UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

PROTÓTIPO DE INGRESSO DIGITAL PARA DISPOSITIVOS MÓVEIS

FILIPE LUCHINI FORTINHO

BLUMENAU
2011

2011/2-10

FILIPE LUCHINI FORTINHO

PROTÓTIPO DE INGRESSO DIGITAL PARA DISPOSITIVOS MÓVEIS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Francisco Adell Péricas, M. - Orientador

BLUMENAU
2011

2011/2-10

PROTÓTIPO DE INGRESSO DIGITAL PARA DISPOSITIVOS MÓVEIS

Por

FILIPE LUCHINI FORTINHO

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Francisco Adell Péricas, M. – Orientador, FURB
Membro:	Prof. Paulo Fernando da Silva, M. – FURB
Membro:	Prof. Sérgio Stringari, M. – FURB

Blumenau, 13 de Dezembro de 2011

Dedico este trabalho a todos os amigos, especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

A Deus, por ter me dado sabedoria e guiado meus passos até onde estou.

À minha família, que aposta em mim e em meus ideais.

Aos meus amigos, que têm fé em mim e acreditam em minha capacidade.

Ao meu orientador, Francisco Adell Péricas, por ter acreditado na conclusão deste trabalho e sempre me ajudar da melhor forma possível.

Siga sempre em frente, cada vez mais forte, avante e avante sem parar...

Zell Ruskea

RESUMO

Este trabalho apresenta uma forma alternativa para o uso de ingressos, onde esses ingressos são utilizados de forma digital dentro de um dispositivo móvel *Android*. Dessa forma os ingressos são disponibilizados e manipulados de forma digital com um aparelho móvel. A obtenção ou compra do ingresso é efetuada através de serviços web solicitados pelo aparelho móvel e, por fim, sua leitura e validação é feita através de QRCodes gerados a partir das informações de cada ingresso digital. Essa sistemática tem por objetivo viabilizar e facilitar a compra de ingressos como também dispensar o uso de papel, que geralmente é mais caro e suscetível a perdas, danos ou falsificações.

Palavras-chave: Ingresso. Dispositivo móvel. Android. QRCode. WebServices. Java.

ABSTRACT

This paper presents an alternative way to use the ticket, where these tickets are used in digital form within a mobile device Android. So tickets are available digitally and manipulated with a mobile device. The acquisition or purchase of admission is done through web services requested by the mobile device, and finally, reading and validation is done through QRCode generated from the digital information in each entry. This system aims to enable and facilitate the purchase of tickets as well dispense with the paper, which is generally more expensive and susceptible to loss, damage or tampering.

Key-words: Ingress. Mobile device. Android. QRCode. WebServices. Java..

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

API – *Application Programming Interface*

BCC – Bacharelado em Ciência da Computação

HTML – *HyperText Markup Language*

HTTP – *HyperText Transfer Protocol*

IDE – *Integrated Development Environment*

JMF – *Java Media Framework*

JSP – *JavaServer Pages*

Wi-Fi – *Wireless Fidelity*

SDK – *Software Development Kit*

USB – *Universal Serial Bus*

UML – *Unified Modeling Language*

URL – *Uniform Resource Locator*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Com a constante evolução das plataformas móveis hoje pode-se informatizar e digitalizar várias tarefas que antes eram totalmente inviáveis. Isto porque há alguns anos atrás as plataformas móveis eram muito limitadas, e alguns tipos de tarefas demandam recursos computacionais que anteriormente havia somente em computadores pessoais.

Uma dessas tarefas é a compra e uso de ingressos para teatros, shows, casas noturnas e demais eventos. Com as plataformas móveis atuais pode-se migrar o uso do ingresso de papel para uma plataforma móvel, constituindo assim o ingresso digital. Dessa forma pode-se facilitar e aperfeiçoar tanto o processo de aquisição de ingressos quanto a sua validação nas entradas e portarias dos eventos.

A sistemática do processo consiste basicamente em três partes:

a) compra do ingresso através de um WebService;
b) ingresso digitalizado, salvo e criptografado na forma de QRCode no dispositivo móvel do usuário;
c) leitura e validação do ingresso (QRCode) através de uma câmera.

Ou seja, é preciso que o usuário efetue a compra do ingresso digital com o seu próprio dispositivo móvel via WebService. Após a compra, este ingresso é gravado e criptografado em seu dispositivo de forma digital. Para a validação e leitura do ingresso nas portarias dos eventos basta o usuário abrir a aplicação de ingressos digitais e exibir seu ingresso digital para a câmera do sistema leitor e validador de ingressos digitais. A aplicação móvel de ingressos digitais sempre exibirá o ingresso no formado de um QRCode, possibilitando assim que a aplicação leitora faça a validação do ingresso digital em poucos segundos.

A Figura 1 apresenta uma ilustração da sistemática deste processo.

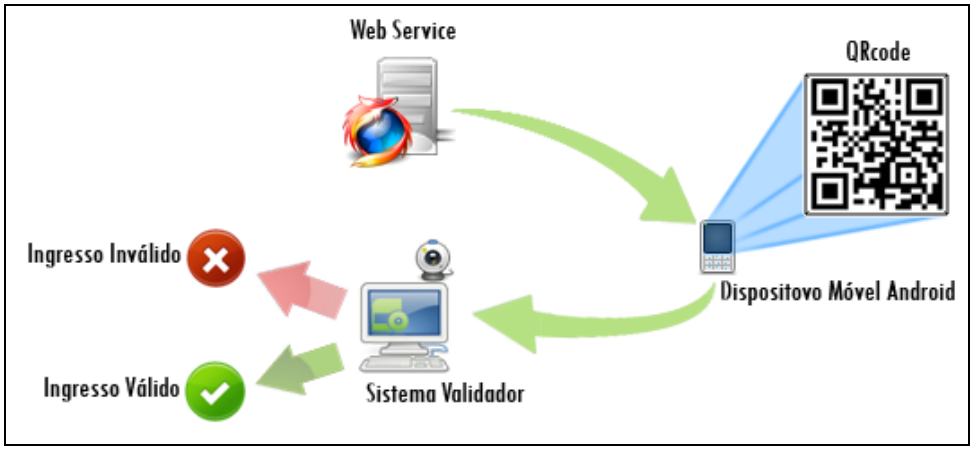


Figura 1 – Ciclo conceitual do processo de uso do ingresso digital

O WebService contém um serviço de ingressos digitais, disponibilizando dessa forma os seus respectivos ingressos. Estes ingressos podem ser acessados e adquiridos através da aplicação cliente *Android* do usuário. A aplicação cliente, além de armazenar o ingresso do usuário, também é responsável por criptografar as informações do ingresso digital no formato de um QRCode.

Por fim a aplicação validadora faz a leitura e a validação dos ingressos. Essa leitura se dá através de uma câmera de vídeo que esteja conectada ao computador com a aplicação validadora. Assim, através da câmera de vídeo, é possível a extração de informação para a execução da validação dos ingressos digitais.

1.1 OBJETIVOS DO TRABALHO

O objetivo principal deste trabalho é tornar viável a execução do ciclo de uso de um ingresso digital, conforme foi exemplificado na Figura 1. Para que a execução deste ciclo seja possível é necessária a desenvolvimento de três aplicações: um WebService, uma aplicação cliente *Android* para o dispositivo móvel do usuário e uma aplicação validadora de ingressos digitais.

Os objetivos específicos do trabalho são:

a) criar um WebService em Java que disponibiliza a aquisição e gerenciamento de ingressos digitais;
b) criar uma aplicação móvel Android que acessa o WebService para consultar e adquirir os ingressos digitais;
c) criar uma aplicação validadora em Java que será utilizada para validação dos ingressos digitais através dos QRCodes.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está estruturado em quatro capítulos.

No capítulo um temos a introdução, conforme já foi visto. O capítulo dois contém a fundamentação teórica necessária para permitir um melhor entendimento sobre este trabalho.

O capítulo três apresenta o desenvolvimento das três aplicações necessárias para uso do ingresso digital, contemplando os principais requisitos do problema e a especificação contendo os casos de uso, diagramas de classes e seqüência. Nesse capítulo são apresentadas também as ferramentas e tecnologias utilizadas na implementação. Por fim são apresentados os resultados e discussões.

O quarto capítulo refere-se às conclusões do trabalho e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nas seções a seguir estão relacionados os principais pontos que envolvem o desenvolvimento deste trabalho. Na seção 2.1 é demonstrado o potencial dos dispositivos móveis para acessar a Internet. A seção 2.2 apresenta o potencial da plataforma Android e como suas ferramentas, recursos e características se comportam. Já a seção 2.3 exibe alguns aplicativos que usam formas de autenticação de usuário, autenticação de dados e os desafios que se tem por trás desses tipos de validadores. Na seção 2.4 vê-se como funciona um QRCode e como ele é usado para a captura de informação através de desenhos gráficos exibidos num plano. Na seção 2.5 é demonstrado que sistemas de pagamentos para dispositivos móveis já não são nenhum mistério, e que seu uso não é algo tão complexo. E por fim, na seção 2.6, são exibidos alguns trabalhos correlatos similares à aplicação proposta por este trabalho.

2.1 ACESSO A INTERNET ATRAVÉS DE DISPOSITIVOS MÓVEIS

O número de acesso a Internet através de dispositivos móveis tem crescido a cada dia. “Uma pesquisa realizada pelo instituto Gartner faz previsões sobre a utilização de internet móvel. Segundo o estudo publicado no começo de 2010, 2013 deve ser o ano em que será mais comum acessar a internet pelo celular do que pelo computador.” (GILSOGAMO, 2010a).

Dessa forma pode-se prever que cada vez mais as empresas estejam inclinadas a conquistar novos mercados nas áreas relacionadas a dispositivos móveis. E essa previsão já é realidade em muitas grandes empresas do Brasil, como a Coelho da Fonseca, que é uma das maiores empresas de mercado imobiliário do Brasil, e a Empresa Pão de Açúcar (NASCIMENTO, 2010).

> Acompanhamos o comportamento do consumidor e as pesquisas que indicam que em poucos anos teremos mais acesso a internet pelo celular do que pelo computador. A gente vê o mobile marketing como um futuro muito próximo. Há dois anos estamos investindo no segmento com ações SMS e neste ano temos investimentos maiores com lançamento de aplicativos. (DIETRICH, 2010).

2.1.1 AUTENTICAçÃO EM DISPOSITIVOS mÓVEIS

Formas de autenticação de usuário ou de validação da integridade de dados são ainda pouco comuns no mundo dos dispositivos móveis. Mas mesmo assim existem algumas empresas que oferecem serviços onde há a necessidade desses tipos de validações. Algumas empresas têm investido em formas de autenticar usuários e validar a integridade de dados. Um exemplo de autenticação de usuário através de dispositivos móveis é o sistema de pagamento de taxi lançado pela Sodexo.

> A Sodexo lançou no mercado um sistema de m-payment para pagar as corridas de táxi... Ela funciona mais ou menos assim: o usuário avisa o taxista que pagará a corrida via celular. Ao chegar ao destino, o motorista comunica o valor para a central de táxi. O usuário, então, recebe uma ligação no celular para que ele confirme o valor e autorize o pagamento. (MIWA, 2009).

Já um bom exemplo de autenticação de integridade de dados é o sistema de compra de ingressos para o *Rock in Rio* que foi usado em Portugal. Esse é um sistema em que o usuário faz a compra do ingresso com seu celular e como retorno recebe um tipo de ingresso digital, chamado de *pincode*.

> Para adquirir ingresso os consumidores devem acessar o site da Vodafone live! (http://www.vodafone.pt/main/particulares) e informar o número de seu celular na área determinada. Após isso, o usuário recebe um SMS com endereço que deve acessar para realizar a comprar.
>
> Para realizar o pedido o usuário deve informar a data e quantidade de ingressos que deseja. Após confirmar o pedido de compra, o usuário é direcionado ao serviço de mobile payment MB Phone da Vodafone para realizar o pagamento dos bilhetes.
>
> Ao efetuar o pagamento o consumidor recebe um SMS por cada bilhete adquirido, nele está presente uma espécie de pincode (bCode) que permitirá o acesso ao evento.
>
> Durante todo o Rock in Rio existirá nos portões de acesso máquinas capazes de decodificar este código. (GILSOGAMO, 2010c).

2.1.2 SISTEMA DE PAGAMENTO ATRAVÉS DE DISPOSITIVOS MÓVEIS

Para facilitar o pagamento de serviços, tornando dessa forma o uso do serviço prático, existe no mercado um sistema conhecido como *mobile-payment* ou *m-payment* (pagamento móvel). Há várias formas de se implementar um sistema de pagamento em uma aplicação para dispositivos móveis. As empresas de telefonia móvel, bancos, operadoras de cartão e companhias do setor de tecnologia vêm investindo a alguns anos em *m-payment*. Diversos projetos comerciais e bem sucedidos são utilizados no mercado nacional (GILSOGAMO,

2010b).

“Este segmento tem um potencial monstruoso. Não vai substituir o cartão tão cedo, mas para alguns mercados se encaixa muito bem pela capilaridade e agilidade que traz.” (GILSOGAMO, 2010b **apud** MOTA, 2010).

2.2 PLATAFORMA ANDROID

O Android é uma plataforma para dispositivos móveis de código aberto pertencente a Open Handset Alliance. A Open Handset Alliance é um grupo de empresas e colaboradores aliadas para o desenvolvimento de uma plataforma móvel aberta. Neste grupo, que é liderado pela Google, existem fabricantes de aparelhos portáteis, fabricantes de componentes, provedores de plataformas e soluções de software e empresas de marketing (OPEN HANDSET ALLIANCE, 2010).

Parando um pouco para analisar seus concorrentes, o BlackBerry e o iPhone são duas plataformas de grande qualidade, mas que trabalham de lados diferentes de um mesmo prisma. O BlackBerry é um dispositivo móvel extremamente seguro no que diz respeito a aplicações e tarefas corporativas, porém no que diz respeito a usabilidade para usuários mais leigos e entretenimento tem pouca competitividade com o iPhone. Já o iPhone é de fácil uso para usuários não familiarizados com software além de possuir uma grande extensão e desenvolvimento em aplicativos de entretenimento (ABLESON, 2009).

O Android é uma plataforma nova que demonstra um grande potencial, podendo jogar nos dois lados desse mesmo prisma em que o BlackBerry e o iPhone se encontra. Assim é possível que a plataforma Android crie uma ponte entre esses dois lados, possibilitando aplicações seguras e de qualidade bem como também a facilidade de uso e o acesso a aplicativos de entretenimento (ABLESON, 2009).

Atualmente a plataforma Android é muito rica em recursos e poder computacional, “seria fácil confundi-lo com um sistema operacional desktop.” (ABLESON, 2009). Isso possibilita que seja possível a construção de uma grande gama de aplicações. A justificativa para tantos recursos aprimorados é que o Android é baseado em um sistema de camadas

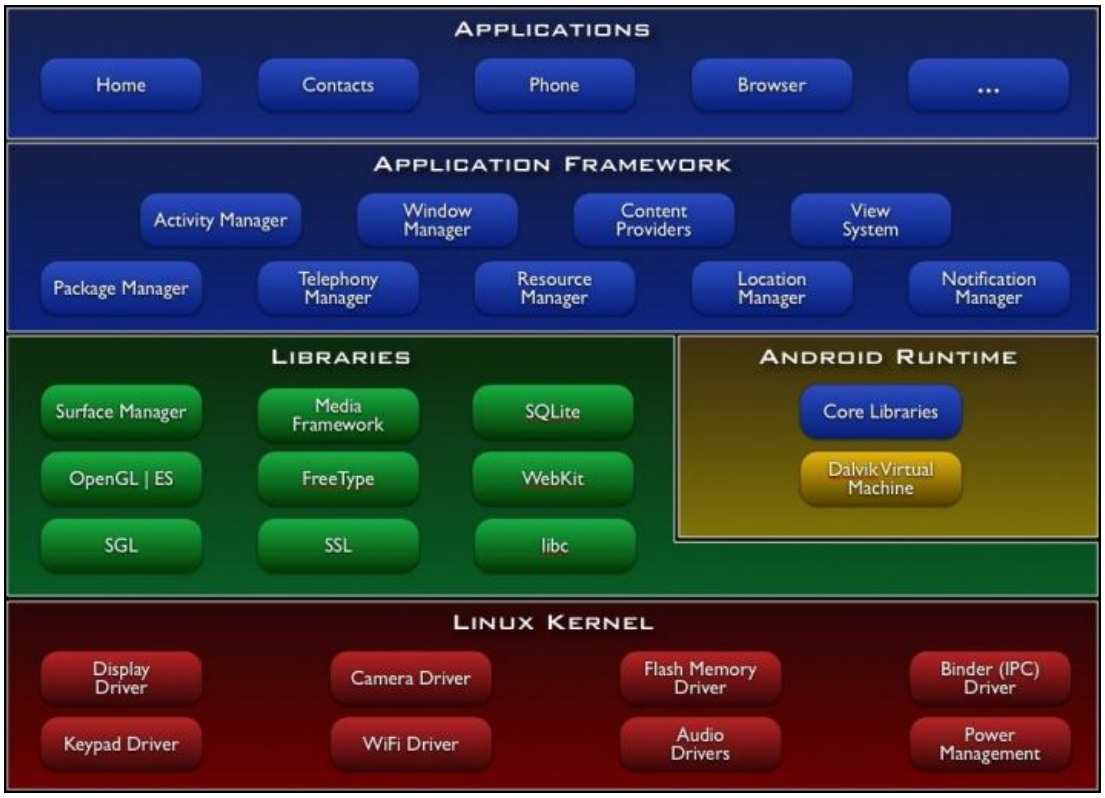
comandada por um *kernel*[1] Linux. Além do mais, os seus subsistema *User Interface* (UI) possuem: janelas, visualizações e Widgets, elementos de interface comuns à aplicação como caixas de edição e listas.

Além dos recursos UI mais comuns, o Android possui uma enorme gama de outros recursos distintos, que variam desde recursos de comunicação de dados e rede até banco de dados SQLine de *design*[2] 2D e 3D com o OpenGL (ANDROID DEVELOPERS, 2010). Abaixo há uma lista desses recursos e suas características:

a) estrutura de aplicativo que permite a reutilização e substituição de componentes;
b) dalvik: máquina virtual otimizada para dispositivos móveis;
c) navegador web integrado com base no código aberto WebKit;
d) gráficos otimizados alimentados por uma biblioteca de gráficos 2D personalizado e gráficos 3D baseado no OpenGL ES 1.0 especificação (aceleração de hardware opcional);
e) SQLite para armazenamento de dados estruturados;
f) suporte de mídias para áudio comum, vídeo e formatos de imagem estática (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF);
g) GSM de telefonia (hardware dependente);
h) Bluetooth, EDGE, 3G e WiFi (dependente de hardware);
i) câmera, GPS, bússola e acelerômetro (hardware dependente);
j) rico ambiente de desenvolvimento, incluindo um dispositivo emulador, ferramentas de depuração, memória e perfil de desempenho, e um *plugin* para o Eclipse IDE.

Como foi mencionado anteriormente, o Android é baseado em um sistema de camadas, possibilitando assim o fornecimento de tais quantidades de recursos. Na Figura 2, pode-se ver de uma forma geral como o sistema de camadas do Android é organizado e respectivamente onde se localiza as camadas e bibliotecas de cada um dos seus recursos. Em seguida é dada uma breve explicação de cada camada.

[1] *kernel*, Em computação, o núcleo ou cerne (em inglês: kernel) é o componente central do sistema operativo da maioria dos computadores; ele serve de ponte entre aplicativos e o processamento real de dados feito a nível de hardware.
[2] *design*, desenho industrial (português brasileiro) ou desenho ou ainda modelo (português europeu) é a configuração, concepção, elaboração e especificação de um artefato. Essa é uma atividade técnica e criativa, normalmente orientada por uma intenção ou objetivo, ou para a solução de um problema. Simplificando, pode-se dizer que design é projeto.



Fonte: ANDROID DEVELOPERS (2010).

Figura 2 – Sistema de camadas Android

2.2.1 Aplicação

Na camada de aplicação é onde ficam as aplicações desenvolvidas. As aplicações são gravadas na linguagem Java e é executado em uma máquina virtual, conhecida também como *Virtual Machine* (VM). A VM não é uma *Java Virtual Machine* (JVM), mas é uma *Dalvik Virtual Machine* (DVM), uma tecnologia de software livre. Cada aplicativo Android é executado em uma instância da DVM, que, por sua vez, reside em um processo gerenciado por *kernel* Linux, conforme apresentado na Figura 3.

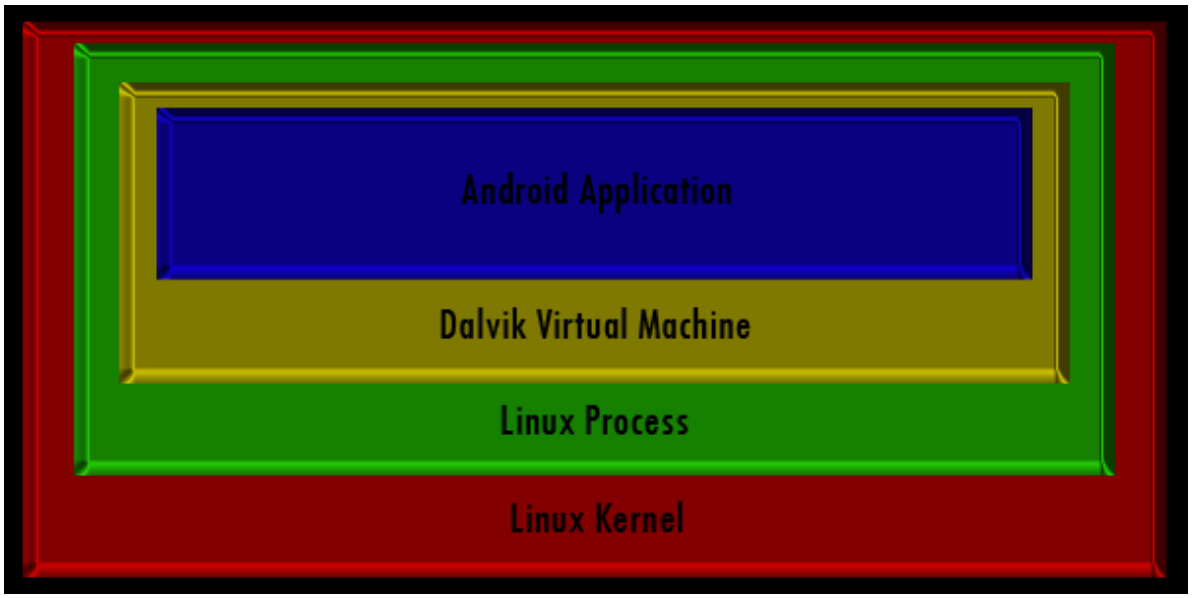


Figura 3 – *Dalvik Virtual Machine*

2.2.2 *Framework*

A camada *framework* fornece uma plataforma de desenvolvimento aberta. O Android oferece aos desenvolvedores a capacidade de criar aplicações inovadoras. Os desenvolvedores podem utilizar os recursos de hardware do dispositivo para localização, acesso, executar serviços de fundo (em *background*), definir alarmes, notificações para adicionar a barra de status, capturar imagens através da câmera e muitas outras variadas funcionalidades.

Os desenvolvedores têm livre acesso às APIs usadas pelos aplicativos nativos e fundamentais do Android. A arquitetura dos aplicativos é projetada para simplificar a reutilização de componentes; qualquer aplicação manter “livre” as suas funcionalidades e qualquer outra aplicação pode então fazer uso dessas funcionalidades.

2.2.3 Bibliotecas

Na camada de bibliotecas o Android inclui um conjunto de bibliotecas em C/C++ usadas por diversos componentes do sistema operacional Android. Estas funcionalidades são expostas aos desenvolvedores através da estrutura de aplicação Android conforme citado na descrição da camada *framework*.

2.2.4 Android *Runtime*

O Android *Runtime* inclui um conjunto de bibliotecas que fornece a maioria das funcionalidades disponíveis nas principais bibliotecas da linguagem de programação Java. Como foi explicado, cada aplicação Android roda em seu próprio processo, com sua própria instância da máquina virtual DVM. A VM é baseada em registradores, e executa classes compiladas por um compilador Java, que por sua vez são transformados para o formato dex por uma ferrameta chamada "dx" incluida no *Software Development Kit* (SDK) do Android. Em seguida A DVM invoca o kernel do Linux para as funcionalidades subjacentes, como gerenciamento de memória *threading* e de baixo nível (ANDROID DEVELOPERS, 2010).

2.2.5 *Kernel* Linux

Por fim na última camada se encontra o *kernel* Linux para os serviços centrais do sistema, como segurança, gerenciamento de memória, gestão de processos, pilha de rede e *drivers*. O *kernel* também atua como uma camada de abstração entre o hardware e o resto da pilha de software.

2.2.6 Fundamentos gerais das aplicações

As aplicações Android são escritos em Java. O SDK Android compila o código, juntamente com todos os dados e arquivos que definem a aplicação. Todo o conteúdo compilado e os arquivos da aplicação desenvolvida são empacotadas em um único pacote, que é transformado em uma arquivo que possui o sufixo ".apk". Todo o código contido em um arquivo "apk" é considerado um aplicativo, e é esse arquivo que a ferramenta *Android Powered Devices* utiliza para instalar a aplicação no dispositivo móvel do usuário.

Uma vez instalado em um dispositivo, cada aplicativo Android atua em sua própria *sandbox*[3] de segurança. O sistema operacional Android é multi-usuário, sendo assim: cada aplicação é considerada um usuário diferente; por padrão, o sistema atribui a cada aplicação

um *Identity Document* (ID) de usuário exclusivo do Linux; o sistema define as permissões para todos os arquivos em um aplicativo, de modo que somente o ID do usuário atribuído a esse aplicativo pode acessá-los; cada processo tem sua própria VM, para que o código de um aplicativo seja executado isoladamente das demais aplicações.

Cada aplicação é executada em seu próprio processo Linux. O Android começa o processo quando qualquer um dos componentes do aplicativo precisa ser executado. Em seguida, destrói o processo quando ele não é mais necessário ou quando o sistema deve recuperar a memória para outras aplicações (ANDROID DEVELOPERS, 2010).

Desta forma, o sistema Android define um privilégio mínimo para cada aplicação. Ou seja, cada aplicação, por padrão, só tem acesso aos componentes que ele necessita para fazer o seu trabalho e nada mais. Isso cria um ambiente muito seguro em que um aplicativo não pode acessar partes do sistema para o qual não tem permissão.

Apesar desta segurança e do conceito de *sandbox*, existem maneiras de uma aplicação compartilhar dados com outras aplicações ou até mesmo acessar serviços e recursos do sistema. Por exemplo, é possível definir para dois aplicativos diferentes o mesmo ID de usuário, dessa forma ambos os aplicativos podem acessar os arquivos um do outro. Também, para economizar recursos do sistema, é possível que essas duas aplicações executem no mesmo processo, compartilhando dessa forma a mesma VM. Lembrando que para que isso seja possível, as duas aplicações devem estar assinadas com o mesmo certificado.

Um aplicativo pode solicitar permissão para acessar dados ou recursos do dispositivo, como contatos do usuário, mensagens SMS, a memória do cartão SD (*Secure Digital Card*), câmera, *bluetooth* entre outros recursos e serviços. Todas as permissões de acesso a esses recursos devem ser concedidas pelo usuário no momento da instalação.

2.2.6.1 Componentes principais de uma aplicação

Os componentes de aplicação são os componentes essenciais de uma aplicação Android. Cada componente representa um ponto diferente, que pode ser de entrada ou saída, através do qual o sistema pode acessar uma aplicação e vice-versa. Nem todos os componentes são pontos de entrada real para a aplicação e alguns destes componentes

[3] *Sandbox*, em segurança de computadores ,um *sandbox* é um mecanismo de segurança para separar os programas em execução do sistema operacional principal. Serve fundamentalmente para evitar que alguma aplicação tenha acessos perigosos a recursos do sistema operacional e o acabe danificando.

dependem uns dos outros, mas cada um existe como uma entidade própria e desempenha um papel específico. Cada um é um bloco de construção singular que ajuda a definir o comportamento global da aplicação.

Existem quatro tipos diferentes de componentes de aplicação. Cada tipo serve a um propósito distinto e tem seus ciclos de vida distintos que definem como o componente é criado e destruído. Esses quatro componentes são: *Activities*, *Services*, *Content providers* e *Broadcast receivers* (ANDROID DEVELOPERS, 2010).

2.2.6.1.1 Activities

Uma atividade (*activity*) representa uma única tela com uma interface de usuário. Por exemplo, na aplicação de ingressos digitais deste trabalho há uma atividade que mostra notícias de eventos, outra atividade que exibe uma lista de eventos e uma terceira atividade que exibe o ingresso com o seu QRCode. Embora as atividades trabalhem em conjunto para formar uma única aplicação, cada atividade é independente das demais atividades. Assim a aplicação pode iniciar em qualquer uma destas atividades.

Uma atividade sempre é implementada como uma subclasse da classe *Activity*. A classe *Activity* é uma das classes disponibilizadas na biblioteca do SDK Android.

2.2.6.1.2 Services

Um serviço (*service*), similar a um serviço do sistema operacional Windows, é um componente que roda em segundo plano (*background*) para executar operações de longa duração ou para realizar trabalhos para os processos de controle remoto. Um serviço não fornece uma interface de usuário. Por exemplo, um serviço pode reproduzir música em segundo plano enquanto o usuário estiver em um aplicativo diferente, ou pode buscar dados sobre a rede sem interação do usuário com o bloqueio de uma atividade. Outro componente, como uma atividade (*activity*), pode iniciar o serviço e deixá-lo executando ou requisitá-lo, a fim de interagir com o serviço.

Um serviço é implementado como uma subclasse de *Service*. A classe *Service* também é uma das classes disponibilizadas na biblioteca do SDK Android.

2.2.6.1.3 Content providers

Um provedor de conteúdo (*content providers*) gerencia um conjunto compartilhado de dados. Uma aplicação pode armazenar os dados no sistema de arquivos do Android, em um banco de dados SQLite, ou qualquer outro local de armazenamento que o aplicativo possa acessar. Através do provedor de conteúdo, outras aplicações podem consultar ou mesmo alterar os dados, isso se o provedor de conteúdo permitir que isso seja feito. Por exemplo, o sistema Android fornece um provedor de conteúdo que gerencia informações de contatos do usuário. Dessa forma qualquer aplicação com as permissões adequadas pode consultar parte do provedor de conteúdo para ler e gravar informações sobre uma pessoa em particular.

Os provedores de conteúdo também são úteis para leitura e escrita de dados que é privado para uma aplicação e não compartilhados. Por exemplo, a aplicação de ingressos digitais desenvolvida por este trabalho guarda seus ingressos e seus QRCodes em um provedor de conteúdo privado a qual somente a aplicação de ingressos digitais pode acessar.

Um provedor de conteúdo é implementado como uma subclasse de *ContentProvider* e deve implementar um conjunto padrão de APIs que permite que outras aplicações realizem transações.

2.2.6.1.4 Broadcast receivers

Um receptor de difusão (*broadcast receivers*) é um componente que responde as “chamadas” ou “avisos” de todo o sistema de transmissão. Muitas transmissões podem ter sua origem no sistema operacional, como uma transmissão avisando que a tela foi desligada, a bateria está fraca, ou uma imagem foi capturada. Os aplicativos também podem iniciar as transmissões, por exemplo, para permitir que outras aplicações saibam que alguns dados foram baixados no dispositivo e que estão disponíveis para serem usados. Apesar de receptores de transmissão não apresentarem uma interface de usuário, eles podem criar uma notificação de barra de status para alertar o usuário quando um evento de transmissão ocorreu. Normalmente um receptor de transmissão é apenas uma "*gateway*" para outros componentes e se destina a fazer uma quantidade de trabalho muito reduzida, como apenas iniciar um serviço.

Um receptor de difusão é implementado como uma subclasse de *BroadcastReceiver* e

cada transmissão é entregue como um objeto *Intent*.

2.2.6.2 Aspectos peculiares dos components

Um aspecto único do sistema Android é que uma aplicação pode iniciar em qualquer componente de outras aplicações. Para isso basta que as demais aplicações compartilhem seus componentes e arquivos. Por exemplo: se uma aplicação vai utilizar uma foto batida pela câmera do dispositivo, esse aplicativo não precisa necessariamente implementar a sua própria interface de acesso à câmera e captura de foto. Se já existir uma aplicação que captura fotos no dispositivo, é necessário apenas que a aplicação inicie na atividade (*activity*) do aplicativo de câmera que captura fotos. Assim, quando a atividade do aplicativo de câmera encerrar a aplicação, o aplicativo, que invocou a aplicação de câmera, pode usar a imagem capturada como quiser.

Quando o sistema inicia um componente, **s**e inicia o processo para que a aplicação instancie as classes necessárias para o componente. Por exemplo, se uma aplicação começa a atividade no aplicativo de câmera que captura uma foto, a atividade é executada no processo que pertence ao aplicativo de câmera e não no processo do aplicativo que invocou o aplicativo de câmera. Portanto, ao contrário das aplicações da maioria dos outros sistemas operacionais, aplicações Android não têm um único ponto de entrada, como, por exemplo, uma função *main*.

Esse intercâmbio entre aplicações é possível porque o sistema cria cada aplicação em um processo separado, definindo permissões de arquivos e restringindo o acesso a outros aplicativos. Dessa forma o aplicativo não pode ativar diretamente um componente de outro aplicativo, já o sistema Android pode. Então, para ativar um componente em outro aplicativo, a aplicação solicitante deve entregar uma mensagem ao sistema que especifica a sua intenção de iniciar um determinado componente, então, sistema Android ativa o componente para a aplicação.

2.3 QR CODE

Códigos de barras tornaram-se popular graças à sua velocidade de leitura, precisão e

características superiores. Como os códigos de barras se tornaram universalmente conhecidos, o mercado começou a chamar a atenção para códigos capazes de armazenar mais informações, com mais tipos de caracteres, e que pudessem ser impresso em um espaço menor. Como resultado, vários esforços foram feitos para aumentar a quantidade de informação armazenada em um código de barras, tais como o aumento do número de dígitos do código de barras empilhando-as ou *layouts*[4] de códigos em múltiplas barras.

Contudo, estas melhorias também causaram vários problemas, como o aumento da área de código de barras, dificultando as operações de leitura, e aumentando o custo de impressão. Assim o código 2D surgiu para tentar solucionar os problemas causados pelo código de barras 1D convencional.

Na Figura 4 pode-se ver a evolução dos códigos até atingir um resultado de código 2D de boa qualidade, chamado de QRCode.

Fonte: QRCODE.COM (2010).

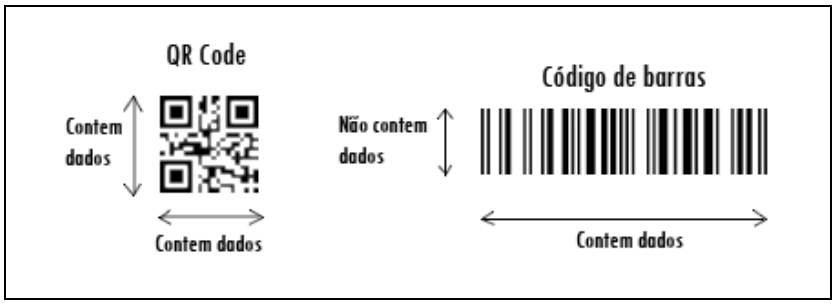
Figura 4 – Evolução dos códigos de barras

O QRCode é um código de barras 2D (bidimensional) criada pela Denso Onda e lançado em 1994 com o objetivo de ser um desenho de fácil interpretação para *scanners*[5] e demais equipamentos de leitura.

O que faz um QRCode ter muito mais vantagens do que um código de barras convencional é que ele contém informações em ambas as direções, tanto na vertical quanto na horizontal, enquanto um código de barras convencional contém informações somente na horizontal. Desse modo um QRCode pode conter uma quantidade de dados consideravelmente maior do que um código de barras comum.

[4] *Layout* gráfico tem como seus componentes a área de design ou formato de página e as margens, que tal como em todo o restante processo de design deve ser bem fundamentado pelo conteúdo do trabalho e pela perspectiva criativa

[5] *Scanners*, do inglês *scanner*, é um periférico de entrada responsável por digitalizar imagens, fotos e textos impressos para o computador, um processo inverso ao da impressora. Ele faz varreduras na imagem física gerando impulsos elétricos através de um captador de reflexos



Fonte: QRCODE.COM (2010).

Figura 5 – Capacidade de armazenamento de cada um dos códigos

2.3.1 Codificação de dados em larga escala

Um código de barras comum geralmente é capaz de armazenar um máximo de aproximadamente 20 dígitos, já o QRCode pode armazenar uma quantidade de informação centenas de vezes maiores. Além de poder armazenar uma quantidade de dados muito maior, um QRCode pode lidar com uma série de tipos diferentes de dados, como, por exemplo, caracteres numéricos e alfabéticos, Kanji, Kana, Hiragana, símbolos binários e códigos de controle, sendo que até 7.089 caracteres podem ser codificados em apenas um QRCode.

QR Code capacidade de dados	
Numéricos	Max. 7089 caracteres
Alfanuméricos	Max. 4296 caracteres
Binários (8 bits)	Max. 2953 bytes
Kanji, de largura total Kana	Max. 1817 caracteres

Um símbolo de código QR deste tamanho pode codificar 300 caracteres alfanuméricos.

Fonte: QRCODE.COM (2010).

Figura 6 – Capacidade de armazenamento de um QRCode

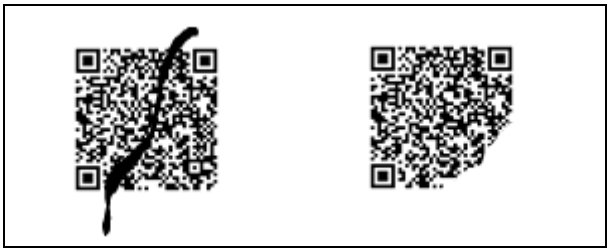
2.3.2 Disposição em pequenos espaços

Além de um QRCode guardar informações tanto na horizontal quanto na verticalmente, ele é capaz de codificar a mesma quantidade de dados de um código de barras comum utilizando uma área dez vezes menor.

Fonte: QRCODE.COM (2010).
Figura 7 – Área de armazenamento do QRCode

2.3.3 Resistência a ruídos

QRCode também é capaz de corrigir erros causados por ruídos em sua imagem. Esses ruídos podem ser causados por sujeira, falha ao desenhar QRCode e demais eventualidades. Os dados perdidos ou ofuscados por ruídos na imagem podem ser restaurados e dessa maneira recuperando até 30% de *codewords*[6].



Fonte: QRCODE.COM (2010).
Figura 8 – Exemplo de áreas danificadas

2.3.4 Varredura em 360 graus

Um QRCode pode ser lido independente da direção em que estiver posicionado (360° graus). Isso é possível devido a suas guias utilizadas para a detecção de posicionamento localizado em três das quatro extremidades do QRCode. Estes padrões de detecção de posição garantem uma leitura de alta velocidade estável, além de evitar a interferência de fundo do ambiente em que o QRCode se encontra.

[6] *Codeword* é uma unidade que define e constrói a área de dados. No caso do QRCode, um *codeword* é igual a 8 bits.

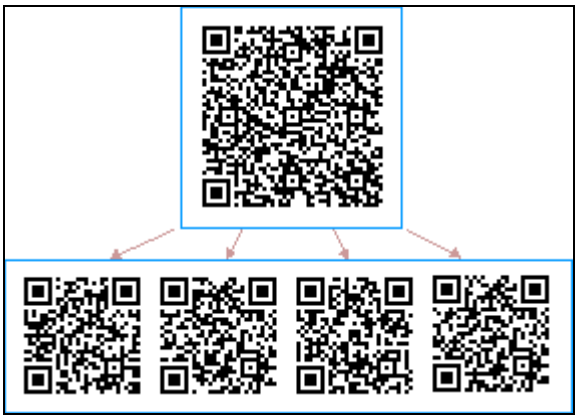
Fonte: QRCODE.COM (2010).

Figura 9 – Guias de detecção de posição

2.3.5 Possibilidade de estruturação dividida

O QRCode oferece o recurso de divisão de informações em QRCode distintos ou a união destas mesmas informações em um único QRCode. Uma mesma quantidade de dados contida em um QRCode pode ser desmembrada em outros QRCodes, como também as informações armazenadas em QRCode diferentes podem ser reconstruídas como um único QRCode de dados.

Um QRCode pode ser dividido em até 16 imagens, permitindo a impressão em uma área estreita.



Fonte: QRCODE.COM (2010).

Figura 10 – Exemplo de desmembramento de um QRCode

2.3.6 Definição de tamanho de um QRCode

O tamanho de um QRCode é definido com base em algumas propriedades:

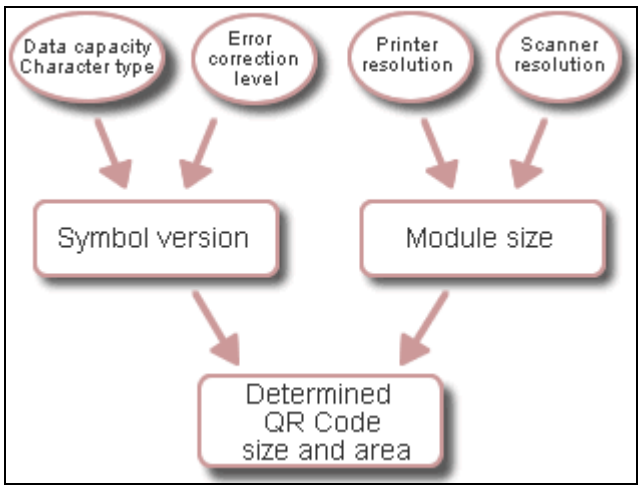
a) versão do símbolo, definido baseado na quantidade de informações que se deseja

codificar no QRCode;

b) tipo de caracter;

c) nível de correção de erros e recuperação de *codewords*;

d) tamanho de módulo, definido com base na resolução do dispositivo que gera\exibe o QRCode (pode ser uma impressora ou a tela de algum dispositivo) e no *scanner* que vai ler o QRCode.



Fonte: QRCODE.COM (2010).

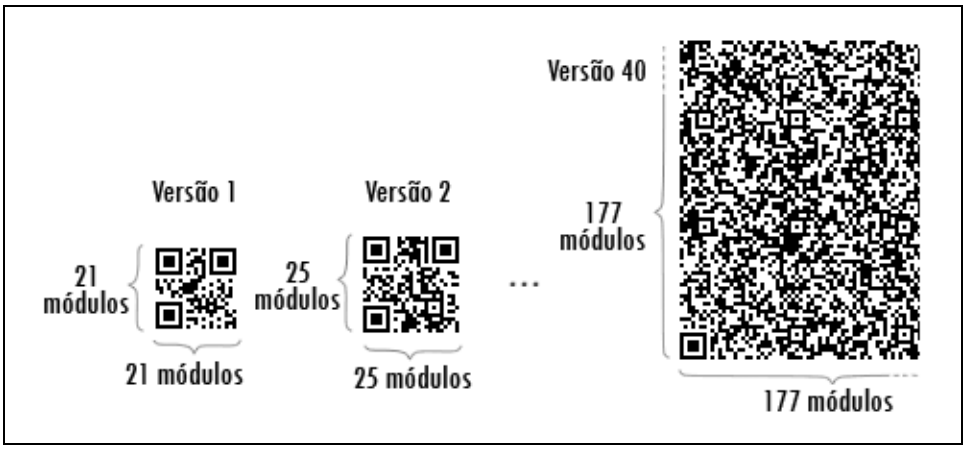
Figura 11 – Definição de tamanho do QRCode

2.3.6.1 Versão de símbolo

As versões de símbolos de uma QRCode podem variar da versão 1 até a versão 40. Cada versão existente tem uma configuração distinta, e são essas configurações que vão definir a quantidade de módulos[7] de cada símbolo.

Por configuração de uma versão entende-se ao número de módulos contidos em um símbolo, iniciando pela versão um (21×21 módulos) até a versão 40 (177×177 módulos). Para cada versão incrementada é adicionado quatro módulos a mais na horizontal e na vertical.

[7] Um módulo refere-se a pontos pretos e brancos que compõem um QRCode.



Fonte: QRCODE.COM (2010).

Figura 12 – Módulos de um QRCode

Cada versão de símbolo de um QRCode tem uma capacidade de armazenamento diferente. Essa capacidade pode ser calculada de acordo com o tipo de caractere codificado e o nível de correção de erros. Ou seja, quanto maior a quantidade de informação que se deseja guardar mais módulos serão utilizados para compor o QRCode, assim resultando em símbolos QRCodes maiores. Na Tabela 1 temos uma relação de algumas configurações existentes dos módulos para cada versão (QRCODE.COM, 2010).

Tabela 1 – Versão e tabela de capacidade máxima de dados

Versão	Módulos	ECC Level	Data Bits	Numérico	Alfanumérico	Binário	Kanji
1	21x21	L	152	41	25	17	10
		M	128	34	20	14	8
		Q	104	27	16	11	7
		H	72	17	10	7	4
2	25x25	L	272	77	47	32	20
		M	224	63	38	26	16
		Q	176	48	29	20	12
		H	128	34	20	14	8
5	37x37	L	864	255	154	106	65
		M	688	202	122	84	52
		Q	496	144	87	60	37
		H	368	106	64	44	27
40	177x177	L	23,648	7,089	4,296	2,953	1,817
		M	18,672	5,596	3,391	2,331	1,435
		Q	13,328	3,993	2,420	1,663	1,024
		H	10,208	3,057	1,852	1,273	784

2.3.6.2 Capacidade de correção de erros

Os QRCodes também possuem recurso de correção de erros e reconstrução de áreas ofuscadas ou que possuam ruídos. Ou seja, mesmo que o QRCode esteja danificado ainda é possível efetuar a sua leitura sem maiores problemas na maioria dos casos. Essa capacidade de recuperação de áreas danificadas é definida em quatro níveis: nível L; nível M; nível Q e nível H respectivamente. O usuário o QRCode deve escolher o nível que se adéqua melhor a sua situação. O nível L é o que possui a menor taxa de recuperação e o nível H a de maior taxa de recuperação. Portanto, se o QRCode é disponibilizado em um ambiente pouco suscetível a danos na imagem, pode-se usar o nível L. Caso a probabilidade de danos a imagem do QRCode seja grande, como por exemplo, um depósito, então se recomenda utilizar o nível H.

Para escolher o nível a ser utilizado também é necessário levar em consideração que quando maior a capacidade de recuperação de dados do QRCode menor a sua capacidade de armazenamento de informação. Sendo assim cada caso deve ser analisado cuidadosamente.

Na maioria dos casos o nível M (com taxa de recuperação de 15%) é o mais freqüentemente usado. A Figura 13 mostra aproximadamente a taxa de recuperação de cada nível.

Capacidade de correção de erros	
Nível L	Aprox. 7%
Nível M	Aprox. 15%
Nível Q	Aprox. 25%
Nível H	Aprox. 30%

Fonte: QRCODE.COM (2010).

Figura 13 – Capacidade de correção de erros

A correção de erros de um QRCode é feita através da adição de um código Reed-Solomon[8] aos dados originais do QRCode. Essa capacidade de correção de erro depende da quantidade de dados a serem corrigidos. Por exemplo, se existem 100 *codewords* para serem codificados, 50 dos quais precisam ser corrigidos, então, 100 *codewords* de código Reed-Solomon são necessárias para a recuperação da informação. Logo, pode-se observar que o

[8] Reed-Solomon é um método de correção de erro matemático utilizado para músicas CDs etc A tecnologia foi originalmente desenvolvida como uma medida contra o ruído de comunicação para satélites artificiais e sondas planetárias. Ele é capaz de fazer uma correção no nível de byte, e é adequado para erros em rajadas concentradas.

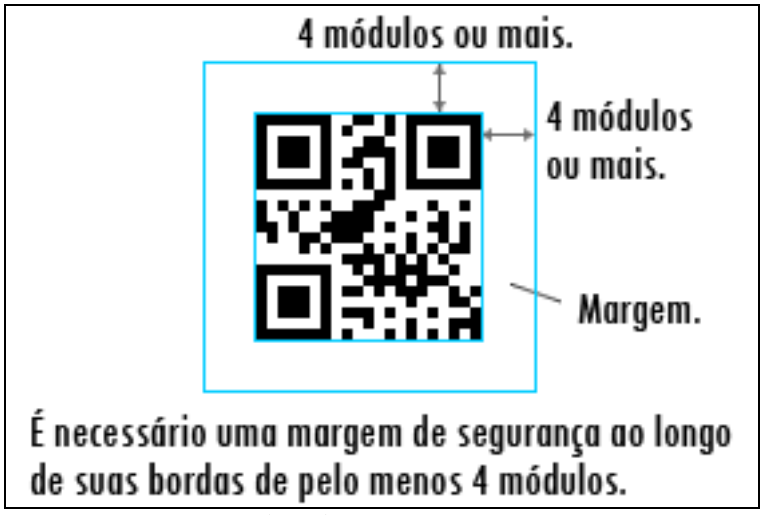
código Reed-Solomon sempre necessita do dobro da quantidade de *codewords* para executar a correção de erros e recuperar os dados perdidos.

Imaginando que em um determinado caso o total de *codewords* do QRCode seja 200, dos quais 50 precisam ser corrigidos, dessa forma, a taxa de correção de erros para o total de *codewords* seria de 25%. Isso corresponderia a um QRCode de nível Q.

2.3.6.3 Margem de segurança

Quando se determina a versão do símbolo e o tamanho do módulo, o tamanho do QRCode é em fim especificado, porém, para uma leitura de precisão é necessário adicionar em torno do QRCode uma margem de segurança, mas conhecida como "zona de silêncio".

A margem de segurança é uma área livre em torno do QRCode, onde nada é desenhado. QRCode exige uma margem de quatro módulos de largura nos quatro lados de seu desenho.



Fonte: QRCODE.COM (2010).

Figura 14 – Margem de segurança de um QRCode

2.3.6.4 Exemplo de cálculo da área do desenho do QRCode

Abaixo está um exemplo do cálculo da área total de um QRCode incluindo sua margem de segurança. Neste exemplo o objetivo é criar um QRCode capaz de codificar 50 caracteres alfanuméricos. Sendo assim devem-se seguir os seguintes passos respectivamente:

a) especificar o nível de correção de erros no nível M;

b) obter uma versão se símbolo de acordo com a Tabela 1. Para isso basta encontrar a intersecção de números de caracteres alfanuméricos que se deseja guardar e o nível M. Dessa forma concluí-se que a verão a ser utilizada é a versão 3, capaz de armazenar 50 ou mais caracteres alfanuméricos no nível M de correção de erros;
c) considerando uma impressora ou tela com resolução de 400 DPI[9] (*Dots Per Inch*) um módulo, constituído de 4 pontos, será desenhado em 0,254 milímetros. Os 0, 254 milímetros são encontrados através da seguinte equação: 25,4 mm/polegadas ÷ 400dpi x 4 pontos/módulo = 0,254mm/módulo;
d) a versão 3 corresponde a 29 módulos, então, o tamanho do QRCode é de 29 módulos × 0, 254 mm/module = 7, 366 milímetros;
e) adicionando ao tamanho do desenho a margem de segurança que possui 4 módulos de largura, o cálculo de tamanho final seria: 7, 366 milímetros + 0, 254mm/module × 8 módulos = 9, 398 milímetros.

Dessa forma pode-se calcular a área total que o desenho do QRCode irá ocupar, assim, conforme o exemplo, calculou-se que a área total igual a 9,398mm² (milímetros aquadrados).

2.4 TRABALHOS CORRELATOS

Embora a forma de uso de serviços proposto por esse trabalho ainda é pouco explorada e conhecida, existem no Brasil e na Europa alguns serviços que têm os mesmos princípios.

Os trabalhos e serviços encontrados são apenas de venda de ingressos, podendo ser citados: Blueticket (BLUETICKET, 2010), ALÔ Ingressos (ALÔ INGRESSO, 2010), mTicket (VODAFONE GROUP, 2010) e Ingressos para o *Rock in Rio* de Portugal (GILSOGAMO, 2010c).

O Blueticket é um sistema de venda de ingressos pela Internet para eventos como teatros, baladas, shows e demais tipos de eventos. Através de um navegador da Internet pode-se entrar no *site* da Blueticket (2010) (www2.blueticket.com.br), selecionar o evento desejado, fazer o pagamento e retirar o seu ingresso no local do evento no dia e na hora em que o evento estiver ocorrendo. O sistema da Blueticket é um sistema feito para PC (*Personal*

[9] DPI, do inglês *dots per inch*, que significa pontos por polegada, é uma medida para medir a densidade de pontos do espaço de impressão ou de vídeo, em particular o número de pontos individuais que podem ser

Computer) que tenha acesso a Internet e um navegador (*Browser*) (BLUETICKET, 2010).

O Alô Ingresso é um sistema de venda de ingressos pela Internet para eventos como teatros, baladas, shows e demais tipos de eventos. Através de um navegador da Internet pode-se entrar no site Alô Ingresso (2010) (www.aloingressos.com.br), selecionar o evento desejado e fazer o pagamento. Após isto se recebe o ingresso no celular (na forma de mensagem) com um código de barras. Ao chegar ao evento, o usuário apresenta a mensagem com o código de barras para o funcionário da portaria. O funcionário efetua a leitura do código de barras utilizando um leitor de código de barras convencional, o qual é aproximado diretamente no visor do celular para executar a leitura. Caso o código de barras seja válido então é liberado a entrada do usuário (ALÔ INGRESSO, 2010).

A empresa de telecomunicações Vodafone também oferece serviços de compra de ingressos para uma série de eventos. O interessante é que todas as transações da compra são feitas através do celular. A diferença é que o meio de comunicação utilizado pelo processo é o próprio serviço de telecomunicações da Vodafone (2010) e não a Internet. Na Europa esse serviço já é bastante comum, conhecido como *m-ticket* (*mobile ticket* ou ingresso móvel). O processo funciona da seguinte forma: o usuário envia uma mensagem para a Vodafone informando qual o evento de seu desejo e quantos ingressos quer comprar. Em seguida a Vodafone envia uma mensagem confirmando o nome do evento e o valor total da compra de seus ingressos. O usuário por sua vez envia uma mensagem de resposta com a palavra "comprar". A Vodafone debita dos créditos (esses créditos são créditos de ligações normais de um celular) do celular o valor correspondente ao valor total dos ingressos e em seguida envia para o usuário uma nova mensagem contendo o número de série (código de barras) dos seus ingressos (VODAFONE GROUP, 2010).

De todos os trabalhos encontrados nessa pesquisa, o mais similar com o proposto neste trabalho foi o sistema de venda de ingressos para o *Rock in Rio* em Portugal. Como já demonstrado no item 2.4 deste trabalho, é um sistema em que quase todas as transações são feitas através do dispositivo móvel do usuário, salvo apenas a consulta e solicitação do serviço. As demais transações são atendidas utilizando apenas o dispositivo móvel: *m-payment*, SMS (*Short Message Service*) e *pincode*. O sistema funciona da seguinte forma: o usuário solicita o serviço via Internet utilizando um computador convencional. O sistema por sua vez envia para o celular do usuário uma URL (*Uniform Resource Locator*) para que o usuário acesse com o seu navegador do celular. Dessa forma, através dessa URL, o usuário

---

colocados em uma linha ao longo de 1 polegada (2,54 cm). O valor de DPI também tende a ser relacionado com resolução de imagens, mas está relacionada apenas indiretamente, de forma informal e não matemático.

efetua a compra e pagamento dos ingressos do *Rock in Rio* e recebe uma mensagem com um código *pincode* para cada ingresso comprado. No local do evento existem nas portarias vários aparelhos que lêem e reconhecem esses *pincodes*. Para entrar o usuário deve aproximar seu celular do equipamento, o qual fará a leitura automática do *pincode* e liberar a entrada (ou não) do usuário para o evento (GILSOGAMO, 2010c).

No Quadro 1 é feita uma comparação entre as formas e tipos de serviços citados em cada trabalho correlato.

Característica	Blueticket	ALÔ Ingressos	mTicket	Ingressos *Rock in Rio* Portugal
Serviço disponível na Internet (*desktop*)	Sim	Sim	Não	Sim
Serviço disponível na Telecom	Não	Não	Sim	Não
Serviço disponível na Internet para dispositivos moveis	Não	Não	Não	Não
Consulta do serviço através do dispositivo móvel	Não	Não	Sim	Não
Compra do serviço através do dispositivo móvel	Não	Não	Sim	Sim
Autenticação do serviço através do dispositivo móvel	Não	Sim	Sim	Sim

Quadro 1 – Quadro comparativo de serviços oferecidos

3 DESENVOLVIMENTO

Neste capítulo são descritos todos os passos utilizados para o desenvolvimento deste trabalho, desde a documentação e especificação até a explicação do funcionamento das aplicações desenvolvidas bem como também um exemplo de funcionamento de cada uma delas.

Na primeira etapa são apresentadas as ferramentas utilizadas para o desenvolvimento do trabalho, com uma breve explicação sobre cada uma delas. Na segunda etapa serão apresentadas os requisitos necessários para atingir os objetivos deste trabalho. A terceira etapa apresenta os casos de uso envolvidos bem como suas características, seus cenários e como eles se relacionam.

Na quarta etapa serão apresentados os diagramas de classes e seus principais pontos de funcionamento. Por fim na última etapa será exibido um diagrama de atividades com todo o fluxo de atividades do trabalho bem como um exemplo completo e ilustrado de seu funcionamento.

3.1 FERRAMENTAS DE DESENVOLVIMENTO E BIBLIOTECAS

Para criar os diagramas UML foi utilizada a ferramenta Enterprise Architect. Esta ferramenta é, em geral, utilizada para construir toda e qualquer modelagem e documentação de *software*. No caso deste trabalho ela foi usada para fazer o diagrama de casos de uso, diagrama de classes e o diagrama de atividades.

Como ferramenta de programação e codificação foi utilizado a IDE Eclipse. O Eclipse oferece uma grande variedade de recursos e funcionalidades para uma rápida e fácil codificação. Aliado com os *plugins* de desenvolvimento Web e emulador Android, o Eclipse se tornou uma poderosa ferramenta para o desenvolvimento deste trabalho.

O WebService deste trabalho, desenvolvido em JSP, é hospedado e sustentado pelo servidor de aplicações Web Apache TomCat, desenvolvido pela Apache Fundation. Para mais detalhes sobre o Apache TomCat pode-se consultar o site http://tomcat.apache.org.

Para o desenvolvimento da aplicação cliente, aplicação que é instalada no dispositivo móvel Android do usuário, foi utilizado o do Android. O SDK Android tem todas as

ferramentas, emuladores e bibliotecas Android necessárias para o desenvolvimento de aplicações Android. Para maiores detalhes pode-se consultar o site http://developer.android.com/sdk/index.html.

Também, para o desenvolvimento deste trabalho, foram utilizadas várias bibliotecas externas:

a) no WebService foi utilizado JSP e a biblioteca JSON, que é uma biblioteca para criar serviços web para qualquer aplicativo externo;
b) na aplicação Android cliente, para gerar QRCodes, é utilizado o *core*[10] da API Zxing, que é uma biblioteca utilizada para gerar QRCodes;
c) na aplicação que lê os QRCodes são usadas duas bibliotecas, API Qrcode e JMF. O JMF é utilizado para acessar a câmero do computador e capturar, atravês de uma *streaming*[11] de vídeo, a imagem do QRcode mostrado para a câmera. E o Qrcode é a biblioteca utilizada para efetuar a leitura da imagem capturada pelo JMF e assim, desta forma, extrair os dados contidos no QRcode.

3.2 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O trabalho é composto por três projetos: um WebService, uma aplicação móvel Android e uma aplicação leitora e validadora de QRCodes.

Foram observados e levantados os seguintes requisitos para o WebService:

a) RF01: deve hospedar os vários tipos de eventos como, cinema, teatro e shows, dos quais o usuário pode obter um ingresso digital;
b) RF02: deve armazenar os dados de cada evento como também gerar o *pincode* (número inteiro que representa o código do ingresso);
c) RNF01: deve ser desenvolvido em Java e JSP.

Foram observados e levantados os seguintes requisitos para a aplicação móvel Android:

---

[10] *Core* é um termo em inglês que significa Núcleo ou Centro. Em informática, a palavra Core é mais usada relacionada ao processador. Mais precisamente, Core significa o próprio núcleo de processamento do processador. Porém, ela também pode ser usada para se referir ao conjunto principal de instruções de uma API.
[11] *Streaming* (fluxo, ou fluxo de média (português europeu) ou fluxo de mídia (português brasileiro)) é uma forma de distribuir informação multimídia numa rede através de pacotes. Em *streaming*, as informações da mídia não são usualmente arquivadas pelo usuário que está recebendo a stream, a mídia geralmente é constantemente reproduzida à medida que chega ao usuário.

a) RF03: permitir que o usuário consulte os ingressos disponíveis para venda;
b) RF04: permitir que o usuário compre um ingresso;
c) RF05: criptografar os dados do ingresso obtido, juntamente com o seu *pincode*, na forma de uma imagem QRCode;
d) RF06: permitir que o usuário consulte a qualquer momento os seus ingressos;
e) RF07: permitir que o usuário consulte as novidades e notícias dos eventos;
f) RF08: permitir que o usuário possa excluir ingressos;
g) RNF02: deve ser desenvolvida para a plataforma Android.

Foram observados e levantados os seguintes requisitos para a aplicação leirora e validadora de QRCodes:

a) RF09: efetuar a captura de QRCodes através de uma câmera;
b) RF10: fazer a leitura do QRCode capturado, extraindo as informações contidas no QRCode;
c) RF11: exibir na tela do computador as informações extraídas do QRCode;
d) RNF03: deve ser desenvolvido em Java.

3.3 ESPECIFICAÇÃO

A especificação deste trabalho foi desenvolvida com a utilização das notações e diagramas UML. Primeiro é demonstrado o diagrama dos casos de uso, com suas respectivas relações. Logo depois do diagrama, cada caso de uso é descrito com maiores detalhes, demonstrando os requisitos contemplados por cada um deles e seus cenários principais e alternativos.

Em seguida são demonstrados os diagramas de classes das três aplicações deste trabalho, as quais são: WebService, cliente e leitor de QRCode. Cada diagrama de classes também é seguido por uma explicação detalhada de cada classe e suas relações.

Por fim é visto um diagrama de atividades demonstrando o ciclo completo do processo do sistema, desde a consulta e compra de um ingresso até sua validação pela aplicação leitora de QRCodes.

3.3.1 Casos de uso

Com base nos requisitos levantados neste trabalho foram desenvolvidos oito casos de uso e dois atores. Cada caso de uso representa as principais operações que o usuário ou o sistema pode executar. Além de estarem dispostos de uma forma simplificada e intuitiva, cada cado de uso contempla pelo menos um requisito, possuindo suas pré-condições, cenário principal, fluxos alternativos e pós-condições.

O diagrama de casos de uso segue o padrão UML e é exibido pela Figura 15.

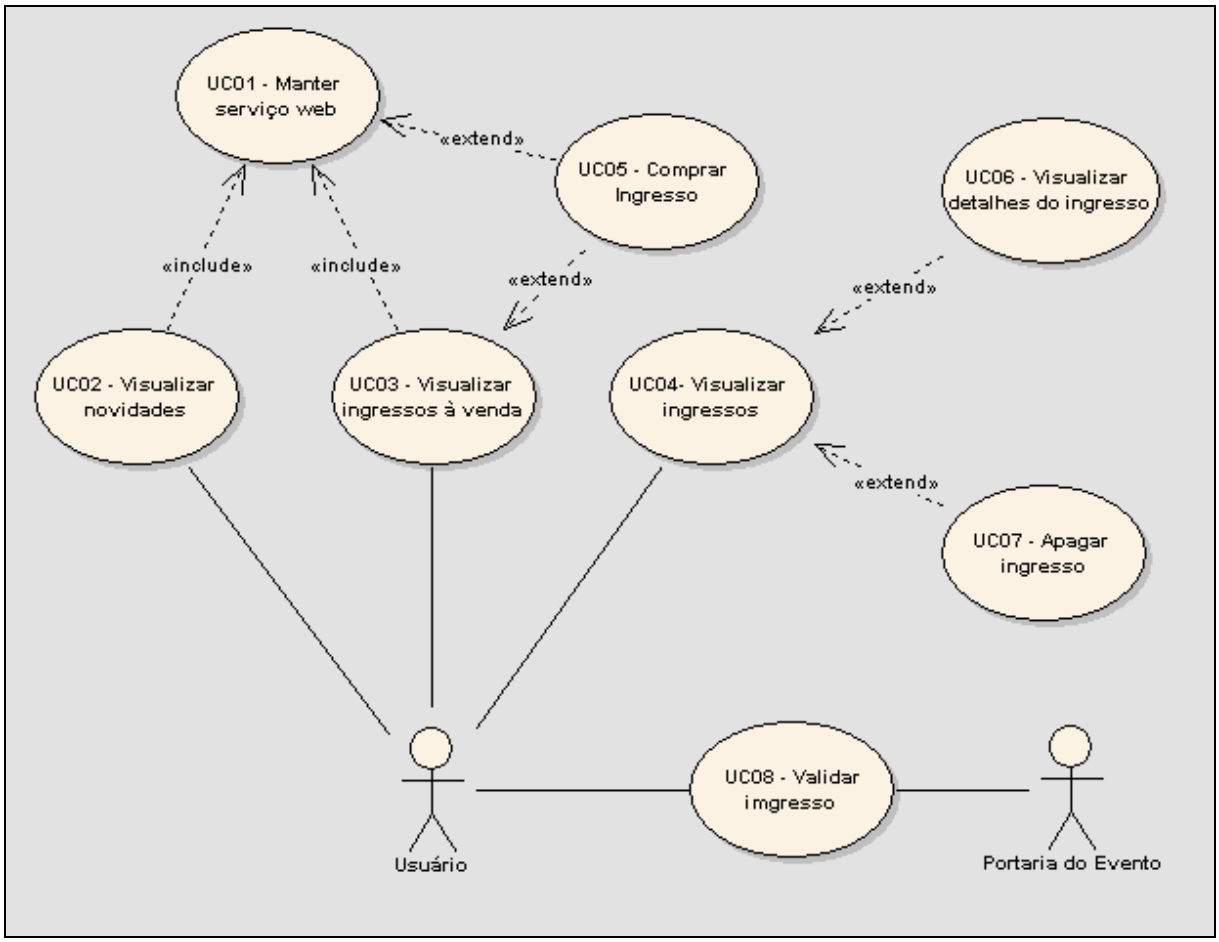


Figura 15 – Diagrama de casos de uso

3.3.1.1 UC01 – Manter serviços web

O caso de uso `Manter serviços web` define o papel do WebService, que é manter os serviços de consulta e compra de ingressos *online*.

Ele é responsável por passar para a aplicação do cliente todas as informações a respeito de ingressos disponíveis bem como receber os pedidos de compra e gerar os *pincondes*

necessários para cada ingresso solicitado pelo cliente. No Quadro 2 há o requisitos contemplados pelo caso de uso e seus respectivos cenários.

UC01 – Manter serviços web: Deve manter os serviços de consulta e compra de ingressos *online*. Bem como gerar *pincondes* e passar todas as informações necessárias dos ingressos à aplicação cliente.	
Requisitos Atendidos	RF01; RF02 e RNF01.
Pré-Condicão	Servidor web Apache Tomcat deve estar ativo.
Cenário Principal	1. Cliente solicita os ingressos disponíveis para venda. 2. WebService informa os eventos com seus títulos e descrições ao cliente.
Cenário Alternativo 1	1. Cliente solicita a compra de um ingresso, informando o número do cartão de crédito e o título do ingresso. 2. WebService gera um *pinconde* e informa ao cliente o título e o *pincode* do ingresso.
Cenário Alternativo 2	1. Cliente solicita as notícias e novidades. 2. WebService informa o texto com as novidades e as notícias ao cliente.
Pós-Condição	Nenhum.

Quadro 2 – UC01 Manter serviços web

3.3.1.2 UC02 – Visualizar novidades

O caso de uso `Visualizar novidades` é responsável em informar ao usuário o conteúdo de notícias e novidades. Essas notícias e novidades são apenas conjuntos de textos para simular uma espécie de propaganda dos eventos disponíveis, como anúncios de filmes em lançamento no cinema, peças de teatros da semana, shows que vão ocorrer na região, entre outros tipos de notícias e novidades.

No Quadro 3 há os requisitos contemplados e seus respectivos cenários.

UC02 – Visualizar novidades: Deve permitir ao usuário que ele consulte as notícias e novidades dos eventos disponíveis para a compra de ingressos.	
Requisitos Atendidos	RF07 e RNF02.
Pré-Condicão	A aplicação cliente deve ter acesso e estar conectada ao WebService.
Cenário Principal	1. Usuário inicia a aplicação cliente Android. 2. Usuário solicita as notícias e novidades. 3. A aplicação cliente solicita ao WebService as notícias e novidades. 4. O WebService recebe o pedido da aplicação cliente e envia o conjunto de textos com as notícias e as novidades. 5. A aplicação cliente exibe para o usuário o conteúdo.
Pós-Condição	Nenhuma.

Quadro 3 – UC02 Visualizar novidades

3.3.1.3 UC03 – Visualizar ingressos a venda

O caso de uso `Visualizar ingressos a venda` é responsável por mostrar ao

usuário a lista de ingressos que podem ser comprados. Esta lista de ingressos é organizada em ordem alfabética, onde cada item dessa lista é uma *string* com o título de cada ingresso.

Ao clicar em algum ingresso da lista o aplicativo cliente exibe para o usuário a descrição do ingresso escolhido e a opção de comprá-lo ou não. A descrição do ingresso pode conter informações como a data e o local do evento, bem como os dados a respeito de preço e demais detalhes que o usuário necessite saber.

No Quadro 4 há os requisitos contemplados e seus respectivos cenários.

UC03 – Visualizar ingressos a venda: Deve permitir que o usuário consulte a lista de ingressos a venda, de forma que seja possível a visualização de seu títulos e descrições.	
Requisitos Atendidos	RF03 e RNF02.
Pré-Condicão	A aplicação cliente deve ter acesso e estar conectada ao WebService.
Cenário Principal	1. Usuário solicita os ingressos a venda. 2. A aplicação cliente solicita ao WebService os ingressos a venda. 3. O WebService recebe o pedido da aplicação cliente e envia a lista de ingressos disponíveis com seus títulos e descrições. 4. A aplicação cliente exibe para o usuário um a lista de ingressos disponíveis. 5. O usuário seleciona um ingresso da lista. 6. A aplicação cliente exibe o título e a descrição do ingresso juntamente com a opção de comprar.
Cenário Alternativo 1	1. Usuário solicita os ingressos a venda. 2. A aplicação cliente solicita ao WebService os ingressos a venda. 3. O WebService recebe o pedido da aplicação cliente e envia a lista de ingressos disponíveis com seus títulos e descrições. 4. A aplicação cliente exibe para o usuário um a lista de ingressos disponíveis.
Pós-Condição	O ingresso selecionado fica apto a compra, podendo dar inicio ao UC05.

Quadro 4 – UC03 Visualizar ingressos a venda

3.3.1.4 UC05 – Comprar ingresso

O caso de uso `Comprar ingresso` é responsável por efetuar as transações necessárias para a compra do ingresso. Para que essa operação seja feira, o usuário informa o número do seu cartão de crédito e o ingresso que deseja comprar. Após isto a aplicação cliente envia o título de identificação do ingresso e o número do cartão do usuário ao WebService. Como resposta o WebService retorna o título do ingresso juntamente com um número *long*, ou seja, o *pincode* gerado para o ingresso solicitado.

Após receber os dados do ingresso, que é o título mais o *pincode* (número *long*) do ingresso, a aplicação cliente transforma toda essa informação em um símbolo QRCode. Por exemplo, supondo que a resposta do WebService seja o título de ingresso `GNC Cinemas - De Volta Para o Futuro` e o *pincode* `606925585666034888`, a aplicação cliente vai gerar um

desenho QRCode usando a seguinte *string*: `GNC Cinemas - De Volta Para o Futuro 606925585666034888`.

No Quadro 5 há os requisitos contemplados e seus respectivos cenários.

UC05 – Comprar ingresso: Deve permitir que o usuário compre o ingresso desejado, no qual ele deve informar o título do ingresso e o número de seu cartão de crédito. Em seguida a aplicação cliente deve gerar um QRCode para que o usuário possa fazer sua devida validação na portaria do evento correspondente ao ingresso obtido.	
Requisitos Atendidos	RF04, RF05 e RNF02.
Pré-Condição	A aplicação cliente deve ter acesso e estar conectada ao WebService.
Cenário Principal	1. Usuário solicita a compra do ingresso. 2. A aplicação cliente solicita ao usuário o título do ingresso e o número de seu cartão de crédito. 3. A aplicação cliente solicita ao WebService um *pincode* para o ingresso a ser obtido. 4. O WebService recebe o pedido da aplicação cliente e reenvia o título do ingresso solicitado e o *pincode* gerado. 5. A aplicação cliente criptografa o título do ingresso mais o *pincode* em um símbolo QRCode.
Cenário Alternativo 1	1. Usuário solicita a compra do ingresso. 2. A aplicação cliente solicita ao usuário o título do ingresso e o número de seu cartão de crédito. 3. Usuário cancela a operação. 4. A aplicação cliente retorna a lista de ingressos disponíveis para a compra.
Pós-Condição	Símbolo QRCode, contendo as informações do título e do *pincode*, salvo na memória do dispositivo móvel do usuário.

Quadro 5 - UC05 Comprar ingresso

3.3.1.5 UC04 – Visualizar Ingressos

O caso de uso `Visualizar ingressos` é responsável por listar todos os ingressos obtidos pelo usuário, permitindo que ele possa acessar os detalhes do ingresso, assim podendo executar as ações descritas pelo UC06 e UC07. Esta lista de ingressos é organizada em ordem alfabética, onde cada item dessa lista é uma *string* com o título de cada ingresso.

Ao clicar em algum ingresso da lista o aplicativo cliente exibe os dados do ingresso conforme o descrito pelo UC06.

No Quadro 6 há os requisitos contemplados pelo UC04 e seus respectivos cenários.

UC04 – Visualizar ingressos: Deve permitir que o usuário visualize uma lista com todos os seus ingressos obtidos, permitindo, desta forma, acessar os detalhes do ingresso.	
Requisitos Atendidos	RF06 e RNF02.
Pré-Condição	Ter gravado na memória do dispositivo móvel pelo menos um ou mais ingressos.
Cenário Principal	1. Usuário solicita a sua lista de ingressos. 2. A aplicação cliente lista os ingressos gravados na memória.
Pós-Condição	Permitir que, ao clicar em um dos ingressos, o usuário possa acessar as informações detalhadas do ingresso selecionado.

Quadro 6 – UC04 Visualizar ingressos

3.3.1.6 UC06 – Visualizar detalhes do ingresso

O caso de uso `Visualizar detalhes do ingresso` é responsável por exibir todas as informações do ingresso obtido pelo usuário. Essas informações são:

a) título do ingresso;
b) descrição do ingresso, onde pode conter data, local e hora do evento;
c) símbolo QRCode do ingresso, contendo o título do ingresso e o seu *pincode*, que será utilizado para validação na portaria do evento.

No Quadro 7 há os requisitos contemplados pelo UC06 e seus respectivos cenários.

UC06 – Visualizar detalhes do ingresso: Deve que o usuário possa visualizar todos os detalhes do ingresso, que é composto pelo título do ingresso, sua descrição e seu símbolo QRCode correspondente.	
Requisitos Atendidos	RF06 e RNF02.
Pré-Condição	Ter gravado na memória do dispositivo móvel pelo menos um ou mais ingressos.
Cenário Principal	1. Usuário solicita a sua lista de ingressos. 2. A aplicação cliente lista os ingressos gravados na memória. 3. Usuário seleciona um ingresso para visualizar seu detalhes. 4. A aplicação cliente exibe o seu título, sua descrição e o seu símbolo QRCode.
Pós-Condição	Possibilitar que o QRCode possa ser lido por uma aplicação leitora de QRCodes.

Quadro 7 – UC06 Visualizar detalhes do ingresso

3.3.1.7 UC07 – Apagar Ingresso

O caso de uso `Apagar ingresso` é responsável pela exclusão do ingresso da memória do dispositivo móvel. Quando o usuário solicita que um determinado ingresso seja apagado a aplicação cliente deve excluir da memória do dispositivo móvel todas as informações do

ingresso, incluindo a imagem (símbolo) QRCode correspondente ao ingresso excluído. A lista de ingressos do usuário também deve ser atualizada, removendo desta lista o ingresso excluído.

No Quadro 8 há os requisitos contemplados pelo UC07 e seus respectivos cenários.

UC07 – Apagar ingresso: Deve permitir ao usuário que ele possa apagar seus ingressos, liberando, desta forma, o espaço de memória ocupado pelo ingresso em seu dispositivo móvel.	
Requisitos Atendidos	RF08 e RNF02.
Pré-Condição	Ter gravado na memória do dispositivo móvel pelo menos um ou mais ingressos.
Cenário Principal	1. Usuário solicita a sua lista de ingressos. 2. A aplicação cliente lista os ingressos gravados na memória. 3. Usuário solicita a exclusão de um ingresso. 4. A aplicação cliente exclui as informações do ingresso, incluindo o seu símbolo QRCode. 5. A aplicação cliente atualiza a lista de ingressos do usuário.
Pós-Condição	A memória interna ocupada pelo ingresso deve ser liberada.

Quadro 8 – UC07 Apagar ingresso

3.3.1.8 UC08 – Validar Ingresso

O caso de uso `Validar ingresso` é responsável por efetuar a leitura de QRCodes e extrair as informações contidas em seu símbolo, possibilitando a validação dos ingressos do usuário. A leitura se dá através de uma imagem do QRCode capturada por uma câmera conectada ao computado onde está a aplicação leitora de QRCodes. Assim, a imagem é varrida, identificando o QRCode e extraindo as informações contidas nele.

Por fim a aplicação leitora de QRCodes deve exibir o conteúdo do QRCode e exibi-la na tela.

No Quadro 9 há os requisitos contemplados pelo UC08 e seus respectivos cenários.

UC08 – Validar ingresso: Deve efetuar a leitura de QRCodes a partir de uma câmera de vídeo e extrair as informações contidas no símbolo. Em seguida as informações extraídas devem ser exibida na tela.	
Requisitos Atendidos	RF09; RF10; RF11 e RNF03.
Pré-Condição	Ter um símbolo QRCode apontado para a câmera.
Cenário Principal	1. Usuário solicita a sua lista de ingressos. 2. A aplicação cliente lista os ingressos gravados na memória. 3. Usuário seleciona um ingresso para visualizar seu detalhes. 4. A aplicação cliente exibe o seu título, sua descrição e o seu símbolo QRCode. 5. Usuário aponta o símbolo QRCode para a câmera da aplicação validadora de QRCodes. 6. A aplicação validadora captura a imagem do QRCode e extrai suas informações. 7. A aplicação validadora exibe na tela o conteúdo contido no QRCode.
Cenário Alternativo 1	No passo 6, se ocorrer algum erro de leitura, o fluxo deve retornar ao passo 5.
Pós-Condição	Nenhuma.

Quadro 9 – UC08 Validar ingresso

3.3.2 Classes do Sistema

Todo o sistema deste trabalho é dividido em três projetos, ou três aplicações, diferentes. Essas aplicações são:

a) aplicação WebService, que é o servidor de serviços web utilizado pela aplicação cliente;
b) aplicação cliente, que é a aplicação que executa dentro do dispositivo móvel do usuário, o qual permite que ele consulte o WebService para a compra e armazenamento de seus ingressos;
c) aplicação *scanner*, que é a aplicação que efetua a leitura de QRCodes, exibindo na tela o conteúdo de cada QRCode lido.

3.3.2.1 Aplicação WebService

O WebService é uma aplicação que simula uma página web, *online*, que mantém todos os eventos (cinema, teatro, shows e etc.) hospedados, os quais podem ser consultado pela própria página web. Além de manter as páginas HTML, a aplicação WebService mantém todos os serviços web necessários para que o usuário possa consultar e comprar ingressos

através de seu dispositivo móvel. Esses serviços são:

a) consultar novidades, no qual o usuário do serviço web pode consultar as notícias e novidades;
b) consultar eventos, no qual o usuário do serviço web pode consultar uma lista com todos os eventos disponíveis;
c) comprar ingresso, no qual o usuário do serviço pode efetuar a compra de um ingresso, recebendo um *pincode* correspondente aquele ingresso.

As Figuras 16, 17, 18 e 19 apresentam os diagramas das classes definidas para a aplicação WebService.

No pacote `data` há as classes de dados. Essas classes são apenas duas, a classe `Xhtml` e a classe `LinkMatcher`. A classe `Xhtml` é a classe que representa uma página HTML. Esta classe tem todas as *tags* HTML necessários para criar uma página completa, e a ela pode-se adicionar várias outras *tags* HTML para formar a página web. Já a classe `LinkMatcher` é apenas usada para a identificação e criação de *links* da página HTML (que é formada pela classe `Xhtml`).

Na Figura 16 há o diagrama de classes do pacote `data` com seus respectivos atributos e métodos.

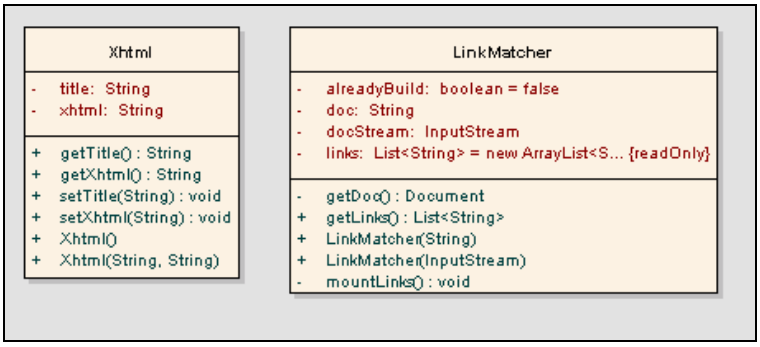


Figura 16 – Diagrama das classes do pacote **`data`**

No pacote `model` há as classes que representam os dados modelo da aplicação WebService. No caso desta aplicação, a única classe de modelo necessária é a classe `Event`. Outras classes podem vir a ser criadas, porem, para atender os objetivos deste trabalho, a única classe necessária é a classe `Event`. As demais classes usadas são apenas para se adequar ao *Design Patterns* DAO (*Data Access Object*), que é uma forma padrão de definir o modo em que as classes acessam os dados das entidades usadas.

Na Figura 17 há o diagrama de classes do pacote `model` com seus respectivos atributos e métodos.

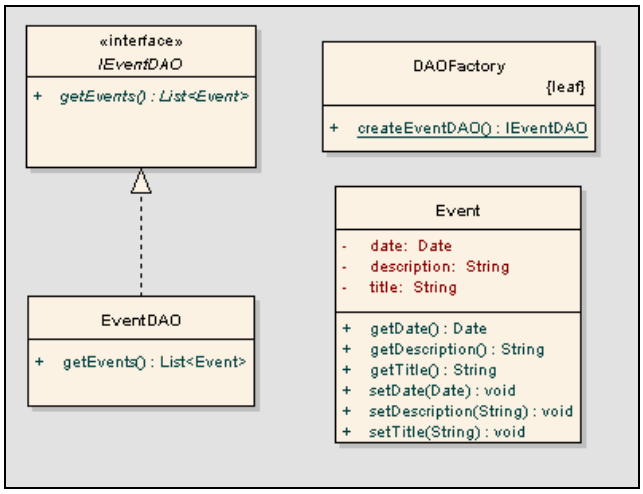


Figura 17 – Diagrama de classes do pacote `model`

O pacote `web` contém todas as classes de controle web da aplicação WebService. As classes de controle tem as funções de receber as requisições HTTP (*HyperText Transfer Protocol*), que pode ser uma simples página ou um dos serviço, e responder de acordo com as suas funções. Além de responder as requisições HTTP elas possuem os padrões para a criação de formulários (ou páginas) web e as demais funções utilizadas pelos serviços, como a montagens das respostas de cada serviço e a geração de *pincodes*.

Na Figura 18 há o diagrama de classes do pacote `web` com seus respectivos atributos e métodos.

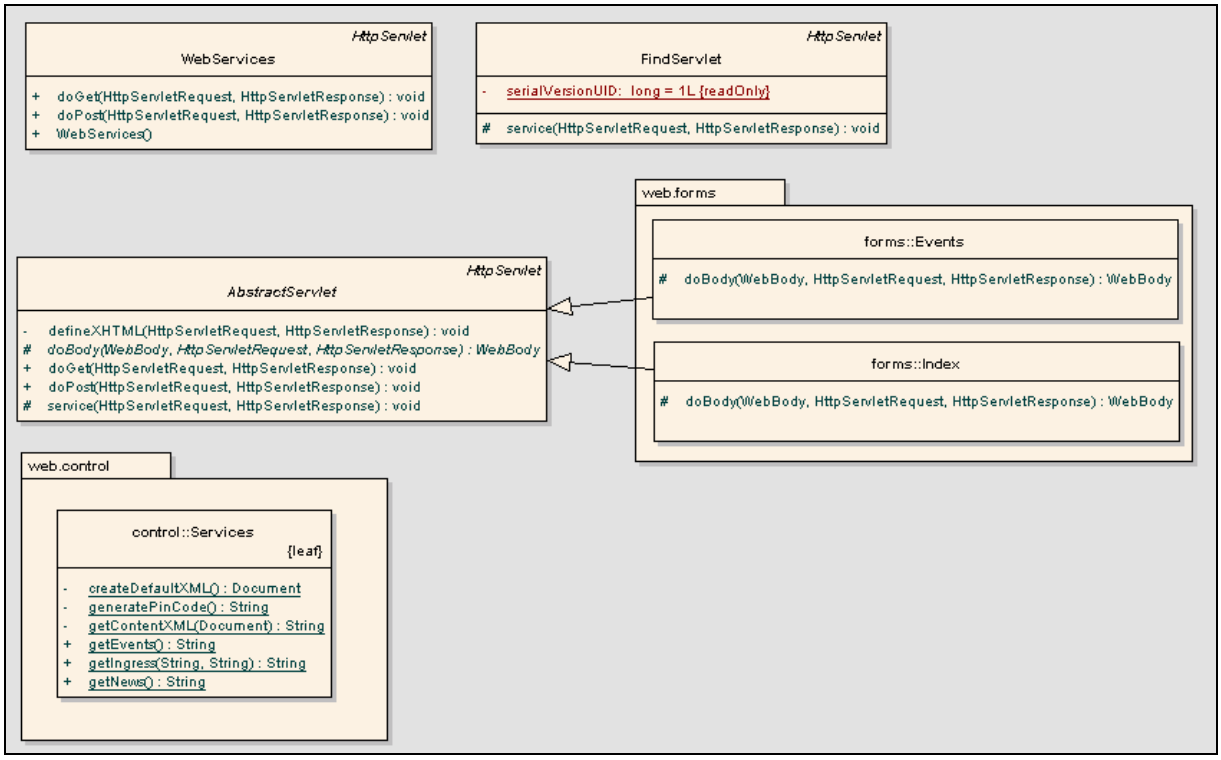


Figura 18 – Diagrama de classes do pacote `web`

A classe `FindServlet` recebe chamadas HTTP externas com seus respectivos dados

(como URL e demais parâmetros) e procura, através de reflexão, a página web correspondente à URL solicitada. Caso a página não exista, então o WebService entende que a solicitação é de um serviço. Assim a chamada HTTP é passada para a classe `WebServices`, que identifica o tipo de serviço solicitado e retorna a devida resposta de acordo com cada serviço.

A classe `Services` é responsável por executar as operações pertinentes ao WebService a respeito de cada serviço. Por exemplo, supondo que uma aplicação externa faça uma chamada HTTP solicitando a compra de um ingresso, o WebService primeiro verificará se esta chamada é correspondente a uma das páginas web. Como é uma chamada de serviço, então o WebService passará a chamada para a classe `WebServices`, que identificará qual serviço está sendo solicitado. Após ser identificado que é o serviço de compra, a classe `WebServices` chama o método `getIngress` da classe `Services` correspondente ao serviço solicitado. Assim a classe `Services` pode executar as operações necessárias e retornar a devida resposta para a classe `WebServices`, que por sua vez devolve esta resposta para a aplicação externa que fez a chamada inicial do serviço.

Já a classe `AbstractServlet` apenas define um padrão para a construção de páginas web mantidas pelo WebService. Todos as páginas web devem estender a esta classe, como é o caso das duas classes que definem duas páginas web, que são as classes `Events` e `Index`. A classe `Events` é a classe que define a página HTML que exibe os eventos disponíveis no WebService, e a classe `Index` é a classe que define a pagina inicial (ou *default*) do *site*.

Por fim, no pacote `web.components`, há as classes que definem componentes de formulários HTML, que podem ser botões, títulos, parágrafos, tabelas e demais componentes comumente utilizadas na criação de páginas HTML.

Na Figura 19 há o diagrama de classes do pacote `web.components` com seus respectivos atributos e métodos.

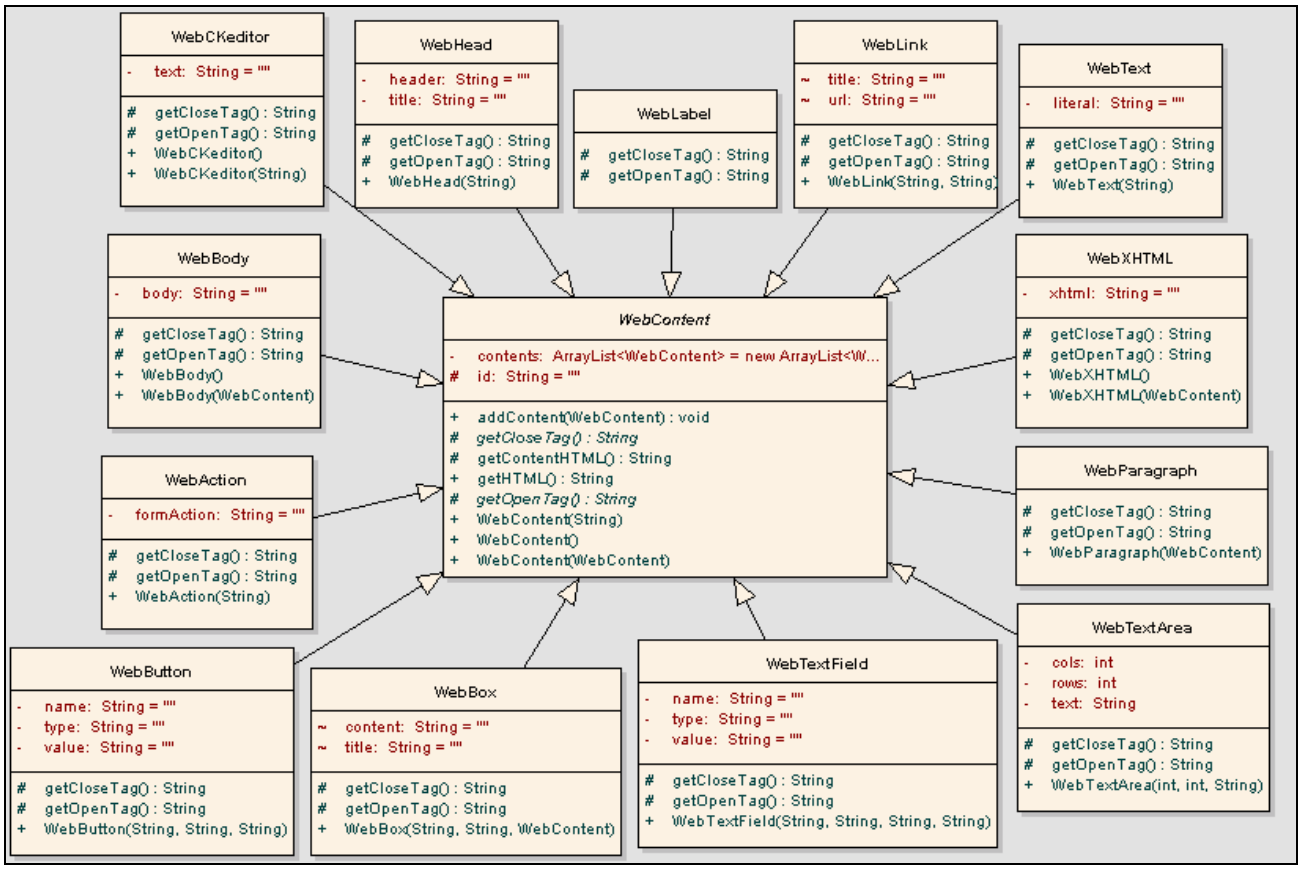


Figura 19 – Diagrama de classes do pacote `web.components`

3.3.2.2 Aplicação Cliente

A aplicação cliente é a aplicação que executa dentro do dispositivo móvel do usuário. É esta aplicação que permite ao usuário acessar os três serviços do WebService. Assim a aplicação cliente é responsável por:

a) permitir que o usuário consulte e leia as notícias e novidades;
b) listar os eventos disponíveis para a compra de ingressos;
c) efetuar a compra do ingresso e armazenar suas informações no dispositivo móvel;
d) gerar o símbolo QRCode de cada ingresso para efetuar sua leitura nas portarias dos eventos.

Na Figura 20 há o diagrama de classes do pacote `com.lazulli.control` com seus respectivos atributos e métodos.

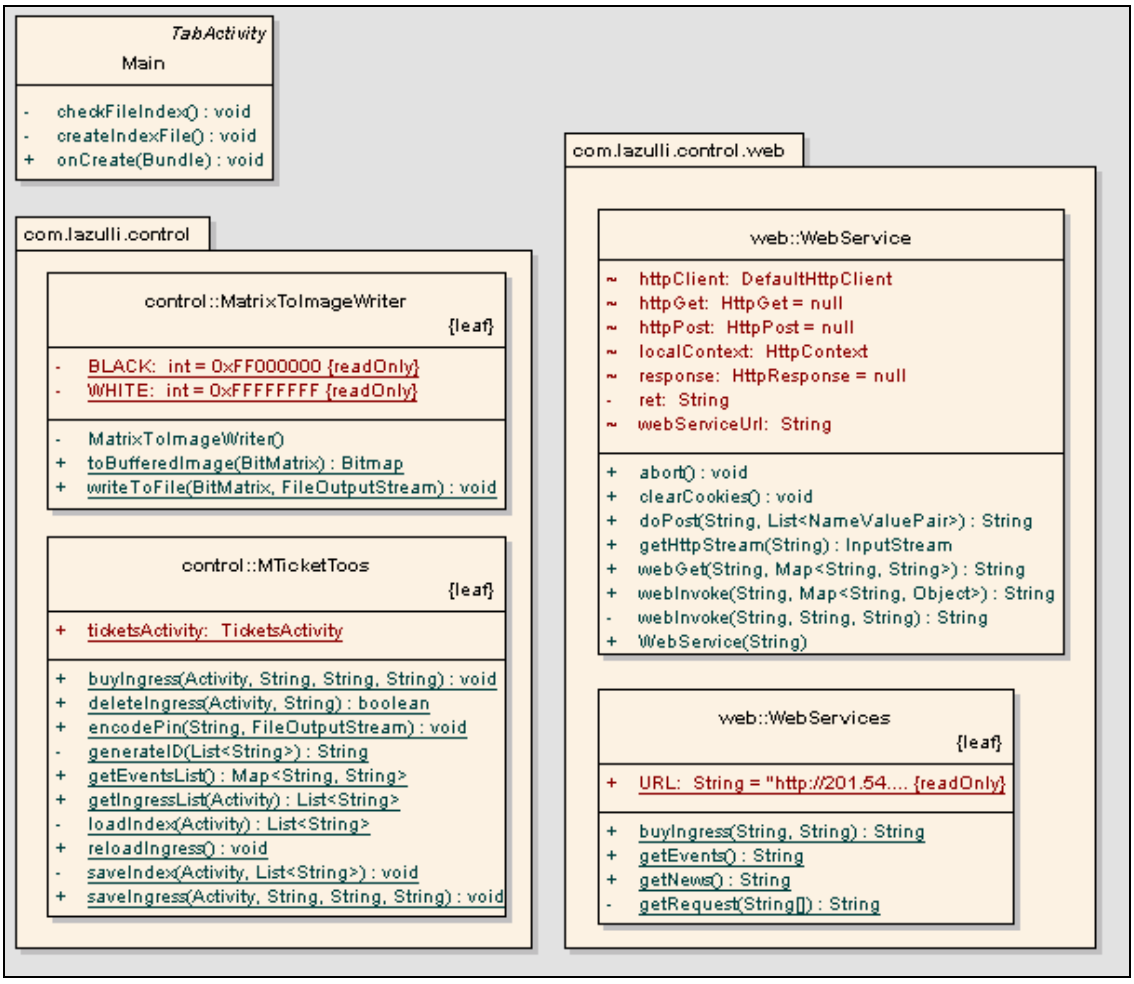


Figura 20 – Diagrama de classes do pacote `com.lazulli.control`

A classe `WebServices` é a classe que executa as solicitações de serviços web. Para cada serviço, a classe `WebServices`, utiliza a classe `WebService` para efetuar as chamadas via *requests* HTTP. Assim ela executa todas as chamadas de serviços e obtém as respostas de cada um destes serviços.

A classe `MticketToos` é a classe responsável por todos os controles internos da aplicação, como por exemplo a encriptação dos ingressos na forma de imagens QRCodes, a gravação dos ingressos no dispositivo móvel, o gerenciamento dos ingressos, entre outros controles necessários. Além dos controles internos ela também serve de interface para as classes de formulário (*Activitys*) que, por sua vez, fazem a interface com o usuário.

No pacote `com.lazulli.view` temos as classes de interface ao usuário que são extenções da classe abstrata `Activity`. Ao todo são usados apenas três classes, a classe `NewsActivity`, `EventsActiviy` e `TicketsActiviy`.

Na Figura 21 há o diagrama de classes do pacote `com.lazulli.view` com seus respectivos atributos e métodos.

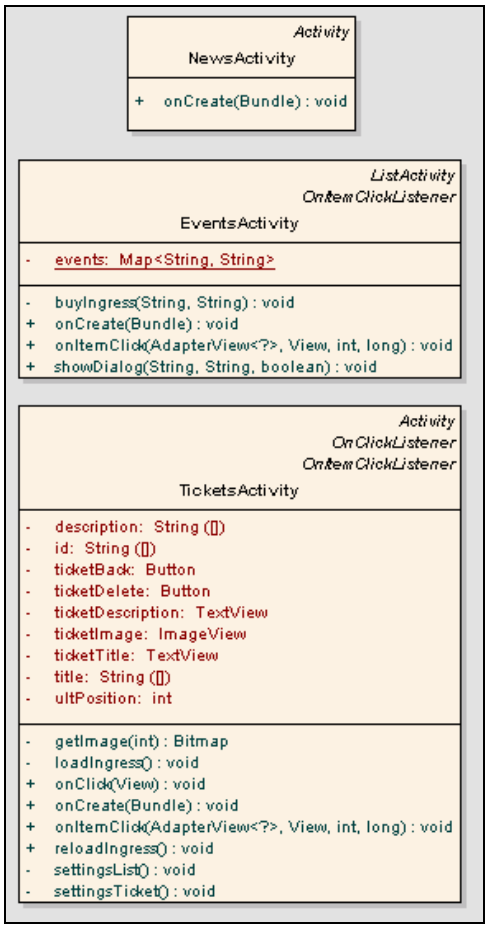


Figura 21 – Diagrama de classes do pacote `com.lazulli.view`

A classe `NewsActivity` exibe ao usuário um conjunto de textos. Esse conjunto de textos são as notícias e novidades sobre os eventos, como uma espécie de propaganda dos eventos. Já a classe `EventsActiviy` é um formulário com uma lista, onde cada item da lista é um evento disponível para a compra de ingressos.

Por fim a classe `TicketsActiviy` é a classe na qual o usuário visualiza os detalhes do seu ingresso e a imagem do símbolo QRCode correspondente a tal ingresso.

3.3.2.3 Aplicação leitora de QRCodes

A aplicação leitora de QRCodes é responsável por ler os QRCodes exibidos nas telas dos dispositivos móveis dos usuário. Esta aplicação fica executando nos computadores das portarias de cada evento. A aplicação leitora faz a captura do QRCode através de uma câmera de vídeo e extrai seus dados que, em seguida, são mostrados na tela do computador validando assim a entrada do usuário.

Na Figura 22 há o diagrama de classes do pacote `com.lazulli.control` com seus

respectivos atributos e métodos.

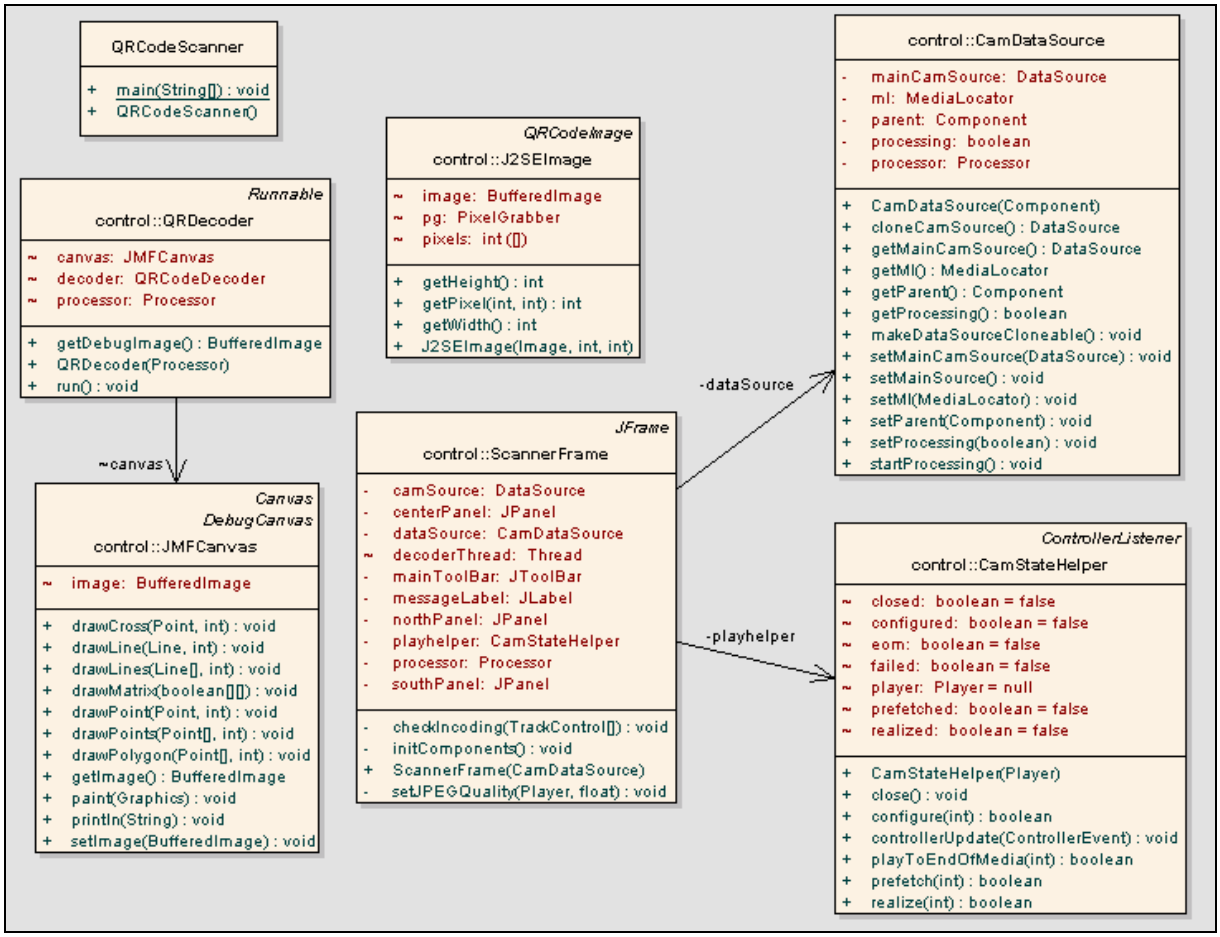


Figura 22 – Diagrama de classes do pacote `com.lazulli.control`

A classe `ScannerFrame` é a classe que faz a interface com o usuário. Ela é uma extensão da classe `JFrame` a qual possui os recursos e utilitários de vídeo necessários para que seja possível acompanhar os processos de leitura do QRCode. Para isso ela utiliza duas classes, a classe `CamDataSource` e `CamStateHelper`. A classe `CamDataSource` carrega todos os *drivers* e recursos necessários para o uso da câmera de vídeo. Já a classe `CamStateHelper` carrega os recursos de estatísticas e dados da câmera, que podem ser, por exemplo, quantidade de *frames* por segundo, padrão de cores e demais detalhes da câmera.

A classe `JMFCanvas` é uma classe que cria um *canvas* através da biblioteca JMF (*Java Media Framework*). E é através desse *canvas* que é feito a captura do QRCode exibido na câmera. A classe `JMFCanvas` captura a imagem da câmera e cria uma imagem `.jpg` através da classe `J2SEImage`. Por fim, a classe `QRDecoder` faz a leitura da imagem e decodifica o QRCode extraindo os dados contidos nele.

3.3.3 Diagrama de atividades

Na Figura 23 é apresentado um diagrama de atividades demonstrando todo o processo deste trabalho, que vai desde a consulta e compra do ingresso até a sua validação na portaria do evento.

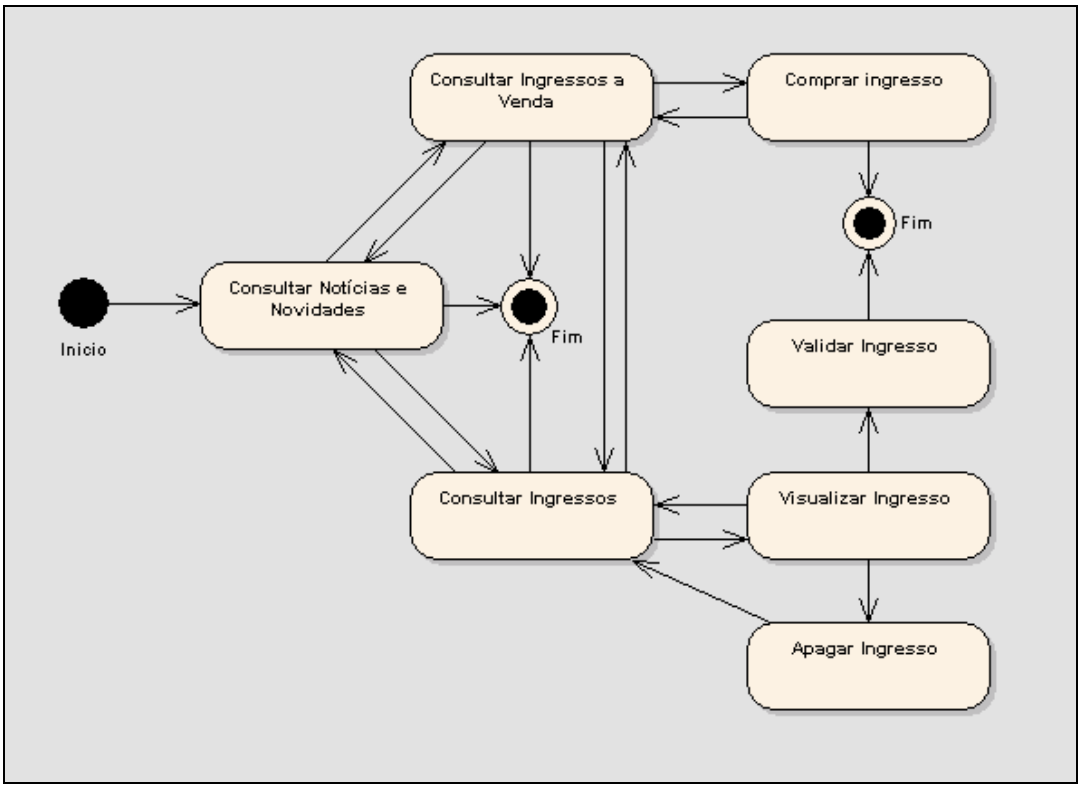


Figura 23 – Diagrama de atividades do processo completo

3.4 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação. Em seguida será demonstrada a forma de implementação de cada uma das três aplicações existente neste trabalho, que são: aplicação servidora, aplicação cliente e aplicação leitora de QRCodes. Por fim será exibida a operacionalidade da implementação de todas as aplicações com algumas imagens ilustrativas das telas para um melhor acompanhamento.

3.4.1 Técnicas e ferramentas utilizadas

Em virtude da característica deste projeto, foram adotados duas técnicas de implementação fundamentais. A primeira é a programação orientada a objetos, visando modularizar e encapsular as rotinas para facilitar a manutenção e o reaproveitamento de código fonte.

A segunda técnica utilizada foi a comunicação entre cliente e servidor através do protocolo HTTP aliado ao recurso XML. Isso se faz necessário em virtude da baixa capacidade de memória e processamento dos dispositivos móveis.

Embora atualmente existam aparelhos móveis com uma grande capacidade de processamento, quanto mais se economizar seus recursos maior será a eficiência da aplicação. Sendo assim, em vez de usar os recursos padrões para o consumo de WebServices do Android (que consome bastante memória do aparelho), foi utilizado a comunicação via HTTP.

No que diz respeito às ferramentas, conforme já se viu anteriormente, foram utilizados a IDE Eclipe (com os *plugns* de emulação Android e servidores web), a biblioteca de mídia *Java Media Framework*, a biblioteca de WebServices Json e as bibliotecas de QRCodes Zxing e Qrcode.

3.4.2 Desenvolvimento das aplicações

A seguir apresentam-se todos os passos e detalhes relativo a implementação de cada uma das três aplicações.

3.4.2.1 Aplicação servidora

A aplicação servidora foi desenvolvida em duas formas. Primeiro a forma no qual é utilizado o JSP, feito para a construção das páginas HTML e demais estruturas acessadas via *browser*. E a segunda forma, onde foi aplicado a API Json, para a utilização dos serviços web do WebService.

Como foi visto no tópico `3.3.2.1 Aplicação WebService`, todas as requisições HTTP são direcionadas para a classe `FindServlet`, que através de reflexão procura a página

correspondente ao URL recebido no *request*.

No Quadro 10 pode-se ver o trecho do código que executa a reflexão para procurar a página solicitada pelo *request* HTTP.

```
protected void service(HttpServletRequest request, HttpServletResponse respose)
        throws ServletException, IOException {
    String forms = "";
    try {
        /* Obtem string de request do cliente. */
        String servletName = request.getRequestURI();

        /* Dismonta a string para encontrar o nome da classe(servlet) */
        String[] projectName = servletName.split("/");
        forms = projectName[projectName.length - 1];

        /* Carrega classe */
        Class<?> servlet = Class.forName(forms);

        /* Instacia objeto da classe e chama o método doGet. */
        AbstractServlet servletRequest = (AbstractServlet) servlet.newInstance();
        servletRequest.service(request, respose);
    } catch (ClassNotFoundException e) {
        System.err.println("Servlet não encontrado: " + e.getMessage());
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        HttpSession s = request.getSession();
        Object o = s.getAttribute("clear");
        if (o != null) {
            boolean b = ((Boolean) o).booleanValue();
            if (b) {
                s.setAttribute("editing", null);
                s.setAttribute("deleting", null);
            }
        }
    }
}
```

Quadro 10 – Trecho do código que buscar pelas páginas web solicitadas

Caso o URL solicitado é de alguma página existente então é instanciado um `AbstractServlet` e seu método `service` é chamado para a montagem da página correspondente. Dessa forma o HTML é montado e devolvido para o usuário gerador do *request*.

Se a solicitação HTTP for de um serviço então quem captura o *request* é a classe `WebServices`. Dessa forma a classe `WebServices` identifica qual serviço deve ser chamado e executa os processos necessários (através da classe `Services`) para a exclusão do serviço. Após isso é devolvido para o solicitante do *request* o resultado da operação.

No Quadro 11 pode-se ver o trecho do método `doPost` da classe `WebServices` que recebe as solicitações de serviços.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    System.out.println("Server request: " + new Date().toString());
    HashMap<String, Object> hm = new HashMap<String, Object>();

    if (request.getParameter("news") != null) {
        System.out.println("Comando: " + request.getParameter("news"));
        String resService = Services.getNews();
        hm.put("news", resService);
    }else if (request.getParameter("events") != null) {
        System.out.println("Comando: " + request.getParameter("events"));
        String resService = Services.getEvents();
        System.out.println("Resposta: " + resService);
        hm.put("events", resService);
    }else if (request.getParameter("buyIngress") != null) {
        System.out.println("Comando: " + request.getParameter("buyIngress"));
        String resService = Services.getIngress(request.getParameter("eventTitle"),
                request.getParameter("cardCode"));
        System.out.println("Resposta: " + resService);
        hm.put("pinCode", resService);
    }

    JSONObject json = JSONObject.fromObject(hm);
    response.setContentType("application/json");
    PrintWriter out = response.getWriter();
    out.print(json);
    out.flush();
}
```

Quadro 11 – Trecho do código que recebe as solicitações de serviços

O resultado das operações executadas pelos serviços sempre são devolvidos na forma de um texto XML. Isso é feito para facilitar a troca e estruturação de dados, já que a aplicação cliente consome o serviço web através do protocolo HTTP e não através das bibliotecas tradicionais para consumo de WebService. O único serviço que não utiliza a estrutura XML é a serviço `news` (notícias e novidades), já que o resultado de sua operação não passa de uma cadeia de caracteres sem a necessidade de estruturação. Já os serviços `events` e `buyIngress` fazem uso destes princípios.

Sempre que o usuário solicita o serviço `events`, que é o serviço de consulta aos eventos disponíveis, é retornado como resultado da operação um XML que segue a estrutura do Quadro 12.

```xml
<!-- Estrutura do XML de eventos. -->
<mticket>
    <events>
        <event title="título1" description="descrição1"/>
        <event title="título2" description="descrição2"/>
    </events>
</mticket>

<!-- Exemplo de um XML real. -->
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<mticket>
    <events>
        <event
            title="GNC - Sucker Punch"
            description="Sinopse: Ambientando na década de 50, uma garota é internada..."/>
        <event
            title="GNC - Sherlock Holmes"
            description="Genero: ação e aventura. Diretor: Guy Ritchie."/>
        <event
            title="Carlos Gomes - Orquestra Filarmônica Philadelfia"
            description="Data/Hora: Domingo, 06 de novembro, às 20h.Informações: Ingressos R$ 20,00."/>
    </events>
</mticket>
```

Quadro 12 – Exemplo de XML retornado ao consultar o serviço `events`

Por padrão em todo o XML tem-se a *tag* `<mticket>`, em seguida vem uma *tag* que identifica uma lista de eventos, que no caso é a *tag* `<events>`. Por fim, para cada evento, é gerado uma *tag* `<event>`, contendo dois atributos, `title` e `description`, no qual contém o título do evento e sua descrição.

Para o serviço de compra de ingressos, ou seja, o serviço `buyIngress`, o retorno da operação é um XML que segue a estrutura conforme demonstrado no do Quadro 13.

```xml
<!-- Estrutura do XML de compra. -->
<mticket>
    <event>Título do Evento</event>
    <card>0000-0000-0000-0000</card>
    <pin>00000000000000000</pin>
</mticket>

<!-- Exemplo de um XML real. -->
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<mticket>
    <event>GNC - Sucker Punch</event>
    <card>1234-2346-7456-9977</card>
    <pin>6069255856660348888</pin>
</mticket>
```

Quadro 13 – Exemplo de XML retornado ao consultar o serviço `buyIngress`

Seguindo o padrão, como primeira *tag* tem-se `<mticket>`, em seguida vem três *tags*. A primeira *tag*, definida como `<event>`, possui o título do evento. A segunda *tag*, definida como `<card>`, possui o número do cartão do usuário. E a última *tag*, definida como `<pin>`, possui o *pincode* gerado para aquele ingresso.

Sendo assim, é dessa forma que se dá a implementação do aplicativo servidor com seus três serviços básicos proposto por esse trabalho.

3.4.2.2 Aplicação cliente

A aplicação cliente, ou seja, a aplicação que executa no dispositivo móvel Android do usuário, é composta por três atividades (ou *activitys*) distintas. Essas atividades são:

a) `NewsActivity`;

b) `EventsActivity`;

c) `TickesActivity`.

Todas essas *activitys* são extensões da classe `Activity` e são responsáveis por criar as interfaces com o usuário. Os componentes de tela das atividades podem ser definidas tradicionalmente via programação ou mapeadas através de um arquivo XML, que mais tarde pode ser acessados pelo código fonte através de uma espécie de "mapa de referências". Por exemplo, as três atividades são organizadas sobre uma quarta atividade que tem seus componentes definidos por um XML. Essa quarta atividade é o própria classe `Main` que estende a classe `TabActivity`, que por sua vez também é uma atividade.

No Quadro 14 é exibido o conteúdo do XML que define o *layout* e o mapa de componentes da atividade `Main` (`TabActivity`).

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <TabHost xmlns:android="http://schemas.android.com/apk/res/android"
3      android:id="@android:id/tabhost"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent">
6      <LinearLayout
7          android:orientation="vertical"
8          android:layout_width="fill_parent"
9          android:layout_height="fill_parent"
10         android:padding="5dp">
11         <TabWidget
12             android:id="@android:id/tabs"
13             android:layout_width="fill_parent"
14             android:layout_height="wrap_content" />
15         <FrameLayout
16             android:id="@android:id/tabcontent"
17             android:layout_width="fill_parent"
18             android:layout_height="fill_parent"
19             android:padding="5dp" />
20     </LinearLayout>
21 </TabHost>
```

Quadro 14 – XML que define o *layout* das abas na classe `Main`

Observando rapidamente ve-se que a *tag* `<TabHost>` define que essa atividade é uma atividade que reúne abas. Em seguida temos as *tags* `<LinearLayout>`, que definem o *layout* (que no caso é linear), e por fim a *tag* `<TabWidget>`, que define que esse conjunto de abas são abas padrões do Android com *layout* também linear.

As três atividades principais, contidas nas abas, além de serem as interfaces do usuário

elas também são responsáveis por chamar determinados métodos da classe `MticketTools`, que executa operações internas essenciais para os processos de consulta e compra de ingressos.

A classe `NewsActivity` é a atividade mais simples dentre as três. Ela apenas exibe o conteúdo de notícias. Quando a atividade `NewsActivity` é chamada ela executa o método `getNews` da classe `WebServices`. Este método retorna uma *string* contendo o conteúdo de notícias o qual é exibido na tela.

Os componentes de atividade `NewsActivity` são os únicos que são criados via codificação tradicional, ou seja, o componente é criado e instanciado diretamente do código. O Quadro 15 mostra este procedimento.

```
10 public class NewsActivity extends Activity {
11     public void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13 
14         /*
15          * Busca conteúdo de noviticias.
16          * */
17         String news = WebServices.getNews();
18         TextView textview = new TextView(this);
19         textview.setText(news);
20         setContentView(textview);
21     }
22 }
```

Quadro 15 – Criando componentes tradicionalmente via código

Pode-se ver na linha 18 um componente do tipo `TextView` sendo instanciado e inicializado. Em seguida é adicionado a *string* contendo o conteúdo de novidades retornado pelo WebService e, por fim, o componente é adicionado na tela principal (linha 20).

A classe `EventsActivity` é a atividade que exibe para o usuário uma lista contendo todos os eventos disponíveis para a compra de ingressos. Quando o usuário seleciona a aba de eventos e inicia esta atividade, a classe `EventsActivity` chama o método `getEventsList` da classe `MTicketToos`. Este método consulta o sertiço `events` do WebService e retorna um `HashMap`, ou seja, um mapa com todos os eventos disponíveis e suas respectivas descrições. Assim a lista de eventos é montada de forma que cada item desta lista é um evento, o qual é identificado pelo seu título.

O layout e o mapa de componentes da atividade `EventsActivity` é definida por um arquivo XML, da mesma forma que a atividade principal da classe `Main`. No Quadro 16 temos o XML que define o layout da atividade `EventsActivity`.

XML que define a lista.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
     android:orientation="vertical"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent">
 <ListView
     android:id="@+id/lista"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"/>
</LinearLayout>
```

XML que define um item de lista.

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:textSize="16sp" >
</TextView>
```

Quadro 16 – XML que define o layout ta atividade `EventsActivity`

Ao clicar sobre algum item da lista a atividade `EventsActivity` exibe uma janela modal com o título do evento, a descrição do evento e a opção de comprar o ingresso. Caso o usuário solicite a compra, então é exibido uma nova janela modal para que o usuário insira o número de seu cartão de créditos.

Quando o usuário insere o número do seu cartão e confirma a compra, então, a atividade `EventsActivity` envia os dados de compra ao WebService através da chamado do método `buyIngress` da classe `MTicketToos`. Os dados enviados são o título do ingresso e o número do cartão de crédito. Como resposta a chamada do serviço o WebService retorna um XML (conforme foi visto o tópico `3.4.2.1 Aplicação servidora`) contendo o *pincode* gerado para aquele ingresso.

Com os dados de retorno do serviço em mãos, a aplicação cliente passa a etapa de salvamento das informações. Os dados salvos, no total, são três:

a) o título do ingresso;

b) a descrição do ingresso;

c) a imagem QRCode com o *pincode* criptografado.

O processo para que todas essas informações sejam salvas e organizadas na memória do aparelho móvel é dada da seguinte forma...

Primeiro é criado um arquivo texto (extensão `.txt`), o qual é utilizado para salvar as informações textuais do ingresso, como é o caso do título e da descrição. Caso o arquivo já exista então ele é apenas reaberto. Este arquivo texto, definido pela aplicação como

`index.txt`, guarda um registro de ingresso a cada linha. Ou seja, cada linha do arquivo `index.txt` corresponde a um registro de ingresso. A configuração ou formatação do registro é composta primeiro por um número inteiro, que corresponde ao seu identificador, em seguida por seu título e por fim a sua descrição. Cada um dos campos do registro (identificação, título e descrição) são separados pelo caractere de controle '`@`'. E cada registro, ou seja, cada linha, também tem um caractere de controle para separá-los, que é o caractere '`@!`'.

No quadro 17 temos um exemplo do arquivo `index.txt`.

Ingress.xml | index.txt

```
1  @!68243@GNC - Sucker Punch@Genero: ação, fantasia, drama e steampunk.Diretor: Zack Snyder.
2  @!11100@Show - Scavengers Festival@Bandas: Not Falling, Rhapsody of Fire, Audio Machine e Theory of a Deadman
3
4
5
```

Quadro 17 – Exemplo de arquivo `index.txt`

Por fim é gerado a imagem QRCode contendo o *pincode* do ingresso. A imagem é salva na extensão `.jpg` e é nomeada com o número do identificado correspondente ao seu ingresso. Por exemplo, considerando o exemplo do Quadro 17 haveriam duas imagens de QRCodes gravadas, a imagem `68243.jpg` e `11100.jpg`.

A Figura 24 demonstra a imagem `68243.jpg`.

Figura 24 – QRCode da imagem `68243.jpg`

Para gerar o QRCode é utilizado o *core* da API Zxing, onde basta passar para ela o conteúdo que se deseja transformar em QRCode que ela se encarrega de gerar a imagem. No Quadro 18 é mostrado o trecho do código que gera a imagem QRCode.

```
public static void encodePin(String content, FileOutputStream file) {
    Log.i("Lazulli", "Gerando QRCode para o conteudo: " + content);
    Charset charset = Charset.forName("ISO-8859-1");
    CharsetEncoder encoder = charset.newEncoder();
    byte[] b = null;
    try {
        java.nio.ByteBuffer bbuf = encoder.encode(CharBuffer.wrap(content));
        b = bbuf.array();
    } catch (CharacterCodingException e) {
        System.out.println(e.getMessage());
    }

    String data = null;
    try {
        data = new String(b, "ISO-8859-1");
    } catch (UnsupportedEncodingException e) {
        Log.e("Lazulli", e.getMessage());
    }

    BitMatrix matrix = null;
    int h = 100;
    int w = 100;
    com.google.zxing.Writer writer = new QRCodeWriter();
    try {
        matrix = writer.encode(data, com.google.zxing.BarcodeFormat.QR_CODE, w, h);
    } catch (com.google.zxing.WriterException e) {
        Log.e("Lazulli", e.getMessage());
    }

    try {
        MatrixToImageWriter.writeToFile(matrix, file);
        Log.i("Lazulli", "Arquivo imagem QRCode salvo com sucesso.");
    } catch (IOException e) {
        Log.e("Lazulli", e.getMessage());
    }
}
```

Quadro 18 – Trecho do código que gera o QRCode

A última atividade da aplicação cliente é a classe `TicketsActivity`. Esta atividade é responsável por listar os ingressos do usuário e exibir seus detalhes com o seus respectivos QRCodes. Da mesma forma que a atividade `EventsActivity` ela é uma lista, onde cada item da lista corresponde a um ingresso obtido. Diferente da lista do `EventsActivity`, a lista da atividade `TicketsActivity` é montada com base no arquivo `index.txt`. Quando a atividade é iniciada o método `getIngressList` da classe `MticketToos` é chamado. Este método retorna uma lista de *strings* (`ArrayList<String>`) com os títulos de cada ingresso comprado pelo usuário. A partir desta lista é montado a lista com todos os ingressos.

Quando o usuário seleciona um dos ingressos da lista, então, a aba da atividade `TicketsActivity` é remontada de forma dinâmica, removendo a lista de ingressos e adicionando os componentes de detalhes do ingresso.

A nova interface utiliza dois `TextView`, uma para exibir o título e outro para a descrição do ingresso, juntamente com um `ImageView` e dois `Button`. O `ImageView` é usado para exibir o QRCode. Já os dois `Buttons`, um é usado para retornar a lista de ingressos e o outro para apagar o ingresso.

O Quandro 19 mostra o código que monta dinamicamente a interface de detalhes.

```
private void settingsTicket() {
    setContentView(R.layout.ticket);
    ticketBack = (Button) findViewById(R.id.buttonBack);
    ticketBack.setOnClickListener(this);
    ticketDelete = (Button) findViewById(R.id.buttonDelete);
    ticketDelete.setOnClickListener(this);

    ticketImage = (ImageView) findViewById(R.id.image);
    ticketTitle = (TextView) findViewById(R.id.title);
    ticketDescription = (TextView) findViewById(R.id.description);
}


private void settingsList() {
    setContentView(R.layout.list);
    ListView listView = (ListView) findViewById(R.id.lista);
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, R.layout.list_item, title);
    listView.setAdapter(adapter);
    listView.setTextFilterEnabled(true);
    listView.setOnItemClickListener(this);
}
```

Quadro 19 – Código que montar interface dinâmica

Precisamente, é o método `settingsTicket` que monta dinamicamente a interface do ingresso. Já o método `settingsList` volta para a lista de ingressos caso o usuário clique no botão de retornar.

Basicamente, além da atividade `TicketsActivity` ser utilizada para que o usuário possa consultar seus ingressos, é essa a tela usada para que o QRCode seja exibido e apontado para a câmera de vídeo onde o aplicação validadora de QRCodes valida os ingressos.

3.4.2.3 Aplicação leitora de QRCodes

A aplicação leitora é a aplicação que efetua a leitura dos QRCodes através de uma câmera de vídeo, extraindo desta maneira o conteúdo de cada QRCode. Esta aplicação foi implementada com base em duas etapas, primeiro a imagem é capturada pela câmera e em segundo é feito a leitura do QRCode contido na imagem.

Para capturar a imagem da câmera é utilizado a API JMF. O JMF permite que a aplicação leitora possa acessar a câmera de vídeo do computador com os *drivers* corretos, possibilitando assim a captura de imagens. Quando a aplicação leitora é iniciada e a câmera ativada, a cada segundo, uma foto é batida. A imagem da foto é passada para um método que chamado `decoder` da classe `QRCodeDecoder`. Esse método, pertencente a uma API chamada Qrcode (utilizada para leitura de QRCode) tenta identificar se nesta imagem existe algum QRCode. Se nenhum QRCode for identificado então a imagem é descartada e a próxima foto batida, após 1 segundo, é avaliada. E assim este processo segue sucessivamente até que em alguma das imagens contenha um QRCode.

Caso em alguma imagem exista um QRCode, então, este QRCode é lido pelo método `decoder` e retorna a *string* contida neste QRCode. Após a leitura a string é impressa na tela de forma simples, através do comando `System.out.println`.

De uma forma bastante simples, a API Qrcode identifica que a imagem contém um QRCode através das três guias de referência do QRCode (conforme foi visto no tópico `2.4.4 Varredura em 360 graus`). Para facilitar a identificação das guias de referência e dos módulos do QRCode (quadrados pretos e brancos) a API Qrcode utiliza um filtro de contraste extremo nas imagens passadas para a decodificação. Esse filtro de contraste faz com que os pixels escuros fiquem mais escuros e os claros mais claros.

A Figura 25 ilustra a aplicação deste filtro na imagem original.

Figura 25 – Comparação da imagem original com a filtrada

Embora a aplicação do filtro de contraste aumente a capacidade de leitura, ainda assim, as vezes acabam ocorrendo ruídos na leitura, fazendo com que a informação retirada do QRCode não seja 100% coerente com a informação correta contida no QRCode.

A Figura 26 mostra um exemplo de uma leitura com ruídos.

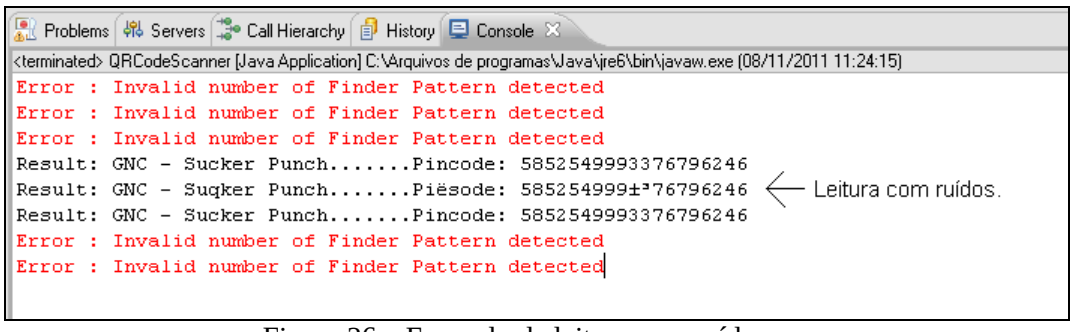


Figura 26 – Exemplo de leitura com ruídos

A aplicação leitora foi implementada de uma forma bastante simples, apenas lendo o

QRCode e imprimindo na tela o resultado. Porém, se houver a necessidade, ela pode ser aprimorada fazendo com que, por exemplo, ela desenhasse um quadro verde para ingressos válidos e um vermelho para ingressos inválidos. Ou ainda poderia ser acoplada a uma catraca ou cancela, a qual liberaria a entrada ou não de forma automática.

3.4.3 Operacionalidade da implementação

A operacionalidade da implementação é demonstrada de três ângulos diferentes. O primeiro ângulo é quando o usuário consulta o *site* mantido pelo servidor web, onde pode-se verificar através de um *browser* a página inicial e os eventos disponíveis.

O segundo ângulo é do ponto de vista do usuário utilizando a aplicação móvel para acessar o conteúdo do servidor web e efetuar a consulta e compra de ingressos, bem como a validação de seus ingressos obtidos.

Por último, o terceiro ângulo, é do ponto de vista do operador da portaria, o qual verificará o conteúdo extraído do QRCode do usuário.

3.4.3.1 Verificando o conteúdo através do *site*

O usuário tem sempre a opção de verificar o conteúdo dos eventos através do *site* mantido pelo servidor web. Por exemplo, utilizando um *browser* comum o usuário pode acessar o *site* e consultar os eventos disponíveis.

As Figuras 27 e 28 mostram as duas páginas do *site*, que é a página de apresentação e a página com a lista dos eventos e suas descrições.

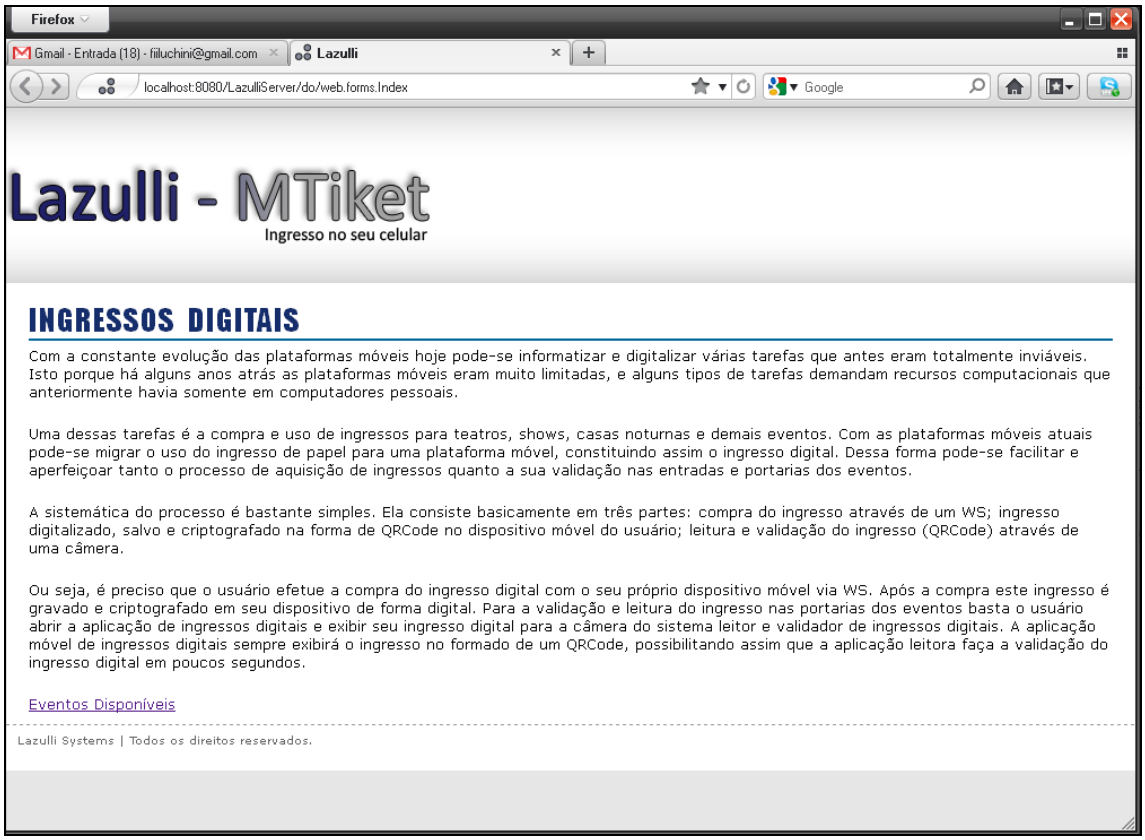


Figura 27 – Página inicial do *site*

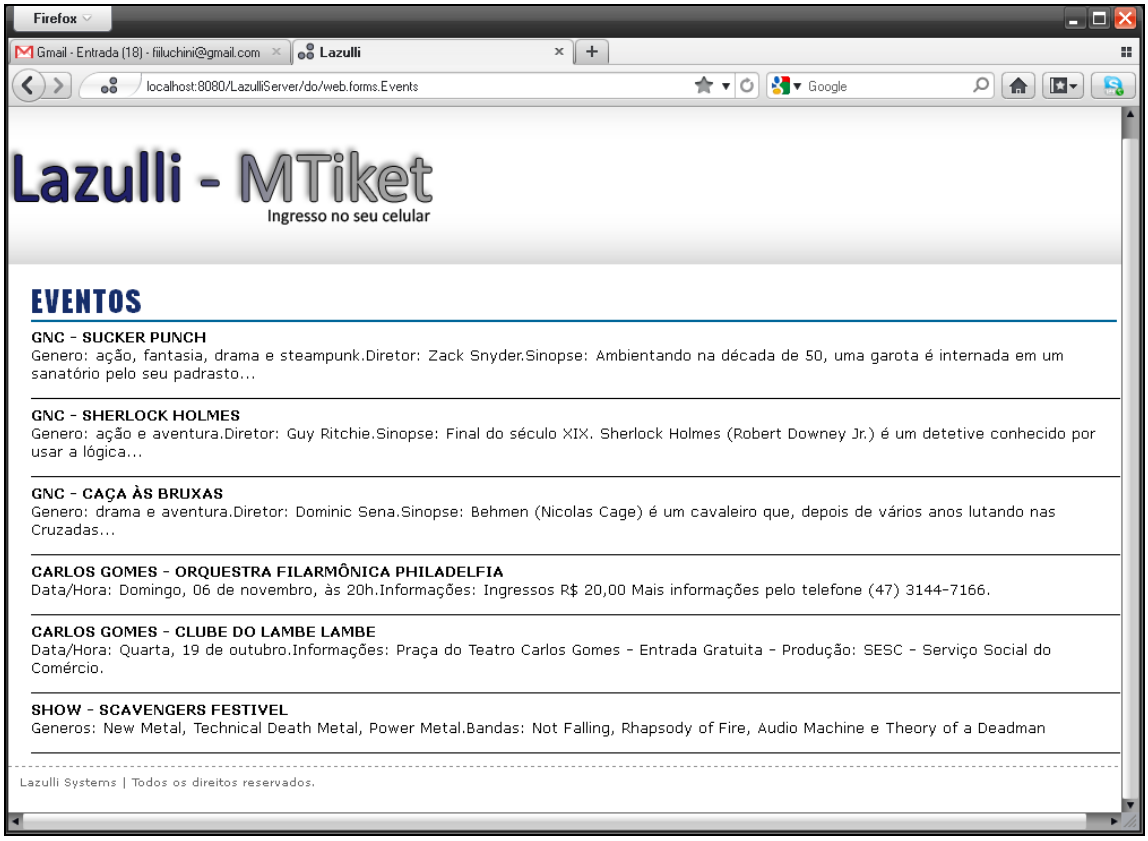


Figura 28 – Página de consulta aos eventos disponíveis

3.4.3.2 Consultando e comprando ingressos

Através de um dispositivo móvel Android pode-se consultar os ingressos disponíveis para a venda com o mesmo princípio da consulta através do *site*. Porém, além de consultá-los o usuário também pode comprá-los.

Quando a aplicação é iniciada o usuário visualiza três abas. A primeira aba possui o contúdo de notícias e novidades e, por padrão, esta é a primeira aba exibida. A segunda aba contém a lista de de ingressos disponíveis e, por fim, a última aba contém os ingressos do usuário.

A Figura 28 mostra o ponto inicial da aplicação, com a aba notícias já sendo exibida.

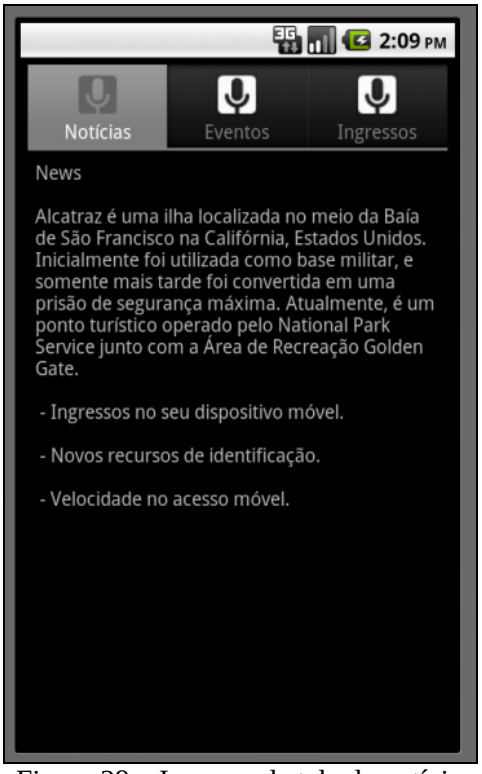


Figura 29 – Imagem da tela de notícias

Na aba seguinte há a lista com todos os ingressos disponíveis. Ao selecionar algum dos ingressos, uma janela aparece mostrando sua descrição completa com a opção de compra. Se o ingresso for comprado então uma nova janela aparece solicitando o número do cartão de créditos (que no caso deste trabalho não passa de uma compra emulada da forma mais simples possível criada apenas a título de demonstração).

Por fim o usuário confirma a compra e seu ingresso já fica gravada em seu aparelho móvel. A Figura 30 mostra os passos executados pelo usuário para efetuar a compra de um ingresso.

Figura 30 – Passos executados na compra de um ingresso

Para consultar seus ingressos basta o usuário ir para a última aba. Nesta última aba o usuário vê a lista de todos os ingressos obtidos e, ao selecionar algum deles, seus detalhes aparecem na tela juntamente com o QRCode do ingresso.

A Figura 31 mostra os passos para visualiza os detalhes do ingresso.

Figura 31 – Passos para visualizar os detalhes do ingresso

Para validar o QRCode basta o usuário apontar este QRCode para a câmera do computador da portaria do evento que possua a aplicação validadora. O tópico à seguir apresenta uma exemplificação deste processo.

3.4.3.3 Validando um ingresso

A validação de ingressos segue um processo simples. Basta que o usuário selecione os detalhes de seu ingresso e aponte para a câmera do computador. A aplicação leitora fará a leitura do ingresso e exibirá na tela. Para auxiliar na visualização do que a câmera está captando a aplicação validadora exibe uma janela com o *canvas* da câmera. O resultado da leitura é impresso no console.

Na Figura 32 há o resultado deste processo.

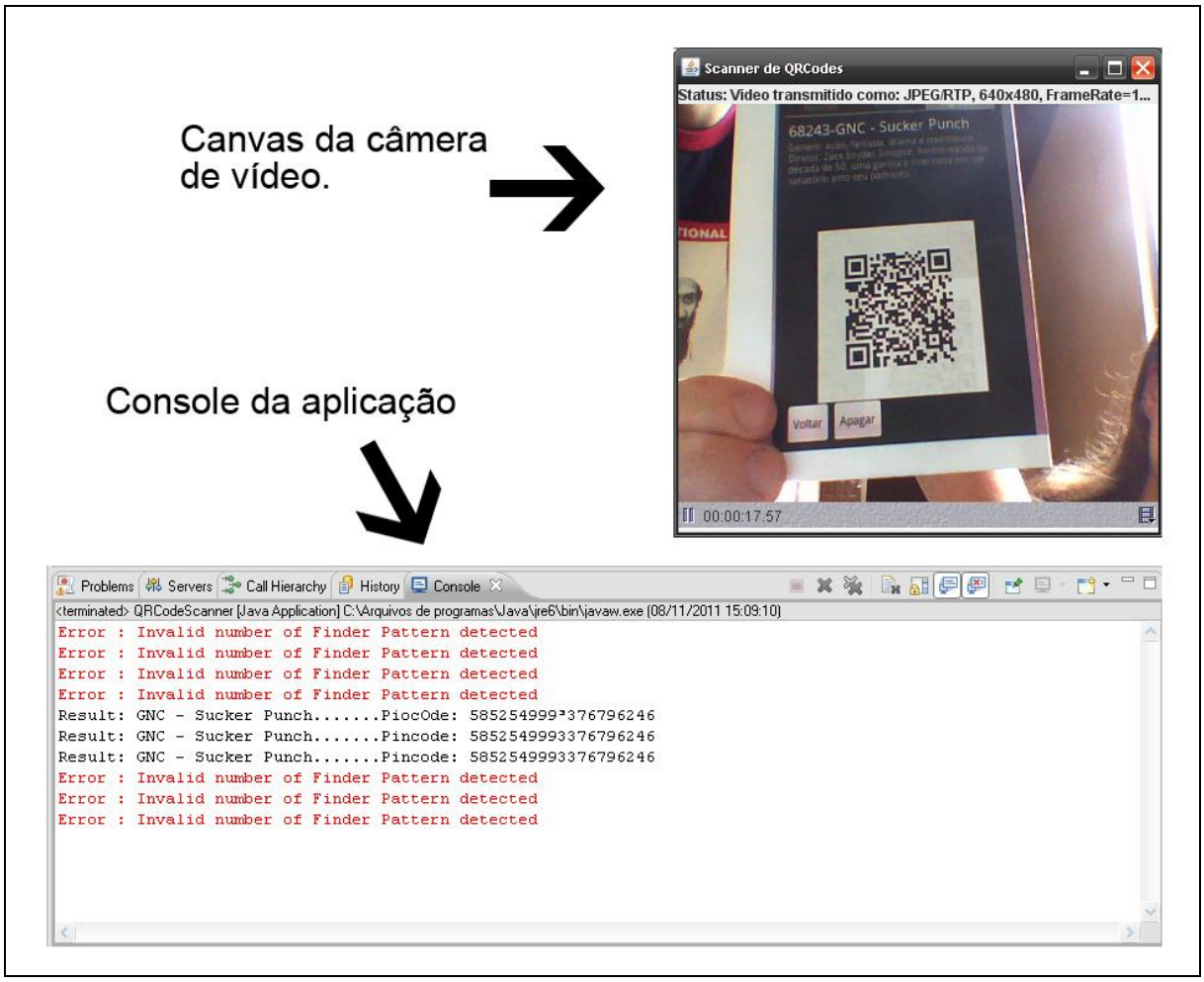


Figura 32 – Exemplo de leitura e validação de QRCodes

3.5 RESULTADOS E DISCUSSÃO

O presente trabalho apresenta uma forma fácil e eficiente para a aquisição de ingressos de forma digital em aparelhos móveis conectados a rede da Internet. Comparado com a forma de venda de ingressos através de *sites*, como o Blueticket e ALÔ Ingressos, este trabalho mostrou mais praticidade, facilidade e velocidade, pois nos casos do Blueticket e do ALÔ Ingresso todas as operações são feitas através de um computador, exceto a validação.

Em relação à validação, este trabalho também tem maiores vantagens do que as formas de validação encontradas nos trabalhos correlatos. Por exemplo, a forma de validação do *mTicket* da Vodafone e do ALÔ Ingresso, embora sejam semelhantes ao deste trabalho, necessita que o usuário entregue o seu aparelho por alguns instantes para que o funcionário da portaria faça a leitura do código de barras exibido no visor. Em contrapartida, na forma de

validação deste trabalho, o usuário por sí só pode fazer a validação de seu ingresso apontando o QRCode para uma câmera, sem precisar entregar o seu aparelho pessoal para nenhum funcionário.

Uma segunda vantagem e relação ao *mTicket* da Vodafone é que todos os processos ocorrem no meio digital, não necessitando que o usuário tenha uma conta telefônica ou créditos em seu celular para utilizar o serviço. Basta que o usuário possua acesso a Internet e um cartão de créditos.

Além das vantagens já citadas existe também a questão do QRCode. Enquanto a forma de armazenamento dos ingressos dos trabalhos correlatos são na forma de códigos de barras a deste trabalho é na forma de QRCodes. Isso oferece uma série de vantagens, pois pode-se armazenar muito mais informação em um QRCode do que em um código de barras, permitindo assim uma infinidade de controles embutidos dentro dos próprios ingressos.

Dos trabalhos correlatos existentes, o único que se assemelhou a este, foi à forma de vendas dos ingressos para o *Rock in Rio* de Portugal em 2008. Todos os processos são muito similares, com a diferença que o passo inicial começa em um computador convencional, onde é solicitado o serviço, e os demais passos, como as operações de consulta e compra dos ingressos, são feitas através do aparelho móvel acessando uma página web `.mob`. Além das diferenças já citadas, outra pequena diferença é na forma da validação dos ingressos, que se dá através de sinal de rádio ao aproximar o aparelho ao leitor (diferente da validação visual dos QRCodes).

Através dos resultados obtidos e destas breves comparações pode-se concluir que um sistema de ingresso digital, seguindo esses princípios, facilitaria bastante o acesso aos eventos de uma cidade. Por exemplo, imaginando que o usuário se encontra num final de semana na rua, e deseja frequentar um cinema, basta apenas que ele retire o seu aparelho do bolso e acesso o aplicativo de ingressos. Assim ele pode consultar os filmes em cartaz e comprar seu ingresso imediatamente, evitando que o usuário acabe de deslocando em vão para o cinema e, ao chegar, descubra que os ingressos para a sua cessão já estavam esgotados.

Além das vantagens que os usuários dos ingressos possuem, as empresas vendedoras de ingresso também podem obter uma série de vantagens com este tipo de serviço. Pois elas podem economizar em dois pontos. Primeiro vão evitar a gastos com papel na fabricação e controle dos ingressos, como também podem economizar tempo e funcionários nas portarias. Pois os próprios usuários podem fazer sua validação, de forma rápida, nas portarias dos eventos. Sem falar no aumento da segurança evitando a passagem de usuários com ingressos suspeitos ou falsificados.

4 CONCLUSÕES

O presente trabalho teve por objetivo principal a construção de um sistema que implementasse os conceitos de uso de ingressos, transformando o convencional ingresso de papel por um rápido e eficiente ingresso digital.

Com o desenvolvimento deste trabalho viu-se como é possível transformar a compra de um ingresso de papel em uma fácil consulta com o aparelho móvel seguida da aquisição de um ingresso digital, evitando filas ou o deslocamento do usuário até os pontos de vendas.

Cada objetivo proposto neste trabalho foi definido e desenvolvido conforme o esperado. O WebService foi relativamente simples de desenvolver, haja visto que esse tipo de tecnologia já é bastante comum e madura. Já no desenvolvimento da aplicação móvel Android, as dificuldades foram maiores, pois como sua tecnologia é recente vários recursos de seu SDK estão em constante mudança evolutiva. Além do SDK evoluindo constantemente, a forma de desenvolvimento para uma plataforma móvel é bastante diferenciada em relação ao desenvolvimento de uma aplicação comum. Os recursos têm uma série de limitações e regras a serem obedecidas, bem como a forma de codificação, como as UI desenvolvidas em XML.

A aplicação validadora, embora não tão complexa quanto a aplicação móvel, também tem uma complexidade razoável. Utilizar JMF em conjunto com as APIs QRCODE requer uma séria de ajustes para que seu intercâmbio funcione corretamente. Dessa forma, aliando o JMF com o QRCODE, foi possível ter um resultado bastante satisfatório no *scanner* de QRCodes.

Uma das limitações da aplicação deste trabalho é o fato de necessitar do acesso a Internet. Mas com o advento da Internet 3G e os vários pontos de acesso a Internet Wi-Fi, que hoje em dia existem até em terminais de ônibus e restaurantes, é muito fácil conseguir acesso a Internet com aparelhos móveis. Esse quadro permite que não só o sistema criado por esse trabalho possa ser prático como também faz com que cada vez mais as aplicações dependentes da Internet migrem para aparelhos móveis.

Uma segunda limitação é que a aplicação, até o presente momento, não saiu da plataforma Android, impedindo que usuários que possuam dispositivos com outras plataformas, como é o caso do iPhone, não possam utilizá-lo.

Por fim uma última limitação importante é a respeito do pagamento móvel. O presente trabalho não possui sistema de pagamento real com macanismos seguros. O que existe é apenas um formulário simples onde o usuário insere o número do seu cartão de crédito, que é

enviado ao servidor sem nenhuma validação ou conferência. Algumas APIs foram pesquisadas para o desenvolvimento de pagamento móvel para estre trabalho, mas todas as APIs encontradas, como a da empresa Pagseguro, não possuem ainda suporte para o Android.

4.1 EXTENSÕES

Como sugestões de extensões para esse trabalho, tem-se:

a) melhorar as interfaces com o usuário, criando interfaces mais amigáveis e simpáticas;
b) fazer com que o conteúdo de notícias e novidades seja um página HTML, permitindo criar links e *banners* animados;
c) transformar cada ingresso em uma imagem personalizada que contenha em algum ponto, de forma discreta, o QRCode. Assim os ingressos de papel literalmente estariam dentro dos aparelhos móveis sendo exibidos por uma galeria de imagens;
d) desenvolver uma versão da aplicação cliente para iPhone;
e) implementar mecanismos de segurança e de pagamento móvel;
f) aprimorar o *site* do servidor web. Com a criação de mais páginas e a opção de compra pelo *site*, onde o ingresso com o QRCode possa ser enviado ao dispositivo móvel via USB (*Universal Serial Bus*).

REFERÊNCIAS BIBLIOGRÁFICAS

ABLESON, **IBM**, developer works. [S.l.], 2009. Disponível em: www.ibm.com/developerworks/br/library/os-android-devel/. Acesso em: 05 out. 2011.

ALÔ Ingressos: soluções tecnológicas de acesso. [S.l.], 2010. Disponível em: <www.aloingressos.com.br/site/ingresso_celular.asp>. Acesso em: 18 set. 2010.

ANDROID DEVELOPERS. [S.l.], 2010. Disponível em: <developer.android.com/guide/basics/what-is-android.html>. Acesso em: 05 out. 2011.

BLUETICKET. [Florianópolis?], 2010. Disponível em: <www2.blueticket.com.br>. Acesso em: 18 set. 2010.

DIETRICH, Andrea. Grupo Pão de Açúcar aposta em mobile marketing nas redes de supermercados. **Mobilepedia**, [S.l.], 20 ago. 2010. Entrevista concedida a Renata M. Nascimento. Disponível em: <www.mobilepedia.com.br/entrevistas/grupo-pao-de-acucar-aposta-em-mobile-marketing-nas-redes-de-supermercados>. Acesso em: 11 set. 2010.

_____. **Gartner realiza pesquisa sobre internet móvel.** [S.l.], 2010c. Disponível em: <www.mobilepedia.com.br/noticias/gartner-realiza-pesquisa-sobre-internet-movel>. Acesso em: 06 set. 2010.

GILSOGAMO, **Mobile Pedia**, Referência em Mobile Marketing. [S.l.], 2010a. Disponível em: <www.mobilepedia.com.br/noticias/gartner-realiza-pesquisa-sobre-internet-movel>. Acesso em: 06 set. 2010.

GILSOGAMO, **Mobile Pedia**, Referência em Mobile Marketing. [S.l.], 2010b. Disponível em: <www.mobilepedia.com.br/noticias/materia-sobre-mobile-payment-no-estadao>. Acesso em: 11 set. 2010.

GILSOGAMO, **Mobile Pedia**, Referência em Mobile Marketing. [S.l.], 2010c. Disponível em: <www.mobilepedia.com.br/cases/vodafone-%E2%80%93-ingressos-do-rock-in-rio-pelo-celular-mobile-marketing>. Acesso em: 11 set. 2010.

_____. **Matéria sobre mobile payment no Estadão**. [S.l.], 2010d. Disponível em: <www.mobilepedia.com.br/noticias/materia-sobre-mobile-payment-no-estadao>. Acesso em: 11 set. 2010.

MIWA, Renata. **Uso de m-payment em táxis**. [S.l.], 2009. Disponível em: <www.mobilepedia.com.br/cases/uso-de-m-payment-em-taxis-mobile-marketing>. Acesso em: 11 set. 2010.

NASCIMENTO, Renata M. **Mobile Pedia**, Referência em Mobile Marketing. [S.l.], 2010a. Disponível em: <www.mobilepedia.com.br/entrevistas/grupo-pao-de-acucar-aposta-em-mobile-marketing-nas-redes-de-supermercados>. Acesso em: 11 set. 2010.

OPEN HANDSET ALLIANCE, **Alliance**. [S.l.], 2010. Disponível em: <www.openhandsetalliance.com/oha_overview.html>. Acesso em: 05 out. 2011.

QRCODE.COM. [S.l.], 2010. Disponível em: www.denso-wave.com/qrcode/aboutqr-e.html. Acesso em: 07 out. 2011.

_____. **Vodafone**: ingressos do Rock in Rio pelo celular (mobile marketing). [S.l.], 2010. Disponível em: <www.mobilepedia.com.br/cases/vodafone-%E2%80%93-ingressos-do-rock-in-rio-pelo-celular-mobile-marketing>. Acesso em: 11 set. 2010.

VODAFONE GROUP. **mTicket**. [S.l.], 2010. Disponível em: <www.vodafone.co.nz/services/mticket.jsp>. Acesso em: 11 set. 2010.