificacaoFalta;
import pt.ipb.sumarios.entity.client.Periodo;
import pt.ipb.sumarios.entity.client.ReScheduleRequest;
import pt.ipb.sumarios.entity.client.ReScheduleStatus;
import pt.ipb.sumarios.entity.client.Sala;
import pt.ipb.sumarios.entity.client.SalaId;
import pt.ipb.sumarios.entity.client.Sumario;
import pt.ipb.sumarios.entity.client.SumarioInfo;
import pt.ipb.sumarios.entity.client.SumariosException;
import pt.ipb.sumarios.entity.client.SumariosModule;
import pt.ipb.sumarios.entity.client.sa.SAAluno;
import pt.ipb.sumarios.entity.client.sa.SACurso;
import pt.ipb.sumarios.entity.client.sa.SAUnidadeCurricular;
import pt.lobo.gwt.user.client.date.StringDate;
import pt.lobo.gwt.user.client.date.StringTimestamp;

import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;
import com.google.gwt.user.client.rpc.StatusCodeException;

@RemoteServiceRelativePath("../svc/SumariosServiceImpl")
//@Remote
public interface SumariosService extends RemoteService {
```

```
public static final class Util {
        public static SumariosServiceAsync getInstance() {
                return GWT.create(SumariosService.class);
        }
}

List<SumarioInfo> getSumarioInfoList(Integer idAula) throws
    SumariosException;

/**
 * @param idDetalhesAula
 * @return Numero sugerido para o proximo sumario para a disciplina
     escolhida
 * @throws SumariosException
 */
Integer sugereNum(Integer idDetalhesAula) throws SumariosException;

Aula invalidaAula(Integer idAula, String motivo) throws
    SumariosException;

/**
 * Valida todos os sumarios de uma determinada aula
 * @param idAula ID da aula
 * @throws SumariosException
 */
Aula validaAula(Integer idAula, Boolean justificacao, String
    justificacaoObs) throws SumariosException;

/**
 * Valida uma determinado sumario
 * @param id O ID do sumario
 * @return A validacao
 * @throws SumariosException
 */
void validaSumario(Integer id, Boolean justificacao, String
    justificacaoObs) throws SumariosException;


/**
 * @return Os anos quem tem eventos para a escola do utilizador
 * @throws SumariosException
 */
List<AnoSemestre> getAnoSemestreListEscola() throws SumariosException;

/**
 * @param docente O Docente
```

```
     * @return Todos os anos/semestres em que o docente tem eventos
     * @throws SumariosException
     */
    List<AnoSemestre> getAnoSemestreList(String docente) throws
        SumariosException;

    void enviarSugestao(String moduleId, String moduleName, String info,
        String subject, String message) throws SumariosException;

    /**
     * Obter a lista de anos que tem sumarios inseridos para as disdciplinas
         do aluno
     * @param codAluno O n. mec do aluno
     * @return A lista de anos
     * @throws SumariosException
     */
    List<Integer> getAnosAluno(Integer codAluno) throws SumariosException;

    /**
     * @param docente
     * @return A lista de anos em que o docente tem distribuicao de servico
     * @throws SumariosException
     */
    List<AnoSemestre> getAnosDocente(String docente) throws
        SumariosException;

    /**
     * Obter a lista de cursos em que o aluno se encontra inscrito no ano
         escpecificado
     * @param codAluno O n. mec do aluno
     * @param anoLect O ano lectivo para as inscricoes
     * @param hasSumario Quando @c TRUE, apenas cursos que tenham sumario
         para o aluno serao retornados
     * @return
     * @throws SumariosException
     */
    List<SACurso> getCursosAluno(Integer codAluno, Integer anoLect, boolean
        hasSumario) throws SumariosException;

    /**
     * Obter a lista de disciplinas para um determinado aluno/curso num ano
         lectivo
     * @param codAluno O n. mec do aluno
     * @param codEscola A escola
     * @param codCurso Curso
     * @param nPlano Plano
     * @param anoLect Ano lectivo
```

```
         * @param semestre Semestre
         * @param hasSumario Quando @c TRUE, apenas disciplinas que tenham
             sumario para o aluno serao devolvidas
         * @return A lista de disciplinas
         * @throws SumariosException
         */
        List<SAUnidadeCurricular> getDisciplinasAluno(Integer codAluno, Integer
            codEscola, Integer codCurso, Integer nPlano, Integer anoLect,
            Integer semestre, boolean hasSumario) throws SumariosException;

        /**
         * Obtem a lista de sumarios para o aluno / disciplina
         * @param codAluno O n. mec do aluno
         * @param uc A disciplina
         * @param anoLect TODO
         * @param presente Quando @c TRUE, apenas sumarios em que o aluno esteve
              presente sao devolvidos
         * @return A lista de sumarios
         * @throws SumariosException
         */
        List<SumarioInfo> getSumariosAluno(Integer codAluno, SAUnidadeCurricular
             uc, Integer anoLect, boolean presente) throws SumariosException;

        /**
         * Obter a lista de periodos para um determinado docente, ou todos os
             periodos no caso de
         * o parametro docente ser @c null
         * @param docente O docente
         * @return A lista de periodos
         * @throws SumariosException
         */
        List<Periodo> getPeriodoList(String docente) throws SumariosException;

        /**
         * Obter a lista de periodos para uma determinada escola
         * @param codEscola O ID da escola
         * @return A lista de periodos
         * @throws SumariosException
         */
        List<Periodo> getPeriodoList(Integer codEscola) throws SumariosException
            ;

        /**
         * Obter a lista de aulas para um determinado docente num determinado
             periodo
         * @param docente O docente
         * @param weekDate Data da semana para a lista de aulas
```

```
         * @return A lista de aulas
         * @throws SumariosException
         */
        List<Aula> getAulaList(String docente, StringDate weekDate) throws
            SumariosException;

        /**
         * Obter a lista de aulas para um determinado docente num determinado
             periodo
         * @param docente O docente
         * @param weekDate Data da semana para a lista de aulas
         * @param detalhesList @c TRUE para obter tambem a lista de detalhes das
              aulas (@c Aula.getDetalhesAula())
         * @return A lista de aulas
         * @throws SumariosException
         */
        List<Aula> getAulaList(String docente, StringDate weekDate, boolean
            detalhesList) throws SumariosException;

        /**
         * Obter a lista de detalhes para uma aula
         * @param idAula ID da aula
         * @return Os detalhes_aula solicitados
         * @throws SumariosException
         */
        List<DetalhesAula> getDetalhesAulaList(Integer idAula) throws
            SumariosException;

        /**
         * Obter o sumario para o detalhes aula
         * @param idDetalhesAula O ID do detalhes aula
         * @param synthetic Quando @c TRUE trunca o sumario a 30 caracteres
         * @return O sumario solicitado
         * @throws SumariosException
         */
        List<Sumario> getSumarioList(Integer idDetalhesAula, boolean synthetic)
            throws SumariosException;

        /**
         * Obter um determinado sumario
         * @param id ID do sumario
         * @return O sumario solicitado
         * @throws SumariosException
         */
        Sumario getSumario(Integer id) throws SumariosException;

        /**
```

```
         * Obter uma determinada aula
         * @param id O ID da aula a obter
         * @param detalhesList @c TRUE para obter tambem os detalhes das aulas
         * @return A aula especificada
         * @throws SumariosException
         */
        Aula getAula(Integer id, boolean detalhesList) throws SumariosException;

        /**
         * Obter os detalhes das aulas para uma determinada aula
         * @param id O ID da aula
         * @param sumariosList @c TRUE para obter a lista de sumarios para cada
             detalhe de aula
         * @return A lista de detalhes aula especificada
         * @throws SumariosException
         */
        DetalhesAula getDetalhesAula(Integer idAula, boolean sumariosList)
            throws SumariosException;

        /**
         * Cria o mesmo sumarios para todos os DetalhesAula de uma aula, a data/
             tipo/inicio/fim sao copiados
         * dos detalhes_aula
         * @param idAula ID da aula
         * @param sumario O sumario
         * @throws SumariosException
         */
        void fastCreateSumario(Integer idAula, String sumario) throws
            SumariosException;

        /**
         * Cria um sumario
         * @param idDetalhesAula ID do detalhe da aula
         * @param sumario O texto do sumario
         * @param inicio A hora de inicio para o sumario
         * @param fim A hora de fim para o sumario
         * @param abrevTipoAula Abreviatura do tipo de aula
         * @param nomeTipoAula Nome do tipo de aula
         * @param bibliografia Bibliografia
         * @return O sumario acabado de criar
         * @throws SumariosException
         */
        Sumario createSumario(Integer idDetalhesAula, String sumario,
            StringTimestamp inicio, StringTimestamp fim, String abrevTipoAula,
            String nomeTipoAula, List<String> bibliografia, Integer numero)
            throws SumariosException;
```

```
        /**
         * Cria um sumario
         * @param sumario O sumario
         * @return O sumario acabado de criar
         * @throws SumariosException
         */
        Sumario createSumario(Sumario sumario) throws SumariosException;

        /**
         * Anula todos os sumarios para uma determinada aula
         * @param idAula O ID da aula
         * @throws SumariosException
         */
        void anulaSumarios(Integer idAula) throws SumariosException;

        /**
         * Substituir os sumarios de um detalhe de aula
         * @param idDetalhesAula O ID do detalhe de aula
         * @param replacement A nova lista de sumarios
         * @return A lista de sumarios criados
         * @throws SumariosException
         */
        List<Sumario> replaceSumarios(Integer idDetalhesAula, List<Sumario>
            replacement) throws SumariosException;

        /**
         * Substituir os sumarios de uma aula
         * @param sumarios Mapa <DetalhesAula.id, List<Sumario>>
         * @throws SumariosException
         */
        void replaceSumarios(Map<Integer, List<Sumario>> sumarios) throws
            SumariosException;

        /**
         * @param sumarioId ID do sumario
         * @return A bibliografia para um determinado sumario
         * @throws SumariosException
         */
        List<String> getBibliografia(Integer sumarioId) throws SumariosException
            ;

        /**
         * Adicionar uma presenca a um sumario
         * @param idSumario O ID do sumario
         * @param codAluno O login do aluno
         * @throws SumariosException
         */
```

```
        void addPresenca(Integer idSumario, Integer codAluno) throws
            SumariosException;

        /**
         * Remover uma presenca de um sumario
         * @param idSumario O ID do sumario
         * @param codAluno O login do aluno
         * @throws SumariosException
         */
        void removePresenca(Integer idSumario, Integer codAluno) throws
            SumariosException;

        /**
         * @param idSumario
         * @param codAluno
         * @throws SumariosException
         */
        boolean isPresenca(Integer idSumario, Integer codAluno) throws
            SumariosException;

        /**
         * Obter a lista de presencas para um determinado sumario
         * @param idSumario ID do sumario
         * @return A lista de n. de alunos presentes no sumario
         * @throws SumariosException
         */
        List<SAAluno> getPresencaList(Integer idSumario) throws
            SumariosException;

        /**
         * Altera a lista de presencas para um determinado sumario
         * @param alunoPresencaMap Mapa [idSumario => list(cod_aluno*)]
         * @param alunoCountMap TODO
         * @throws SumariosException
         */
        void setPresencas(Map<Integer /* idSumario */, List<Integer> /* [
            cod_aluno] */> alunoPresencaMap, Map<Integer /* idSumario */,
            Integer /* aluno_count */> alunoCountMap) throws SumariosException;

        /**
         * Alterar as presencas para um determinado sumario
         * @param idSumario ID do sumario
         * @param codAlunos Os codigos dos alunos
         * @param alunoCount TODO
         * @throws SumariosException
         */
```

```
    void setPresencas(Integer idSumario, List<Integer> codAlunos, Integer
        alunoCount) throws SumariosException;

    /**
     * @param idPeriodo ID do periodo
     * @return O periodo seleccionado
     * @throws SumariosException
     */
    Periodo getPeriodo(Integer idPeriodo) throws SumariosException;

    /**
     * @return A lista de todos os modulos disponiveis
     * @throws SumariosException
     */
    List<SumariosModule> getModuleList() throws SumariosException;

    /**
     * Obter a lista de modulos activos para o utilizador
     * @param user Utilizador
     * @return A lista de modulos
     * @throws SumariosException
     */
    List<SumariosModule> getModulesForUser(String user) throws
        SumariosException;

    /**
     * Alterar a permissao de um modulo para o utilizador
     * @param user O utilizador
     * @param moduleId O ID do modulo
     * @param enabled O estado
     * @throws SumariosException
     */
    void setModuleEnabled(String user, String moduleId, boolean enabled)
        throws SumariosException;

    /**
     * Definir os modulos de um utilizador
     * @param user O login do utilizador
     * @param modules A lista de modulos
     * @throws SumariosException
     */
    void setUserModules(String user, List<String> modules) throws
        SumariosException;

    /**
     * Verificar se o utilizador tem acesso ao modulo
     * @param user Utilizador
```

```
         * @param moduleId O ID do modulo
         * @return Se o utilizador tem acesso
         * @throws SumariosException
         */
        boolean isModuleEnabled(String user, String moduleId) throws
            SumariosException;

        /**
         * @return A lista de todos os docentes
         * @throws SumariosException
         */
        List<Docente> getDocentesList() throws SumariosException;

        /**
         * Obter a lista de docentes para a escola do supervisor
         * @return A lista de docentes
         * @throws SumariosException
         */
        List<Docente> getDocentesEscolaList() throws SumariosException;

        /**
         * @param login O login
         * @return O docente com o login especificado
         * @throws SumariosException
         */
        Docente getDocente(String login) throws SumariosException;

        // Supervisao

        /**
         * Lista de periodos a que o docente tem acesso para supervisao
         * @return A lista de periodos
         * @throws SumariosException
         */
        List<Periodo> getPeriodosSupervisao() throws SumariosException;

        /**
         * Lista de Cursos que teem sumarios inseridos para um determinado
             periodo
         * @return A lista de cursos
         * @throws SumariosException
         */
        List<SACurso> getCursosComSumario() throws SumariosException;

        /**
         * Lista de disciplinas com sumarios inseridos para um determinado curso
             /periodo
```

```
         * @param codEscola ID da escola
         * @param codCurso ID do curso
         * @param nPlano ID do plano
         * @return A lista de disciplinas solicitada
         * @throws SumariosException
         */
        List<SAUnidadeCurricular> getDisciplinasComSumario(Integer codEscola,
            Integer codCurso, Integer nPlano) throws SumariosException;

        /**
         * Obter a lista de Sumarios com base na unidade curricular
         * @param codEscola ID da Escola
         * @param codCurso ID do curso
         * @param nPlano ID do plano
         * @param nDisciplina ID da disciplina
         * @param nOpcao ID da opcao
         * @param turma TODO
         * @return A lista de sumarios
         * @throws SumariosException
         */
        List<SumarioInfo> getSumarioList(Integer codEscola, Integer codCurso,
            Integer nPlano, Integer nDisciplina, Integer nOpcao, String turma)
            throws SumariosException;

        // ReSchedule

        /**
         * Efectuar um pedido de alteracao de Aula
         * @param idAula ID da aula a alterar
         * @param newDate A nova data para a aula
         * @param inicio O inicio
         * @param fim O fim
         * @param newIdSala A nova sala para a aula
         * @param obs Observacoes do pedido (razao)
         * @return O pedido criado
         * @throws SumariosException
         */
        ReScheduleRequest requestReschedule(Integer idAula, StringDate newDate,
            StringTimestamp inicio, StringTimestamp fim, SalaId newIdSala,
            String obs) throws SumariosException;

        /**
         * Processar um pedido de alteracao de aula
         * @param id ID do pedido
         * @param obs Observacoes do resultado
         * @param result O resultado
         * @throws SumariosException
```

```
     */
    void processReschedule(Integer id, String obs, ReScheduleStatus result)
        throws SumariosException;

    void approveWithChanged(Integer id, StringDate newDate, StringTimestamp
        inicio, StringTimestamp fim, SalaId newIdSala, String obs) throws
        SumariosException;

    /**
     * @param id ID do pedido de alteracao
     * @return O pedido de alteracao solicitado
     * @throws SumariosException
     */
    ReScheduleRequest getReScheduleRequest(Integer id) throws
        SumariosException;

    /**
     * @param status NULL to retrieve all
     * @param supervisor O login do supervisor (serve para filtrar as
         escolas)
     * @return
     * @throws SumariosException
     */
    List<ReScheduleRequest> getReScheduleRequestList(ReScheduleStatus status
        , String supervisor) throws SumariosException;

    /**
     * @param status O estado do pedido
     * @return A lista de pedidos de alteracao para o utilizador actual
     * @throws SumariosException
     */
    List<ReScheduleRequest> getReScheduleRequestList(ReScheduleStatus status
        ) throws SumariosException;

    /**
     * @param id ID do pedido de alteracao
     * @throws SumariosException
     */
    void cancelReScheduleRequest(Integer id) throws SumariosException;

    /**
     * @return A lista de todas as salas para a escola do utilizador actual
     * @throws SumariosException
     */
    List<Sala> getSalaList() throws SumariosException;

    /**
```

```
         * A lista de salas para a escola
         * @param codEscola O codigo da escola
         * @return A lista de salas solicitada
         * @throws SumariosException
         */
        List<Sala> getSalaList(Integer codEscola) throws SumariosException;

        /**
         * Obter os nomes dos alunos com base no codigo dos servicos academicos
         * @param codAlunos A lista de cod_alunos a obter
         * @return Os nomes do alunos
         * @throws SumariosException
         */
        Map<Integer/* codAluno */,String/* nome */> getAlunosNomes(List<Integer>
            codAlunos) throws SumariosException;

        /**
         * Unidades Curriculares leccionadas pelo docente
         * @param docente
         * @return
         * @throws SumariosException
         */
        List<SAUnidadeCurricular> getUnidadesCurriculares(String docente,
            Integer anoLect, Integer semestre) throws SumariosException;

        /**
         * Obter a justificacao de falta para uma determinada aula, se existir
         * @param idAula O ID da aula
         * @return A justificacao solicitada
         * @throws SumariosException
         */
        JustificacaoFalta getJustificacaoFalta(Integer idAula) throws
            SumariosException;

}
```

B.1.6 SessionService

```
package pt.ipb.sumarios.web.client.svc;

import pt.ipb.sumarios.web.client.User;

import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;
import com.google.gwt.user.client.rpc.SerializationException;

@RemoteServiceRelativePath("../svc/SessionServiceImpl")
```

```
public interface SessionService extends RemoteService {

        public static class Util {
                public static SessionServiceAsync getInstance() {
                        return GWT.create(SessionService.class);
                }
        }

        User remoteUser();

        void logout();

        User login(String login, String passwd) throws SerializationException;

        User becomeUser(String login) throws SerializationException;

        long checkSession() throws SerializationException;

}
```

B.2 Código Fonte das Entidades

B.2.1 User

```
package pt.ipb.sumarios.web.client;

import java.io.Serializable;

import pt.ipb.sumarios.entity.client.annotations.Id;

@SuppressWarnings("serial")
public class User implements Serializable {

        @Id
        private String login;

        private String nome;

        private String email;

        private Integer codEscola;

        private Integer codAluno = null;

        private boolean aluno = false;

        public User() {
```

```
	}

	public String getLogin() {
		return login;
	}

	public void setLogin(String login) {
		this.login = login;
	}

	public String getNome() {
		return nome;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	public Integer getCodEscola() {
		return codEscola;
	}

	public void setCodEscola(Integer codEscola) {
		this.codEscola = codEscola;
	}

	public boolean isAluno() {
		return aluno;
	}

	public void setAluno(boolean aluno) {
		this.aluno = aluno;
	}

	public Integer getCodAluno() {
		return codAluno;
	}

	public void setCodAluno(Integer codAluno) {
```

```
                this.codAluno = codAluno;
        }

}
```

B.2.2 SumarioInfo

```
package pt.ipb.sumarios.entity.client;

import java.io.Serializable;
import java.util.List;

import pt.ipb.sumarios.entity.client.sa.SAAluno;
import pt.ipb.sumarios.web.client.User;

@SuppressWarnings("serial")
public class SumarioInfo implements Serializable {

        private Sumario sumario;

        private DetalhesAula detalhesAula;

        private Aula aula;

        private Periodo periodo;

        private User docente;

        private List<SAAluno> presencas;

        private boolean alunoPresente;

        public SumarioInfo() {

        }

        public Sumario getSumario() {
                return sumario;
        }

        public void setSumario(Sumario sumario) {
                this.sumario = sumario;
        }

        public DetalhesAula getDetalhesAula() {
                return detalhesAula;
        }
```

```
        public void setDetalhesAula(DetalhesAula detalhesAula) {
                this.detalhesAula = detalhesAula;
        }

        public Aula getAula() {
                return aula;
        }

        public void setAula(Aula aula) {
                this.aula = aula;
        }

        public Periodo getPeriodo() {
                return periodo;
        }

        public void setPeriodo(Periodo periodo) {
                this.periodo = periodo;
        }

        public User getUser() {
                return docente;
        }

        public void setUser(User docente) {
                this.docente = docente;
        }

        public boolean isAlunoPresente() {
                return alunoPresente;
        }

        public void setAlunoPresente(boolean alunoPresente) {
                this.alunoPresente = alunoPresente;
        }

        public List<SAAluno> getPresencas() {
                return presencas;
        }

        public void setPresencas(List<SAAluno> alunos) {
                this.presencas = alunos;
        }

}
```

B.2.3 ReScheduleRequest

```
package pt.ipb.sumarios.entity.client;

import java.io.Serializable;

import pt.ipb.sumarios.entity.client.annotations.Id;
import pt.ipb.sumarios.web.client.User;
import pt.lobo.gwt.user.client.date.StringDate;
import pt.lobo.gwt.user.client.date.StringTimestamp;

@SuppressWarnings("serial")
public class ReScheduleRequest implements Serializable {

        @Id
        private Integer id;

        private User user;

        private Aula aula;

        private StringDate data;

        private StringTimestamp inicio;

        private StringTimestamp fim;

        private SalaId salaId;

        private String obsPedido;

        private String obsResultado;

        private ReScheduleStatus resultado;

        private StringDate oldData;

        private StringTimestamp oldInicio;

        private StringTimestamp oldFim;

        private SalaId oldSalaId;

        private StringTimestamp dataPedido;

        private StringTimestamp dataResultado;

        private String changedby;
```

```
        private Integer refId;

        public ReScheduleRequest() {
        }

        public Integer getId() {
                return id;
        }

        public void setId(Integer id) {
                this.id = id;
        }

        public User getUser() {
                return user;
        }

        public void setUser(User docente) {
                this.user = docente;
        }

        public Aula getAula() {
                return aula;
        }

        public void setAula(Aula subject) {
                this.aula = subject;
        }

        public StringDate getData() {
                return data;
        }

        public void setData(StringDate data) {
                this.data = data;
        }

        public StringTimestamp getInicio() {
                return inicio;
        }

        public void setInicio(StringTimestamp inicio) {
                this.inicio = inicio;
        }

        public StringTimestamp getFim() {
                return fim;
```

```
    }

    public void setFim(StringTimestamp fim) {
        this.fim = fim;
    }

    public SalaId getSalaId() {
        return salaId;
    }

    public void setSalaId(SalaId salaId) {
        this.salaId = salaId;
    }

    public String getObsPedido() {
        return obsPedido;
    }

    public void setObsPedido(String obsPedido) {
        this.obsPedido = obsPedido;
    }

    public String getObsResultado() {
        return obsResultado;
    }

    public void setObsResultado(String obsResultado) {
        this.obsResultado = obsResultado;
    }

    public ReScheduleStatus getResultado() {
        return resultado;
    }

    public void setResultado(ReScheduleStatus resultado) {
        this.resultado = resultado;
    }

    public StringDate getOldData() {
        return oldData;
    }

    public void setOldData(StringDate oldData) {
        this.oldData = oldData;
    }

    public StringTimestamp getOldInicio() {
```

```
        return oldInicio;
    }

    public void setOldInicio(StringTimestamp oldInicio) {
        this.oldInicio = oldInicio;
    }

    public StringTimestamp getOldFim() {
        return oldFim;
    }

    public void setOldFim(StringTimestamp oldFim) {
        this.oldFim = oldFim;
    }

    public SalaId getOldSalaId() {
        return oldSalaId;
    }

    public void setOldSalaId(SalaId oldSalaId) {
        this.oldSalaId = oldSalaId;
    }

    public StringTimestamp getDataPedido() {
        return dataPedido;
    }

    public void setDataPedido(StringTimestamp dataPedido) {
        this.dataPedido = dataPedido;
    }

    public StringTimestamp getDataResultado() {
        return dataResultado;
    }

    public void setDataResultado(StringTimestamp dataResultado) {
        this.dataResultado = dataResultado;
    }

    public String getChangedby() {
        return changedby;
    }

    public void setChangedby(String changedby) {
        this.changedby = changedby;
    }
```

```
        public Integer getRefId() {
                return refId;
        }

        public void setRefId(Integer refId) {
                this.refId = refId;
        }

}
```

B.2.4 SalaId

```
package pt.ipb.sumarios.entity.client;

import java.io.Serializable;

import pt.ipb.sumarios.entity.client.annotations.Id;

@SuppressWarnings("serial")
public class SalaId implements Serializable {

        @Id
        private String codSala;

        @Id
        private Integer codEscola;

        public SalaId() {
        }

        public SalaId(Integer codEscola, String codSala) {
                super();
                this.codEscola = codEscola;
                this.codSala = codSala;
        }

        public String getCodSala() {
                return codSala;
        }

        public void setCodSala(String codSala) {
                this.codSala = codSala;
        }

        public Integer getCodEscola() {
                return codEscola;
        }
```

```
        public void setCodEscola(Integer codEscola) {
                this.codEscola = codEscola;
        }

        @Override
        public int hashCode() {
                return toString().hashCode();
        }

        @Override
        public boolean equals(Object obj) {
                if (obj instanceof SalaId) {
                        SalaId o = (SalaId) obj;

                        if(o.codEscola == null || codEscola == null || codSala
                            == null || o.codSala==null)
                                return false;

                        return
                                ( o.codEscola.equals(this.codEscola) ) &&
                                ( o.codSala.equals(this.codSala) );
                }
                return false;
        }

        @Override
        public String toString() {
                StringBuilder sb = new StringBuilder();
                sb.append(codEscola).append("-").append(codSala);
                return sb.toString();
        }

}
```

B.2.5 Sumario

```
package pt.ipb.sumarios.entity.client;

import java.io.Serializable;
import java.util.List;

import pt.ipb.sumarios.entity.client.annotations.Id;
import pt.ipb.sumarios.entity.client.annotations.ReadOnly;
import pt.lobo.gwt.user.client.date.StringDate;
import pt.lobo.gwt.user.client.date.StringTimestamp;

@SuppressWarnings("serial")
public class Sumario implements Serializable {
```

```
@Id
private Integer id;

private String sumario;

private Integer idDetalhesAula;

private StringDate changedDate;

private String changedBy;

private StringTimestamp inicio;

private StringTimestamp fim;

private String abrevTipoAula;

private String nomeTipoAula;

private List<String> bibliografia;

private Integer numero;

@ReadOnly
private Integer idValidacao;

@ReadOnly
private StringDate dataValidacao;

@ReadOnly
private String loginValidacao;

@ReadOnly
private Integer alunoCount;

public Sumario() {
}

public Integer getId() {
        return id;
}

public void setId(Integer idSumario) {
        this.id = idSumario;
}
```

```
        public String getSumario() {
                return sumario;
        }

        public void setSumario(String sumario) {
                this.sumario = sumario;
        }

        public Integer getIdDetalhesAula() {
                return idDetalhesAula;
        }

        public void setIdDetalhesAula(Integer idDetalhesAula) {
                this.idDetalhesAula = idDetalhesAula;
        }

        public StringDate getChangedDate() {
                return changedDate;
        }

        public void setChangedDate(StringDate changedDate) {
                this.changedDate = changedDate;
        }

        public String getChangedBy() {
                return changedBy;
        }

        public void setChangedBy(String changedBy) {
                this.changedBy = changedBy;
        }

        public StringTimestamp getInicio() {
                return inicio;
        }

        public void setInicio(StringTimestamp inicio) {
                this.inicio = inicio;
        }

        public StringTimestamp getFim() {
                return fim;
        }

        public void setFim(StringTimestamp fim) {
                this.fim = fim;
        }
```

```
    public String getAbrevTipoAula() {
            return abrevTipoAula;
    }

    public void setAbrevTipoAula(String abrevTipoAula) {
            this.abrevTipoAula = abrevTipoAula;
    }

    public String getNomeTipoAula() {
            return nomeTipoAula;
    }

    public void setNomeTipoAula(String nomeTipoAula) {
            this.nomeTipoAula = nomeTipoAula;
    }

    public List<String> getBibliografia() {
            return bibliografia;
    }

    public void setBibliografia(List<String> bibliografia) {
            this.bibliografia = bibliografia;
    }

    public Integer getIdValidacao() {
            return idValidacao;
    }

    public void setIdValidacao(Integer idValidacao) {
            this.idValidacao = idValidacao;
    }

    public StringDate getDataValidacao() {
            return dataValidacao;
    }

    public void setDataValidacao(StringDate dataValidacao) {
            this.dataValidacao = dataValidacao;
    }

    public String getLoginValidacao() {
            return loginValidacao;
    }

    public void setLoginValidacao(String loginValidacao) {
            this.loginValidacao = loginValidacao;
```

```
        }

        public Integer getNumero() {
                return numero;
        }

        public void setNumero(Integer numero) {
                this.numero = numero;
        }

        public Integer getAlunoCount() {
                return alunoCount;
        }

        public void setAlunoCount(Integer alunoCount) {
                this.alunoCount = alunoCount;
        }

}
```

B.2.6 ReScheduleStatus

```
package pt.ipb.sumarios.entity.client;

public enum ReScheduleStatus {

        Modified(2, "Aceite/Modificado"),
        Accepted(1, "Aceite"),
        Pending(0, "Pendente"),
        Rejected(-1, "Rejeitado"),
        Canceled(-2, "Cancelado");

        private Integer value;

        private String descricao;

        private ReScheduleStatus(Integer value, String descricao) {
                this.value = value;
                this.descricao = descricao;
        }

        public Integer value() {
                return value;
        }

        public String descricao() {
                return descricao;
        }
```

```
        public static ReScheduleStatus statusFor(Integer value) {
                switch (value) {
                case 0: return Pending;
                case 1: return Accepted;
                case -1: return Rejected;
                case 2: return Modified;
                default: return null;
                }
        }

}
```

B.2.7 Aula

```
package pt.ipb.sumarios.entity.client;

import java.io.Serializable;
import java.util.List;

import pt.ipb.sumarios.entity.client.annotations.Id;
import pt.ipb.sumarios.entity.client.annotations.ReadOnly;
import pt.lobo.gwt.user.client.date.StringDate;
import pt.lobo.gwt.user.client.date.StringTimestamp;

@SuppressWarnings("serial")
public class Aula implements Serializable {

        @Id
        private Integer id;

        private StringDate data;

        private StringTimestamp inicio;

        private StringTimestamp fim;

        private Integer idPeriodo;

        private SalaId salaId;

        // ReadOnly

        @ReadOnly
        private Integer codEscolaSala;

        @ReadOnly
        private String escolaSala;
```

```
@ReadOnly
private String abrevEscolaSala;

@ReadOnly
private String nomeSala;

@ReadOnly
private String abrevSala;

@ReadOnly
private List<DetalhesAula> detalhesAula;

@ReadOnly
private Integer sumarioCount;

@ReadOnly
private List<Docente> docentes;

@ReadOnly
private boolean rescheduled;

@ReadOnly
private boolean editManual;

@ReadOnly
private boolean justificacao;

public Aula() {
}

public Integer getId() {
        return id;
}

public void setId(Integer id) {
        this.id = id;
}

public StringDate getData() {
        return data;
}

public void setData(StringDate data) {
        this.data = data;
}
```

```
        public StringTimestamp getInicio() {
                return inicio;
        }

        public void setInicio(StringTimestamp inicio) {
                this.inicio = inicio;
        }

        public StringTimestamp getFim() {
                return fim;
        }

        public void setFim(StringTimestamp fim) {
                this.fim = fim;
        }

        public Integer getIdPeriodo() {
                return idPeriodo;
        }

        public void setIdPeriodo(Integer idPeriodo) {
                this.idPeriodo = idPeriodo;
        }

        public Integer getCodEscolaSala() {
                return codEscolaSala;
        }

        public void setCodEscolaSala(Integer codEscolaSala) {
                this.codEscolaSala = codEscolaSala;
        }

        public String getNomeSala() {
                return nomeSala;
        }

        public void setNomeSala(String nomeSala) {
                this.nomeSala = nomeSala;
        }

        public String getAbrevSala() {
                return abrevSala;
        }

        public void setAbrevSala(String abrevSala) {
                this.abrevSala = abrevSala;
        }
```

```
    public String getEscolaSala() {
            return escolaSala;
    }

    public void setEscolaSala(String escola) {
            this.escolaSala = escola;
    }

    public List<DetalhesAula> getDetalhesAula() {
            return detalhesAula;
    }

    public void setDetalhesAula(List<DetalhesAula> detalhesAula) {
            this.detalhesAula = detalhesAula;
    }

    public Integer getSumarioCount() {
            return sumarioCount;
    }

    public void setSumarioCount(Integer sumarioCount) {
            this.sumarioCount = sumarioCount;
    }

    public void setDocentes(List<Docente> docentes) {
            this.docentes = docentes;
    }

    public List<Docente> getDocentes() {
            return docentes;
    }

    public boolean isRescheduled() {
            return rescheduled;
    }

    public void setRescheduled(boolean reschedule) {
            this.rescheduled = reschedule;
    }

    public SalaId getSalaId() {
            return salaId;
    }

    public void setSalaId(SalaId idSala) {
            this.salaId = idSala;
```

```
        }

        public boolean isEditManual() {
                return editManual;
        }

        public void setEditManual(boolean editManual) {
                this.editManual = editManual;
        }

        public String getAbrevEscolaSala() {
                return abrevEscolaSala;
        }

        public void setAbrevEscolaSala(String abrevEscolaSala) {
                this.abrevEscolaSala = abrevEscolaSala;
        }

        public boolean isJustificacao() {
                return justificacao;
        }

        public void setJustificacao(boolean justificacao) {
                this.justificacao = justificacao;
        }

}
```