UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM TECNOLOGIA JAVA E DESENVOLVIMENTO PARA DISPOSITIVOS MÓVEIS

ILSON JOSÉ SABINO DA SILVA

PROTÓTIPO DE UM SISTEMA DE VISTORIA VEICULAR ATRAVÉS DE WEBSERVICES

MONOGRAFIA DE ESPECIALIZAÇÃO

CURITIBA
2013

ILSON JOSÉ SABINO DA SILVA

PROTÓTIPO DE UM SISTEMA DE VISTORIA VEICULAR ATRAVÉS DE WEBSERVICES

Monografia de Especialização apresentada ao Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de Especialização em Tecnologia Java e Desenvolvimento para Dispositivos Móveis.

Orientador: Prof. João Alberto Fabro

CURITIBA

2013

AGRADECIMENTOS

A Deus que me deu forças para terminar este trabalho. Ao professor Dr. João Alberto Fabro pela sua dedicação e pela orientação na elaboração do trabalho. Aos professores do Curso de Especialização em Tecnologia Java e Desenvolvimento para Dispositivos Móveis pelos momentos de aprendizado e inspiração.

Portanto, desde já peço desculpas àquelas que não estão presentes entre essas palavras, mas elas podem estar certas que fazem parte do meu pensamento e de minha gratidão porque contribuíram de forma direta e indireta no desenvolvimento deste trabalho.

Agradeço a minha família, especialmente minha mãe que sempre me incentivou a estudar e a minha esposa pelo carinho, amor e compreensão.

RESUMO

O projeto apresenta a integração de informações veiculares provindas de bases de dados de órgãos responsáveis por divulgá-las, que são pagas por consulta, com dados que são registrados na execução da atividade do serviço de vistoria veicular. O registro dos diversos dados relativos a um determinado veículo que sofre vistoria em uma base única é uma maneira adequada de armazenar as informações. Além disto, no caso de uma nova consulta para o mesmo veículo, os dados já estão disponíveis para consulta e comparação. Empresas que utilizam o serviço de vistoria, tendo acesso as bases de dados sobre registro de veículos, também possuem a vantagem de redução de custos, pois invés de coletar os dados em formulários de papel para posterior digitação no sistema computacional, pode permitir o acesso direto à base de dados, através do uso de *smartphones*, e de um formato digital próprio para transmissão. Outra vantagem é a otimização do tempo de envio e recebimento dos dados remotamente, dispensando o deslocamento para a sede visando digitação dos dados. Com a utilização deste serviço a empresa pode também obter vantagens pela não necessidade de re-consultar as bases de dados pagas para acessar dados de um veículo já consultado previamente, em vistoria anterior, pois os dados ainda se encontram disponíveis na base local da empresa. Este projeto foi desenvolvido utilizando dispositivos móveis da plataforma Android (*Smartphones* ou *Tablets*), que acessa o sistema de informações e a base de dados desenvolvidos através de serviços *Web (WebServices)*, através de uma rede de comunicação de dados disponível (*3G* ou *WiFi*).

Palavras-chave: Dispositivos móveis. Vistoria veicular. Serviço Web. Redes de comunicação móvel.

ABSTRACT

The project presents the integration of information coming from vehicular databases (made available from federal agencies, regarding legal situation of the vehicles, and which are paid by consultation) with a local database, to be installed at the vehicle inspection company, that are filled with information about the vehicle in the moment of the inspection. The registration of the data related to a particular vehicle undergoing inspection in a single base is a proper way to store the information. Besides that, when the same vehicle is again inspectioned, the data regarding its legal situation, provided by the federal vehicular databases, are already available for consultation and comparison. Companies using the developed information system, having access to the databases on vehicle registration, also have the advantage of reducing costs because instead of collecting data on paper forms for later typing in computer system, can allow direct access the database through the use of smartphones, and a digital format suitable for transmission. Another advantage is the economy of time using the remote transmission of data, eliminating the need to return to the headquarters to manual data entry. By using this service the company may also obtain advantages by not need to re-query the paid database (since each access to the data is paid!). To access data from a vehicle already consulted in a previous inspection is free of costs, since the data are still available in the local database of the company. This project was developed using the Android platform (as an app for Smartphones and Tablets), which accesses the information system and database developed through Web services using any communication network available (such as 3G or WiFi).

Key-words: Device Mobile. Survey Vehicle. Web Service. Mobile communication network.

LISTA DE SIGLAS

BIN	Base de Índice Nacional
RPC	*Remote Procedure Call*
RMI	*Remote Method Invocation*
JVM	*Java Virtual Machine*
HTTP	*Hypertext Transfer Protocol*
XML	*eXtensible Markup Language*
SOAP	*Simple Object Access Protocol*
REST	*Representational State Transfer*
WSDL	*WebService Definition Language*
UDDI	*Universal Discovery Description Integration*
JAX-WS	*Java API for XML WebServices*
SDK	*Software Development Kit*
SQL	*Structuted Query Language*
UML	*Unified Modeling Language*
WEB	*World Wide Web*

SUMÁRIO

1. INTRODUÇÃO

No Brasil o mercado de carros usados e seminovos é bastante amplo, ao contrário de alguns países onde a vida útil em média dura cerca de 10 anos, sendo então feita a reciclagem dos mesmos.

No mercado brasileiro o veículo sinistrado ou apreendido é geralmente disponibilizado para leilão, mesmo apresentando estrutura comprometida onde negociantes usam deste meio para recuperá-lo e vendê-lo ao preço da tabela Fipe (Fundação Instituto de Pesquisas Econômicas).

Para oferecer garantia de procedência e integridade até mesmo de carros "supostamente" zero quilômetro podem ser efetuados vários tipos de verificações que visam oferecer o máximo de dados sobre o veículo, e são efetuadas através de consultas a bases mantidas por órgãos governamentais (DENATRAN, 2013), sendo as principais:

BIN - Base de Índice Nacional que contém dados de fabricação do veículo e registro de alterações.

Leilão - Contém registro da situação do veículo quando leiloados, muitos inclusive com fotos.

Sinistros - Contém informações sobre veículos segurados que sofreram sinistros com indenização integral.

No comércio de veículos existem empresas que realizam vistoria para Seguradora, Transferência e Prévia ou Cautelar (SINALIZA.NET, 2013). A vistoria veicular prévia será o objeto de estudo apresentado no trabalho, onde é necessário acesso as bases BIN e Leilão e verificação de originalidade de itens específicos do veículo assim como uma descrição da estrutura do mesmo.

A não realização do procedimento de vistoria apresenta risco de adulteração na numeração do chassi, motor ou ponto de identificação. Existe ainda risco de estrutura comprometida. As irregularidades descritas impedem que carros sejam segurados, financiados, etc. Veículos que apresentam chassis enferrujados impossibilitando a identificação da numeração serão remarcados e atualizados na

base BIN, tais veículos são vendidos com custo até 30% abaixo do praticado no mercado. Existem veículos que, após serem dados como perdidos em sinistros ou provenientes de roubos, são recolocados no mercado com seu chassi remarcado. Um dos objetivos da vistoria veicular é detectar este tipo de fraude (OLHO VIVO VISTORIAS AUTOMOTIVAS, 2013).

As empresas que realizam serviços de vistoria veicular prévia fazem um exame físico do chassi, motor, câmbio, etiquetas, nº dos vidros, etc. Também verificam a estrutura para garantir que o veículo não sofreu reparos de grande monta. Geralmente os dados da vistoria são armazenados em um formulário de papel preenchido pelo vistoriador no momento da análise do veículo. Os dados coletados são verificados quanto à sua originalidade quando comparados com as informações da Base de Índice Nacional (BIN). Empresas de vistoria veicular acessam as bases BIN e Sinistro e disponibilizam diversas informações, sendo uma ferramenta importante nas transações com veículos. Estas empresas acessam também a base Leilão, obtendo dados e fotos, quando disponíveis, sobre a situação do veículo quando foi leiloado. Quanto mais bases consultadas maior será o valor cobrado pelas empresas fornecedoras de dados (SINALIZA.NET, 2013).

Com o intuito de melhorar o processo de vistoria será implementado um sistema para dispositivos móveis desenvolvido na plataforma *Android* denominado *VistoCar.* O sistema *VistoCar* automatiza processos da vistoria prévia, reduz custos e tempo, além de replicar e integrar dados. A elaboração do aplicativo será descrita nas seções seguintes.

1.1. SOLUÇÃO

A solução do problema seria automatizar serviço de vistoria veicular preenchendo formulário eletrônico e enviando para uma base de dados remotamente. Replicar as informações provindas da Base de Índice Nacional (BIN), que contém informações da fabricação do veículo e os dados da base Leilão para uma base única de vistoria. Integrá-los com as informações descritas pelo perito. Gerar relatório de perícia veicular dos principais itens, inclusive com informações da originalidade do motor, chassi e foto do veículo.

1.2. OBJETIVO GERAL

O objetivo do projeto é desenvolver na plataforma *Android* uma aplicação para dispositivos móveis que possibilite a automatização do processo de coleta de dados de vistoria.

1.3. OBJETIVO ESPECÍFICO

Os objetivos específicos consistem em:

- Desenvolver um sistema de informações acessível por *WebServices*;
- Criar uma aplicação *Android* para dispositivos móveis que possibilite envio e consulta de dados através de rede de comunicação *3G* e *WiFi*;
- Solucionar a falta de uma aplicação móvel para vistoria veicular preventiva;
- Replicar dados provindos da base de dado privada para base local, tendo como vantagem secundária evitar consultas repetidas às bases privadas que são pagas;
- Integrar dados provindos da base de dado privada com os coletados pelo perito e armazená-los na base local;
- Desenvolver um sistema que auxilie na redução de custos operacionais e otimização de tempo.
- Gerar relatório de serviços de vistoria veicular.

1.4. JUSTIFICATIVA

As informações veiculares provindas de bases de dados de órgãos responsáveis por divulgá-las são pagas por consulta[1]. Com o aplicativo de vistoria veicular alguns destes dados são armazenados em outra base e consultados em novas vistorias do mesmo veículo.

A análise técnica feita pelo perito descritas no formulário eletrônico, e as informações provindas de outras bases de dados compõem relatório de perícia veicular que agrega valor na compra e venda, além da satisfação do cliente.

Automatização e gerenciamento serviços de vistoria veicular em dispositivo móvel como *smartphone*, que integra várias tecnologias, que facilita o envio e consulta de dados através de rede de comunicação móvel como *3G* e *WiFi*, possibilita vantagens de custos na entrada de dados, em vez de escrever em papel, será num formato digital próprio para transmissão. Outra vantagem é a otimização do tempo de envio e recebimento dos dados remotamente, dispensando o deslocamento para outros locais.

1.5. ORGANIZAÇÃO DO TRABALHO

O capítulo 1 contém a descrição do cenário, os problemas envolvidos, os objetivos, a justificativa do porquê estudar o tema.

O Capítulo 2 apresenta descrição de como empresas com características semelhantes realizam suas atividades. Descreve as tecnologias empregadas no desenvolvimento do trabalho. Entre elas podemos citar: RPC, *WebServices*, SOAP, WSDL, UDDI, JAX-WS, *Android* e *PostgreSQL*.

O capítulo 3 mostra o desenvolvimento do trabalho. São apresentados os componentes do sistema, a modelagem e suas funcionalidades.

O capítulo 4 contém as conclusões e considerações finais sobre o trabalho, além de possíveis trabalhos futuros.

Por fim, são apresentadas todas as referências consultadas para a realização deste trabalho.

---

[1]CONSULTAAUTO. **Consulta de Veículos**. Disponível em http://www.consultaauto.com.br/consulta-veiculo. Acesso em: Agosto/2013.

2. REFERENCIAL BIBLIOGRÁFICO

Uma empresa disponibiliza uma base de dados através da *Web* para consulta sobre dados veiculares e outra busca estes dados para comparação com os coletados manualmente através de um exame físico em partes específicas do veículo.

A elaboração do protótipo de um sistema de vistoria veicular contempla tecnologias descritas nesta seção a partir do item 2.2, Tecnologias Empregadas, e possibilita acesso remoto e armazenamento de dados através de uma aplicação desenvolvida em *Android*.

2.1. SOLUÇÃO EXISTENTE

Na *Web* são disponibilizados dados de identificação de peças agregadas no veículo quando fabricado, além de apresentar, geralmente por foto, a situação do automóvel no pátio do leiloeiro. São dados de grande importância no processo de vistoria pois possibilitam identificação de carros remontados, partes pertencentes a outro, estes são geralmente de fruto de roubo. Também auxiliam na identificação de avarias que podem comprometer a estrutura veicular. Processos de vistoria foram automatizados pela aplicação móvel com intuito de redução de custo operacional e no tempo de atendimento ao cliente.

2.1.1. FORNECEDOR DE DADOS

A empresa, fornecedora de dados, que oferece um sistema de consulta através da *web* é uma importante ferramenta para transação com veículo. Para acesso ao sistema é necessário contrato jurídico entre as partes, sendo que a empresa solicitante receberá login e senha para autenticação.

O acesso ao sistema é por *web* site e para realização de consultas na base é necessário que o usuário esteja logado.

Alguns serviços de consultas oferecidos pela empresa: Base de Índice Nacional, Leilão, Perda total, Agregados, entre outros. Cada base consultada tem um valor a ser pago. No final é gerado um relatório de vistoria do veículo com os dados encontrados nas bases consultadas. Este relatório ficará disponível para o cliente por um determinado tempo.

A tabela abaixo representa os valores cobrados pelas consultas feitas na base de dado.

Tipo de consulta	**Preço**
Base de Índice Nacional (BIN)	R$ 8,00
Base Leilão	R$ 7,00
Base Perda Total	R$ 2,80
Agregados	R$ 3,80

BIN - A Base de Índice Nacional oferece informações em tempo real sobre o cadastro do veículo. Toda fonte de informações desta consulta vem direto do Sistema Nacional de Veículos Automotores (Renavan).

Leilão - Apresenta o histórico de ocorrência do veículo com dados e fotos, quando disponíveis. A fonte de consulta vem da base Detran, Ciretran. Todos os veículos leiloados são registrados nesta base.

Perda Total - Apresenta descrição da informação de indenização integral quando o veículo tem grande avaria que compromete sua estrutura. A fonte de informação é a base das Seguradoras.

Agregados - Oferece a descrição de originalidade do motor, chassi e câmbio. É uma consulta que verifica se o veículo foi clonado. Este tipo de consulta solicita entrar com os dados da numeração do motor, chassi, câmbio e eixo traseiro.

2.1.2. CONSUMIDOR DE DADOS

A empresa *Consultacar Vistoria Veiculares* é especializada em perícia identificadora de adulterações veiculares com foco na gravação de chassi, motor, câmbio e agregados, de forma a atender o público comprador ou vendedor de veículo usado.

A *Consultacar* faz um exame minucioso do veículo e alerta o comprador ou proprietário quanto as possíveis adulterações na numeração do chassi, do motor, do câmbio ou de qualquer outro elemento identificador do veículo, bem como na existência de danos de grande monta.

A empresa coleta os dados manualmente e os descrevem num formulário de papel, depois utiliza o sistema de consulta para comparação dos dados e geração de relatório de vistoria.

Caso a empresa *Consultacar* necessite realizar uma nova vistoria do mesmo veículo, terá que acessar o sistema da empresa fornecedora de dados*,* realizar uma outra consulta e pagar novamente pelo serviço prestado.

2.1.3. INOVAÇÃO APRESENTADA

O protótipo da aplicação móvel *VistoCar* apresentado neste trabalho é baseado nos serviços descritos pela empresa fornecedora de dados e pela *Consultacar.* Tem como finalidade desenvolver um aplicativo móvel para *smartphone,* automatizar serviço de vistoria em um formulário digital próprio para transmissão, criar base local, reduzir custos, integrar e replicar dados. Os serviços descritos evitam consultas já realizadas previamente, deslocamento para sede e redigitação dos dados.

2.2. TECNOLOGIAS EMPREGADAS

Esta seção apresenta um referencial teórico sobre as tecnologias abordadas na elaboração do protótipo para o desenvolvimento do sistema proposto.

2.2.1. RPC

A chamada remota a procedimento *(Remote Procedure Call – RPC)* é uma tecnologia de comunicação entre processos que permite um programa de computador chamar um procedimento em outro espaço de endereçamento (geralmente em outro computador, conectado por uma rede). Atualmente esse conceito é bastante utilizado em Sistemas Distribuídos, cada vez mais as aplicações são executadas de forma distribuída e com *RPC* é possível fazer a integração de várias aplicações distribuídas na rede (RIBEIRO, 2011).

Existem várias tecnologias que implementam o conceito de *RPC*, entre elas podemos citar:

- *Corba (Common Object Request Broker Architecture)*
- *Java RMI (Remote Method Invocation)*
- *WebServices*
- *EJB (Enterprise Javabeans)*

2.2.2. WEBSERVICES

WebServices é a solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes, possibilitando a independência de plataforma e de linguagem de programação. A tecnologia possibilita que uma aplicação possa invocar outra para efetuar tarefas, e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis. A padronização foi possível pelos componentes que permitem às aplicações enviar e receber dados através da linguagem universal, formato XML. O *WebServices* disponibiliza serviços na Web, podendo ser acessado por outras aplicações que usam, por exemplo, o protocolo SOAP.

Um exemplo de integração é um sistema desenvolvido em Java e executando em um servidor Linux pode acessar com transparência, um serviço desenvolvido na plataforma *.Net* sendo executado em um servidor de aplicações da empresa Microsoft (RECKZIEGEL, 2013).

O modelo dos serviços *Web* destaca dois elementos importantes: *papéis* e *operações*. Papéis são os diferentes tipos de entidades; as operações representam as funções executadas por essas entidades, para que o serviço *Web* possa funcionar (HENDRICKS, et al, 2002).

O Diagrama mostra o modelo de serviço *Web*.

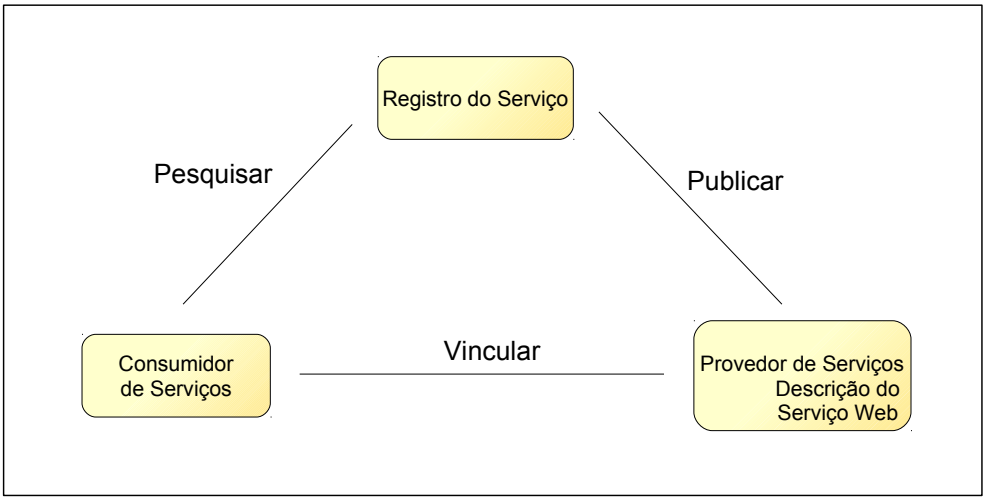


Figura 1: Diagrama representa modelo de um *WebServices*.
Fonte: Profissional Java *WebServices* (2002).

Os papéis do *WebServices*	
Provedor de serviços	É a entidade que cria o *WebServices*. Ela disponibiliza o serviço para que alguém possa utilizá-lo. Mas, para que isto ocorra, é preciso descrever o serviço *Web* em um formato padrão, que seja compreensível para qualquer um utilizá-lo. Publica os detalhes sobre seu *WebServices* em um registro central.
Consumidor de serviços	Qualquer um que utilize um *WebServices* criado por um provedor é chamado de consumidor de serviço. Este conhece a funcionalidade do serviço *Web*, a partir da descrição disponibilizada pelo provedor, recuperando os seus detalhes através de uma pesquisa sobre o registro publicado. Através da pesquisa é obtido o mecanismo para ligação com o *WebServices*.

Registro dos serviços	Um registro de serviço é a localização central onde o provedor pode relacionar seus *WebServices*, e um consumidor pode pesquisá-lo. Ele contém informações como detalhes de uma empresa, quais os serviços que ela fornece e a descrição técnica de cada um deles.

Comunicação entre as Operações
As operações *Pesquisa*, *Vínculo* e *Publicação* são fundamentais para o funcionamento do serviço *Web.* Elas precisam obter uma comunicação entre as aplicações independente de plataforma e linguagem. Para isto fazem uso de quatro camadas que empacotam a requisição e a resposta entre um servidor e um cliente. As camadas utilizadas são: *XML (Extensible Markup Language)* *SOAP (Simples Object Access Protocol)* *WSDL (Web Services Definition Language)* *UDDI (Universal Discovery Description Integration)*

2.2.2.1. XML

A Linguagem de Marcação Extensível *(Extensible Markup Language – XML)* foi criada em 1990 pelo *World Wide Web Consortium (W3C)* para ser uma linguagem de marcação que combinasse a flexibilidade da linguagem *SGML* com a simplicidade da *HTML*. Tem como propósito principal auxiliar o sistema no compartilhamento de informação através da *internet*. Tem como característica criar uma infraestrutura única para diversas linguagens. *XML* provê uma representação estruturada dos dados que mostra ser amplamente implementável e fácil de ser desenvolvida (MEDEIROS, 2013).

Os principais benefícios em utilizar *XML* são:

- Buscas mais eficientes;
- Integração de dados de fontes diferentes;
- Desenvolvimento de aplicações *Web* mais flexíveis;
- Distribuição dos dados via rede de forma mais comprimida e escalável;
- Padrão aberto.

2.2.2.2. SOAP

O protocolo de acesso simples a objetos *(Simple Object Access Protocol – SOAP)* é usado para troca de mensagens entre aplicações, independente de plataforma e de linguagem de programação. Ele baseia em *XML* para seu formato de mensagem, em *RPC* para negociação e no *HTTP* (Protocolo de Transferência de Hipertexto*)* para transporte de mensagens, embora possa usar o SMTP, FTP ou outros protocolos para transporte. Por utilizar *XML* e *HTTP*, a comunicação entre sistemas é facilitada (HENDRICKS, et al, 2002).

O *SOAP* baseia-se numa invocação remota de um método e para tal necessita especificar o endereço do componente, o nome do método e os argumentos para esse método. Estes dados são formatados em *XML* com determinadas regras e enviados normalmente por *HTTP* para o componente.

Uma mensagem *SOAP* é um documento *XML* que contém os elementos que definem a mensagem:

- *Envelope:* identifica o documento *XML* como uma mensagem *SOAP*.
- *Header:* informações sobre o cabeçalho do documento.
- *Body:* é o corpo do documento contendo informações de chamada e resposta. É o elemento que contém as informações principais da mensagem.
- *Fault:* contém informações de erro. Quando ocorre um erro no *WebServices,* as informações e detalhes do erro são enviadas nesse elemento.

2.2.2.3. WSDL

A linguagem de definição de serviços *Web (Web Services Definition Language – WSDL)* é baseada em *XML* e utilizada para descrever *WebServices,* funcionando como um contrato do serviço. Trata-se de um documento escrito em *XML* que além de descrever o serviço, especifica como acessá-lo e quais as operações ou métodos disponíveis (HENDRICKS, et al, 2002).

Um documento *WSDL* descreve um serviço *Web* como uma coleção de extremidades ou portas operando independentemente. Apresenta um serviço *Web* em duas partes:

- A primeira representa uma definição abstrata independente do protocolo de transporte;
- A segunda representa uma descrição de ligação específica para o transporte na rede.

Representação da descrição de um serviço *Web* de acordo com a *WSDL*.

Definição abstrata independente do protocolo de transporte		
Definição de Interface de Serviços Web	*Tipo de porta*	Descrição das operações e das mensagens associadas. Dentro de cada operação são especificadas as entradas e saídas esperadas.
	Mensagens	Contém uma definição dos dados a serem transmitidos. A descrição de parâmetros de entrada e saída e valores de retorno.
	Tipos	Define tipo de dado presente na mensagem.
	Ligações	Descreve o protocolo que é utilizado para acessar um *Tipo de porta*, bem como os formatos de dados para as mensagens que são definidas por um elemento *Tipo de porta,* específico.
Descrição de ligação específica para o transporte na rede		
Implementação de Serviços da Web	*Serviço*	Contém o nome do serviço da *Web* e uma lista de portas.
	Portas	Contêm o local do serviço da *Web* e a ligação usada para acesso do serviço.

2.2.2.4. UDDI

O protocolo (*Universal Description, Discovery and Integration – UDDI)* é uma especificação técnica que tem como objetivo descrever, descobrir e integrar *WebServices* (HENDRICKS, et al, 2002).

O *UDDI* permite que os provedores publiquem detalhes sobre suas empresas. A descrição (*Description)* são os serviços fornecidos em um registro central. A descoberta *(Discovery)* é a parte em que os consumidores localizam provedores e detalhes sobre seus serviços *Web.*

No momento que é construído um *WebServices*, necessita-se que os serviços sejam acessados em algum lugar da *Internet* por um cliente. Uma das maneiras é fazer com que a aplicação cliente conheça o localizador padrão de recursos (*Uniform Resource Identifier – URI*) do serviço.

Comparando *UDDI* com uma lista telefônica de maneira que às páginas:

- *Brancas:* possuem informações sobre nomes, endereços, números de telefone, além de outras informações sobre os fornecedores do serviço.
- *Amarelas:* contêm listagens comerciais baseadas nos tipos desses negócios, de maneira organizada por categoria específica ou regiões demográficas.
- *Verdes:* são usadas para indicar os serviços oferecidos por cada negócio, incluindo todas as informações técnicas envolvidas na interação com o serviço.

Uma implementação de *UDDI* corresponde a um *WebServices registry*, que provê um mecanismo para busca e publicação de serviços *Web*. Um *UDDI registry* contém informações categorizadas sobre os serviços e as funcionalidades que eles oferecem, e permitem a associação desses serviços com suas informações técnicas, geralmente definidas usando *WSDL*. O arquivo de *WSDL* descreve as funcionalidades do *WebServices*, a forma de comunicação e sua localização.

2.2.3. JAX-WS

Java para Serviços *Web XML (Java API for XML WebServices – JAX-WS)* é uma API Java para criação de *WebServices* que utiliza *XML/SOAP* como forma de comunicação e faz parte da plataforma *JEE*. Utilizando *JAX-WS* é possível transformar uma classe com diversas operações em um *WebServices* e expô-la na rede para aplicações consumidoras. A *JAX-WS* oferece anotações que simplifica o

desenvolvimento de serviços web, como *@WebService* e *@WebMethod* para definição dos *WebServices* e operações.

Exemplo de uso de anotações para a definição de um *WebServices* com JAX-WS:

```
package endpoint;
import javax.jws.WebService;
import javax.jws.WebMethod;
 @WebService(
     name="Calculator",
     serviceName="CalculatorService",
     targetNamespace="http://techtip.com/jaxws/sample"
 )
   public class Calculator {
         public Calculator() {}

         @WebMethod(operationName="add", action="urn:Add")
               public int add(int i, int j) {
                 int k = i +j ;
                 System.out.println(i + "+" + j +" = " + k);
               return k;
         }
 }
```

Anotação *@WebService*

Propriedade / valor	**Descrição**
name="Calculator"	O valor da propriedade identifica o *(WSDL) portType*.
ServiceName= "CalculatorService"	É um serviço *WSDL*.
TargetNamespace	Especifica o *namespace XML* utilizado para o *WSDL*.

Anotação *@WebMethod*

Propriedade / valor	**Descrição**
operationName="add"	O valor da propriedade identifica uma operação *WSDL* (adiciona).
action="urn:Add"	Ação do valor de propriedade especifica um *namespace XML* para o *WSDL* dos elementos gerados a partir da operação do *webservice*.

A anotação *@WebService* define a classe como sendo um *WebService* e o *@WebMethod* indica que ele será exposto na web, podendo ser invocado por qualquer módulo cliente.

A execução do *WebService* gera *URL (Universal Resource Locator)* com o *WSDL*, ao clicar no *link* mostra as características que o *WebServices* tem descrito no *WSDL* (endereços, métodos disponíveis, parâmetros dentre outros em formato *XML*). Este mecanismo serve para que os consumidores do serviço saibam o que *WebService* possui.

Com o arquivo de especificação *WSDL* é possível gerar um *Stub* que possa simular o comportamento do serviço original e utilizar exatamente a mesma interface. O *Stub* é a aplicação que encapsula a invocação do *WebServices* remoto. Em geral são utilizadas ferramentas que geram o código do *Stub* automaticamente, a partir do *WSDL*. Possui todo o código necessário para abrir *sockets HTTP*, enviar a mensagem *SOAP*, obter a resposta e transformar em objetos. Entre as responsabilidades do *Stub*, estão:

- *Marshalling*: transforma o objeto de entrada do cliente em *XML* para ser enviado pela rede.
- *Transporte:* envia a requisição e obtém a resposta utilizando mensagens *SOAP*.
- *Unmarshalling*: transforma o *XML* de retorno em objeto e devolve para o cliente.

2.2.4. ANDROID

O *Android* é um sistema operacional desenvolvido para dispositivos móveis. Possui núcleo baseado no sistema operacional *Linux*, utiliza a linguagem Java para desenvolvimento das aplicações. Foi criado com a intenção de padronização para um ambiente de software de dispositivos móveis de código aberto e de fácil migração para os fabricantes, justamente para atender expectativas e tendência do mercado (LECHETA, 2010).

2.2.4.1. Plataforma Android

A plataforma *Android* foi projetado e desenvolvida pelo *Google* e posteriormente pelo grupo *Open Handset Alliance (OHA)* que é formado por empresas (Motorola, LG, Samsung, Sony Ericson e muitas outras) líderes do mercado de tecnologia de dispositivos móveis, provedoras de serviços móveis, fabricantes, etc.

Esta plataforma foi desenvolvida utilizando o sistema operacional *Linux*. Sendo assim, todas as características intrínsecas deste sistema foram incorporadas, bem como sistema de arquivos, o *kernel*, os servidores de terminais *(X server).*

Aplicações escritas em Java são compiladas em *bytecodes Dalvik,* formato *.dex (Dalvik Executable),* e executadas usando a máquina virtual *Dalvik*, que é especializada no desenvolvimento no uso em dispositivos móveis, o que permite que programas sejam distribuídos em formato binário *(bytecode)* e possam ser executados em qualquer dispositivo *Android*, independentemente do processador utilizado. Apesar das aplicações *Android* serem escritas na linguagem Java, não executa *bytecode JVM*.

Algumas características suportadas pela plataforma:

- *Armazenamento: SQLite* para armazenamento de dados em formato de estruturas relacionais;
- *Mensagens:* SMS, MMS são formas disponíveis de envio de mensagens;
- *Máquina Virtual Dalvik:* executa códigos compilados em *bytecodes Dalvik* que é otimizado para dispositivos móveis;
- *Hardware: Bluetooth, EDGE, 3G, e WiFi* (dependente de hardware), Câmera, GPS, bússola, e acelerômetro (dependente de hardware);
- *Multimídia:* áudio (formatos MP3, AAC, AMR),
  vídeo (MPEG4 e H.264),
  imagens (formatos JPG, PNG, GIF);
- *Navegador:* navegador web integrado baseado no *framework* de código aberto *Webkit*;

- *Layouts:* gráficos otimizados por meio de uma biblioteca de gráficos 2D; e gráficos 3D baseados na especificação *OpenGL* ES;
- Telefonia GSM (dependente de hardware);
- *Ambiente de Desenvolvimento (SDK):* incluindo um emulador de dispositivo, ferramentas para depuração, analisador de memória e performance; e um *plugin* para a IDE Eclipse.

2.2.4.2. Arquitetura Android

A seguir será explicado a arquitetura, que é dividida em várias camadas, e os componentes que juntos compõem a arquitetura da plataforma *Android*.

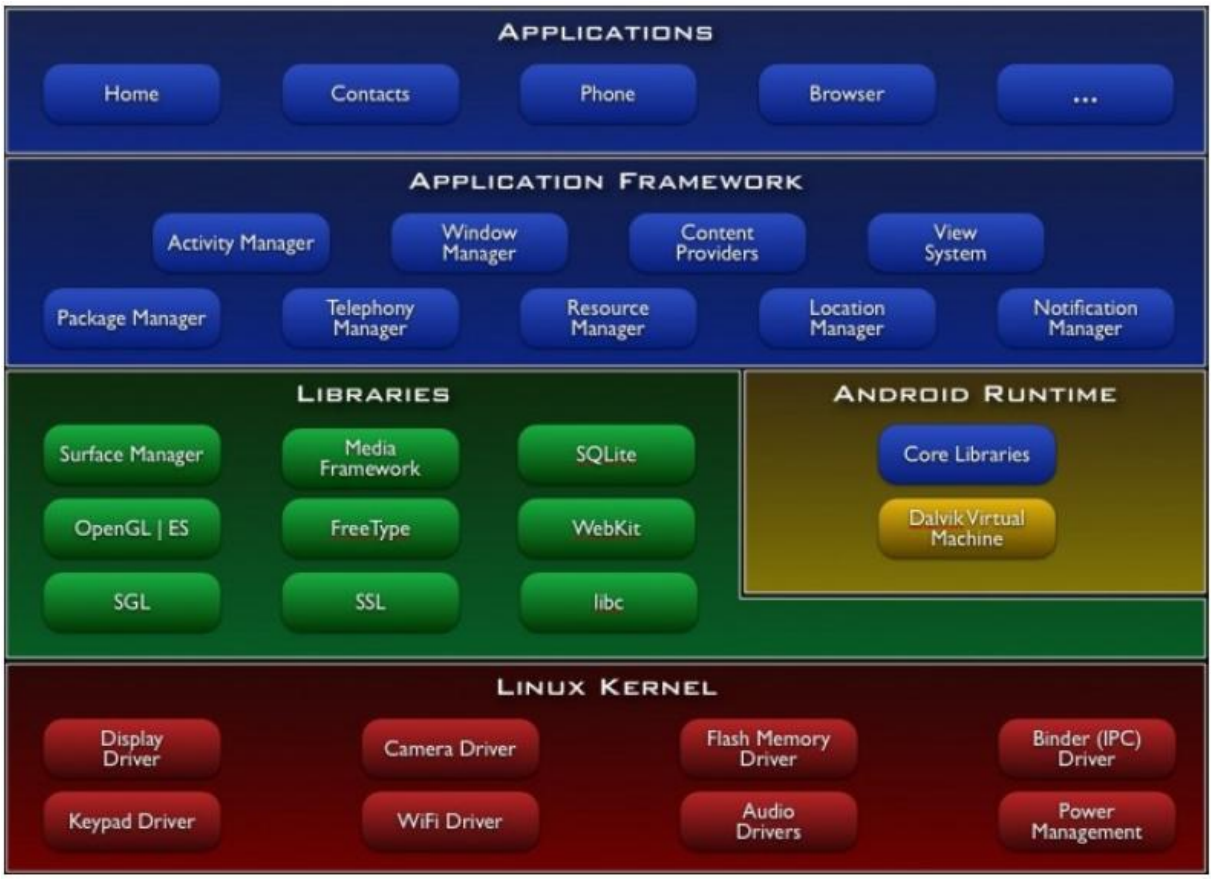


Figura 2: Arquitetura da plataforma Android.
Fonte: Conhecendo o Android.

Applications: Nesta camada está localizada uma lista de aplicações padrões que incluem um cliente de e-mail, programa de SMS, calendário, mapas, navegador, gerenciador de contatos, calculadora e outros que serão desenvolvidos pela comunidade, sendo todas estas aplicações escritas na linguagem Java.

Framework: A camada *Aplication Framework* disponibiliza componentes que possibilitam a criação de novas aplicações, enfatizando a reutilização de código. Podemos citar entre elas: *Activity Manager* (ciclo de vida da aplicação), *Content Providers* (provedores de conteúdo para compartilhar dados: Contatos), *View System* (componentes gráficos: caixas de textos botões, etc), *Location Manager* (localização por GPS ou Rede), *Notification Manager* (exibe alertas na barra de *status*), etc.

Libraries: Apresenta as bibliotecas nativas escritas em C/C++ que são compostas por uma coleção de bibliotecas como: *Surface Manager* (gerencia o acesso ao *display* do dispositivo), *Media Framework* (suportam execução e gravação em áudio e vídeo, como exibição de imagens), *SQLite* (banco de dados relacional), *WebKit* (renderização para exibir páginas Web), *OpenGl* | ES (Renderização 3D), etc.

Runtime: Esta camada da condição para que as aplicações baseadas na plataforma sejam executadas. Um dos componentes desta camada é a *Core Libraries* que disponibiliza uma API Java utilizada para programação. Já o outro componente é a máquina virtual denominada *Dalvik* que disponibiliza ambiente necessário para execução de uma aplicação *Android*.

Kernel: Nela está localizado o *kernel Linux*, que fornece serviços do núcleo do sistema como segurança, gerenciamento de memória, processos, pilhas de redes, etc.

2.2.4.3. Android SDK

É o software utilizado para desenvolver aplicações no *Android*, tem emulador para simulação do celular, ferramentas utilitárias, códigos de exemplo, tutoriais e uma API completa para a linguagem Java. O SDK é suportado nos sistemas operacionais *Windows*, *Mac* e *Linux*. Abaixo citamos algumas classes utilizadas para a elaboração do aplicativo móvel:

– *Activity:*

Representa a tela da aplicação, é responsável por tratar os eventos gerados nesta tela. Para haver interação com a interface de usuário uma classe deve herdar de uma *activity*. No entanto para desenhar na tela esta utiliza a classe *View* que por sua vez se encarrega de desenhar os componentes visuais, como campos de textos, botões e imagens.

– *View:*

É a classe mãe de todos os componentes visuais do *Android*, é responsável pela criação de qualquer componente gráfico, ela é a base para os componentes visuais.

– *Spinner:*

É utilizada para criar um combo na tela, é responsável para exibir uma lista com opções que poderá ser escolhida pelo usuário. Este componente geralmente apresenta uma descrição e quando é clicado exibe outras opções de escolha.

– *ListView:*

É um *widget* que permite apresentar uma lista "rolável" de itens, onde cada um pode ser composto de vários componentes. Os itens da lista podem ser selecionados. É relativamente simples preencher um *ListView* com dados provenientes de uma matriz ou base de dados, através de um *Adapter*.

– *Adapter:*

Funciona como uma ponte entre um *AdapterView* e os dados para esta visão. O adaptador permite o acesso aos itens é também responsável por fazer uma visão para cada item do conjunto de dados.

– *Gallery:*

Possibilita organizar componente lado a lado na horizontal. É utilizado frequentemente para exibir uma galeria de imagens. Ao selecionar uma imagem ela é exibida ao centro deslocando juntamente as demais.

– *Handler:*

Utilizada para enviar ou agendar mensagens que devem ser executadas geralmente depois de um determinado intervalo de tempo. É muito importante na arquitetura porque somente por meio dela é possível atualizar a interface gráfica a partir de uma outra *thread,* diferente da *thread* principal que controla a tela.

– *iText:*

iText é a API utilizada para manipular e gerar arquivos, entre eles o PDF, é gratuito e *open source.* O formato PDF tem vantagem de manter o *layout* original, além de ser um padrão de distribuição de documento.

Para usar os recursos da API deve adicionar o arquivo *jar iText* ao projeto, criar um objeto da classe *Document* e outro objeto da classe *OutputStream* com nome do arquivo de saída. O documento criado permite inserir texto, imagem, tabela, metadados, código de barra, além de apresentar variados tipos de fontes.

Abaixo é apresentada uma classe que gera um arquivo PDF.

```
import java.io.FileOutputStream;
import java.io.OutputStream;
import com.itextpdf.text.Document;
import com.itextpdf.text.Font;
import com.itextpdf.text.Font.FontFamily;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.pdf.PdfWriter;

public class GerarPDF {
   public static void main(String[] args) throws Exception
   {

      Document doc = null;
      OutputStream os = null;
```

```
        try {
          //cria o documento tamanho A4
          doc = new Document();

          //cria a stream de saída. Nome do arquivo de saída
          os = new FileOutputStream("VistoCar.pdf");

          //associa a stream de saída ao
          PdfWriter.getInstance(doc, os);

          //abre o documento
          doc.open();

          //adiciona metadados
          doc.addTitle("PDF da VistoCar");
          doc.addSubject("Usando o iText");
          doc.addKeywords("Java, PDF, iText");
          doc.addAuthor("Ilson Sabino");
          doc.addCreator("Ilson José Sabino");

          //Alterando fonte
          Font f = new Font(FontFamily.COURIER, 20, Font.BOLD);

          //adiciona o texto ao PDF
          Paragraph p1 = new Paragraph("Criação do arquivo PDF!");
          doc.add(p1);

          Paragraph p2 = new Paragraph("Estou utilizando a classe
                         Paragraph para criar um bloco de texto na geração
                         do meu primeiro arquivo PDF.");
          doc.add(p2);

        } finally {

          if (doc != null) {
              //fechamento do documento
              doc.close();
          }

          if (os != null) {
              //fechamento da stream de saída
              os.close();
          }
        }
    }
}
```

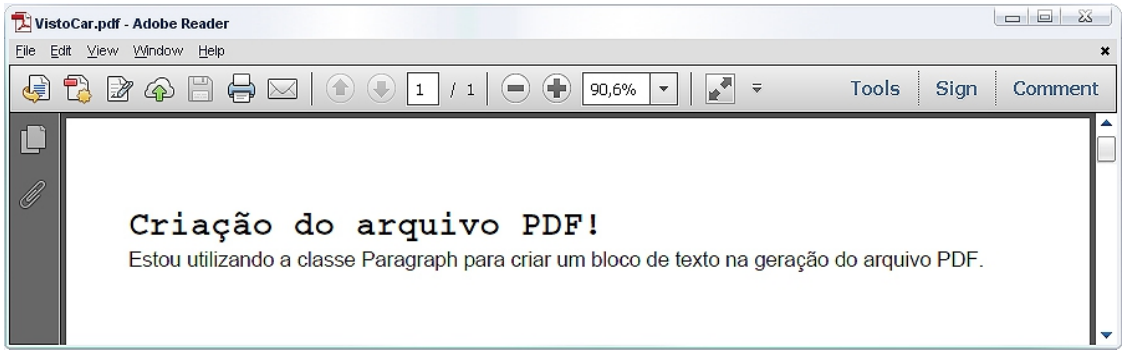


Figura 3: Arquivo VistoCar.pdf gerado com a execução dos códigos da classe *public class GerarPDF*.

A execução dos códigos da classe pública GerarPDF cria na pasta de desenvolvimento da aplicação um arquivo de formato PDF com os textos da classe *Paragraph* inseridos, conforme apresentado na figura 3. As informações da propriedade do arquivo descritas no metadado são apresentadas na figura 3a.

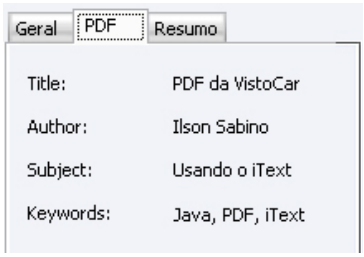


Figura 3a: Apresenta a propriedade do arquivo PDF gerado a partir do metadado.

O *iText* oferece métodos que permitem a manipulação do arquivo PDF.

Abertura do documento	`document.open();`
Formato da página	`document.setPageSize(PageSize.A3);` O tamanho *default* criado é A4
Adiciona página	`document.newPage();`
Adiciona parágrafo	`document.add(new Paragraph("Novo parágrafo adicionado"));`
Adiciona imagem	`Image figura = Image.getInstance("C:\\imagem.jpg");` `document.add(figura);`

Adiciona metadados	`document.addTitle("PDF da VistoCar");`
	`document.addSubject("Usando o iText");`
	`document.addKeywords("Java, PDF, iText");`
	`document.addAuthor("Ilson Sabino");`
	`document.addCreator("Ilson José Sabino");`

Altera fonte	`Font f = new Font(FontFamily.COURIER, 20, Font.BOLD);`

2.2.4.4. Componentes de Aplicação

Uma aplicação *Android* pode ser desenvolvida pela composição de uma série de componentes, blocos de construção, que interagem entre si, ou com a aplicação, sendo que alguns são pontos de entradas para o usuário. Uma aplicação pode não possuir todos eles, mas será construído com a combinação de alguns deles. Eles tem propósito e ciclo de vida distinto que define como o componente é criado e destruído (TOSIN, 2013).

Os componentes da aplicação são declarados no *AndroidManifest.xml* que é um arquivo *XML* obrigatório e único para cada aplicação. É nele que são feitas as configurações gerais da aplicação e dos componentes.

O *Android Core*, que é a plataforma do *Android*, proporciona a interação entre os componentes e as aplicações e torna possível a execução do código.

A figura 4 representa o *Android Core*, o *AndroidManifest.xml* e os quatro tipos de componentes de aplicação:

Figura 4: Componentes de uma aplicação Android.
Fonte: Conhecendo o Android.

Activities: Uma *Activities* geralmente representa a tela da aplicação para interação com o usuário. Cada atividade é responsável por tratar os eventos gerados nesta tela e definir qual *View* é responsável por desenhar a interface gráfica do usuário.

Services: São serviços que rodam em segundo plano realizando operações que levam muito tempo de execução. Não possuem interação direta com o usuário. Não fornecem interface com usuário e portanto não possuem *View* associada. Um bom exemplo é um *media player* tocando músicas em segundo plano enquanto o usuário pode realizar outra atividade.

Content Providers (provedores de conteúdos): É a maneira utilizada para compartilhar dados entre as aplicações que executam no dispositivo. Um bom exemplo é a aplicação de gerenciamento de contatos nativo do *Android* que fornece um *Content Provider* que possibilita consultar e modificar os dados do contato.

Broadcast Receivers: É um componente que responde a eventos gerados pelo *Android*. Podem ser eventos originados pelo sistema ou disparados por aplicações. Pelo sistema: anunciando que a tela ficou escura, a bateria está baixa, recebimento de ligação, etc. Pela aplicação: gerados por aplicativos que permitem que outras aplicações saibam que algum dado foi baixado para o dispositivo e está disponível para uso. *Brodcast Receivers* não possui interface com usuário, mas pode enviar notificação na *status bar* avisando o usuário que um evento ocorreu.

2.2.4.5. Ciclo de Vida de uma Activity

São os estados que uma atividade se encontra que podem ser: executando, temporariamente interrompida em segundo plano ou completamente destruída.

Pode-se citar, por exemplo, um usuário jogando e de repente decide navegar na *internet*, para isso, ele pausou *(pause)* o jogo. O *Android* passa a executar o navegador da *internet* e deixa em segundo plano o jogo que está temporariamente parado. O jogo pausado pode voltar a executar, de onde parou, quando o usuário achar necessário. Se o sistema operacional precisar liberar recursos e memória pode decidir encerrar o processo do jogo.

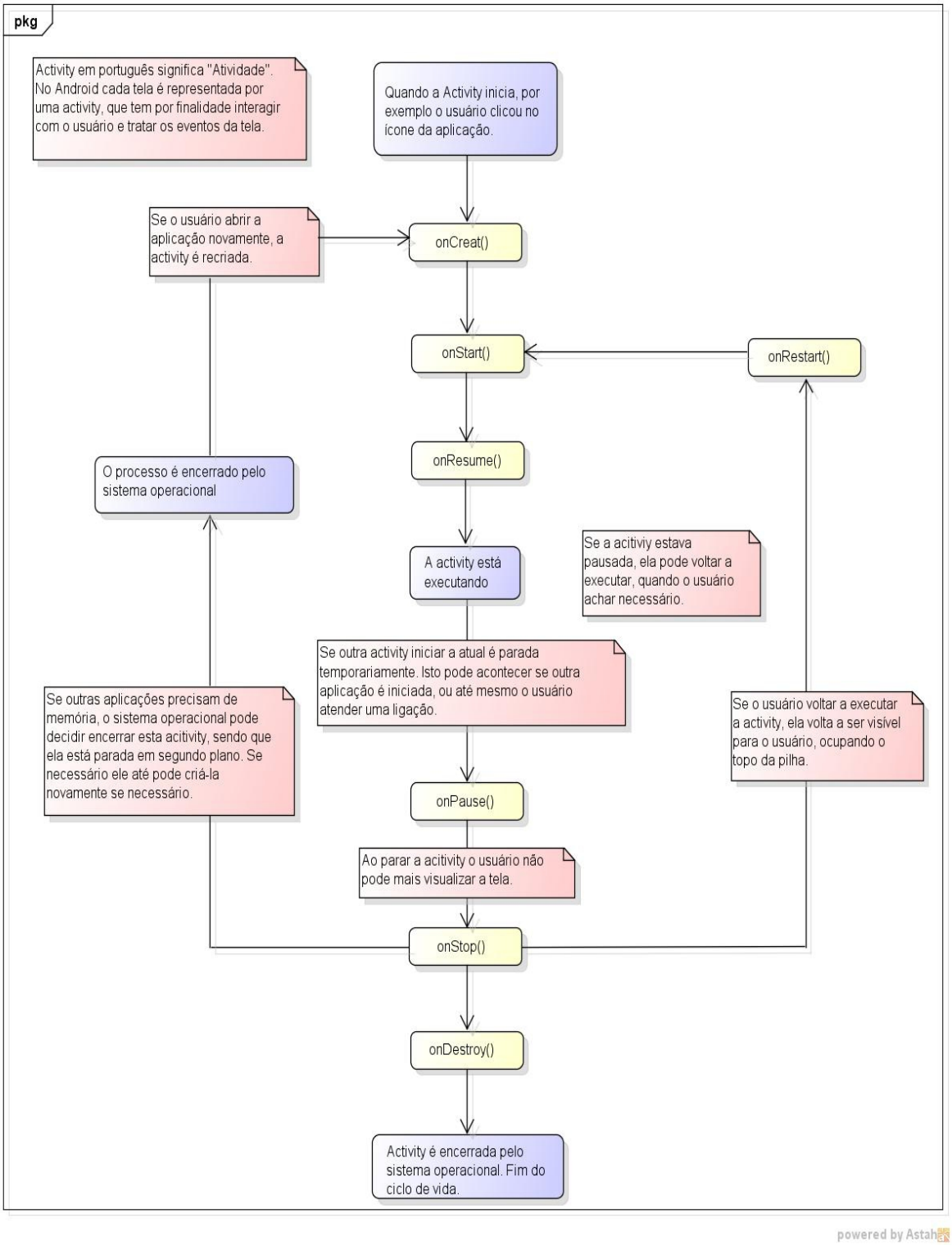


Figura 5: Ciclo de vida de uma activity Android.
Fonte reprodução: Google Android: aprenda a criar aplicações para dispositivos móveis com o Android SDK

O *Android* oferece métodos que controlam cada fase do ciclo de vida de uma *activity*. Abaixo é descrito qual método é relevante em cada ciclo:

Método	**Descrição**
onCreate()	O método é obrigatório, é chamado uma única vez, é o primeiro a ser executado em uma *activity*. Geralmente é a responsável por carregar o *layout XML.*
onStart()	É chamado na inicialização da *activity* e também quando uma *activity* que estava em segundo plano voltar a ter foco. Executado quando uma *activity* não está visível ao usuário.
onRestart()	Quando uma *activity* volta a ter foco depois de estar em segundo plano. O método *onRestart()* chama automaticamente o *onStart()*.
onResume()	Sempre invocado quando a *activity* que estava em segundo plano e volta a ter foco. Este método é chamado nas retomadas de foco.
onPause()	Salva o estado da aplicação para continuar processamento do ponto que parou quando *activity* foi interrompida temporariamente. Evento do celular quando entra em modo “dormir” para economizar energia.
onStop()	Quando a *activity* não está mais visível ao usuário, pode ocorrer quando outra *activity* é iniciada. A *activity* parada pode ser reiniciada ou destruída.
onDestroy()	É a chamada final da *activity* que não pode ser mais relançada. O método *finish()* pode ser chamado pela aplicação ou automaticamente pelo sistema para liberar recursos quando a *activity* estiver no estado *onPause()* ou *onStop()*.

2.2.5. POSTGRESQL

O *PostgreSQL* é um sistema gerenciador de banco de dados objeto-relacional de código aberto. A arquitetura ganhou reputação de confiabilidade, integridade de dados e conformidade a padrões. Roda em todos os grandes

sistemas operacionais, incluindo *Linux*, *Unix, Mac OS X, Solaris* e *Windows*. É totalmente compatível com ACID (Atomicidade, Consistência, Isolamento e Durabilidade). Tem suporte a chaves estrangeiras, junções (JOINs), visões, gatilhos e procedimentos armazenados (em múltiplas linguagens). Inclui a maior parte dos tipos de dados. Possibilita armazenamento de objetos binários, incluindo figuras, sons ou vídeos. Possui interfaces nativas de programação para *C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC*, entre outros, e uma excepcional documentação. É sensível a caixa (maiúsculas e minúsculas) e formatação. É escalável, tanto na quantidade enorme de dados que pode gerenciar, quanto no número de usuários concorrentes que pode acomodar.

Como um banco de dados de nível corporativo o *PostgreSQL* possui funcionalidades sofisticadas como o controle de concorrência, *tablespaces*, replicação assíncrona, transações agrupadas *(savepoints)*, consultas *SQL* declarativas e registrador de transações sequencial para tolerância a falhas (SOBRE, 2013).

2.2.6. ECLIPSE

É uma ferramenta para desenvolvimento interativo de software *(Integrated Development Environment – IDE)*, compatível com Java e outras linguagens, como C, C++, PHP, etc. Usa o *Standard Widget Toolkit (SWT)* para exibir elementos gráficos, acessa as bibliotecas gráficas nativas do sistema operacional. Possibilita a instalação de *plugins* para torná-lo compatível com outras tecnologias existente. É um ambiente de desenvolvimento interativo de código aberto, desenvolvido e mantido pela comunidade *open source* (OLIVEIRA, 2013).

O *Eclipse* fornece uma vasta gama de funções que abrangem todos os pacotes de desenvolvimento Java, incluindo a *Standard Edition, Mobile Edition* e *Enterprise Edition*, sendo uma ferramenta bastante adequada para desenvolvimento Java em qualquer plataforma.

2.2.7. SERVIDOR WEB TOMCAT

O *Apache Tomcat* é um servidor de aplicações Java para web que implementa as tecnologias *JavaServlets* e *JavaServer Pages*. Ele também pode comportar-se como um servidor web (HTTP) ou funcionar integrado a um servidor web dedicado.

Ele é um software livre de código aberto, nascido no Projeto *Apache Jakarta, e* é oficialmente autorizado pela Sun, tem a implementação de referência para as tecnologias *Java Servlet* e J*avaServer Pages (JSP)*. Ele cobre parte da especificação J2EE com as tecnologias relacionadas como *Realms* e segurança, *JNDI Resources* e *JDBC DataSources*, contudo, ele não implementa pacotes *Enterprise JavaBeans (EJB)*.

O Apache *Tomcat* é inteiramente escrito em Java, portanto, para ser executado em seu computador ele necessita de uma Máquina Virtual Java *(Java Virtual Machine – JVM)* instalada. A instalação do servidor é simples, porém a configuração requer conhecimento prévio sobre o assunto ou uma leitura criteriosa nos manuais ou em sites na internet.

Para escolher a versão mais adequada a sua necessidade é recomendado efetuar uma consulta à documentação e para o desenvolvimento, faz-se necessário, também, conhecimento da linguagem de programação Java (PEREIRA, 2013).

3. DESENVOLVIMENTO

3.1. MELHORIAS NAS ATIVIDADES APRESENTADAS

O trabalho da empresa fornecedora de dados, conforme descrito no item 2.1.1, tem como foco fornecer dados veiculares e procedência do veículo através da rede de comunicação *Internet.*

A empresa *Consultacar Vistoria Veiculares*, conforme descrito no item 2.1.2, tem como objetivo principal atender o cliente da melhor forma possível, com absoluta imparcialidade, independente do resultado em relação ao contratante. Busca nos veículos usados possíveis adulterações, danos de grande monta, estruturas comprometidas e remontagem de veículos. Coleta dados comparando-os com os disponibilizados pela empresa fornecedora de dados.

O trabalho proposto neste protótipo utiliza as atividades apresentadas pelas empresas fornecedoras de dados e *Consultacar Vistoria Veiculares* adicionando os seguintes serviços:

- Replicação dos dados fornecidos das bases externas para a base de vistoria da aplicação *Android*.
- Integração dos dados consultados nas bases externas com os descritos pelo perito no formulário eletrônico;
- Desenvolvimento de um aplicativo de *smartphone* com comunicação de rede sem fio, sem necessidade de deslocamento para envio de dados, otimizando o tempo.
- Criação de formulário digital para entrada de dados próprio para transmissão.
- Geração do relatório de vistoria contendo dados consultados, coletados e fotos do veículo.

3.2. DESCRIÇÃO GERAL DO PROTÓTIPO

O Perito da empresa de vistoria veicular executa o aplicativo pelo *smartphone*, entra com os dados usuário e senha e enviam para o servidor Centralizador, que verifica as informações e responde para o aplicativo. Em caso de resposta afirmativa é liberado acesso ao sistema disponibilizando um formulário com opção das telas Cliente, Vistoria e Relatório.

O Perito preenche o campo placa da tela de Vistoria e envia para consulta no servidor Centralizador, que ao encontrar o registro retorna os dados solicitados, inclusive os da BIN e Leilão. Esta tela apresenta a opção para novas consulta BIN e Leilão que acessa o servidor Terceirizado replicando os dados para o servidor da *VistoCar*. Algumas informações fornecidas pela base Terceirizada são comparadas com os dados inseridos pelo Perito para constatação de originalidade ou falsificação de itens de vistoria.

O aplicativo móvel envia os dados do formulário, via rede sem fio, utilizando a rede de comunicação *Wi-fi* ou *3G,* para a *VistoCar* onde são armazenados na base de dados do *PostgreSQL*.

Para este processo foi usada uma solução conhecida como *WebServices*.

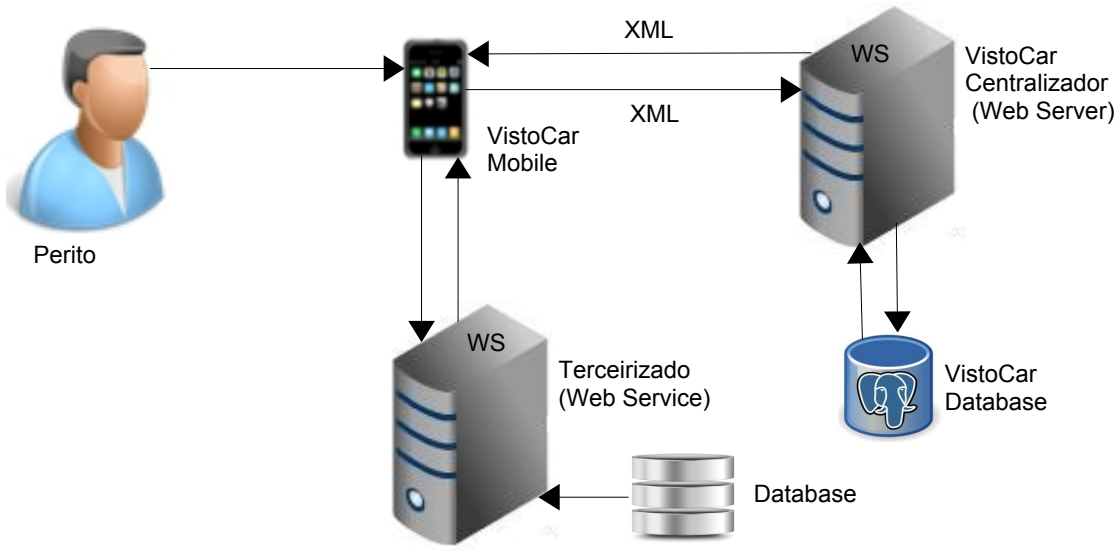


Figura 6: Arquitetura geral do protótipo.

A figura 6 ilustra uma visão geral do desenvolvimento do protótipo proposto. Ela é resultado da análise dos objetivos citados no primeiro capítulo.

Na figura apresentada temos os principais componentes da arquitetura:

Componentes	Descrição
VistoCar Mobile	*Módulo Móvel* é um *smartphone* com aplicativo desenvolvido em *Android*, utilizado pelo Perito para logar no sistema, gerar relatório, inserir e consultar dados de vistoria.
VistoCar Centralizador	*Módulo Centralizador*, *WebServices*, é um servidor com um conjunto de serviços que capta as informações provindas do módulo *VistoCar Mobile* e enviam para a base de dados.
VistoCar Database	*Módulo Databa*se faz persistência dos dados enviados pelo módulo *Centralizador*. Contém os dados dos perito, dos clientes, das vistorias, dos leilões e dos veículos.
Terceirizado	*Módulo Terceirizado*, *WebServices*, fornece dados da BIN e Leilão que são replicados para o servidor da *VistoCar Centralizador*.

3.3. REQUISITOS

Requisitos	Descrição
Efetuar login	O sistema deve oferecer autenticação do Perito através de login. Deve validar usuário e senha para permitir acesso ao módulo *VistoCar Mobile*.
Disponibilizar formulário	O sistema deve disponibilizar formulário para preencher com dados coletados e os provindos das bases de dados.
Consultar dados (Centralizador)	O sistema deve permitir consultas na base *Centralizada* e disponibilizá-la no formulário de vistoria.
Consultar dados (Terceirizado)	O sistema deve permitir consultas na base *Terceirizada*, empresa privada, e disponibilizá-la no formulário de vistoria.
Transferir formulário	O sistema deve enviar os dados do formulário eletrônico usando rede de comunicação sem fio.

Registrar nova vistoria	O sistema deve armazenar os dados enviados pelo formulário de vistoria no banco de dados do servidor centralizador.
Registrar novo cliente	O sistema deve armazenar os dados enviados pelo formulário de cliente no banco de dados do servidor centralizador.
Gerar relatório	O sistema deve emitir relatório contendo dados do veículo, perito, cliente, vistoria e fotos da situação do veículo.

3.4. DIAGRAMA DE CASOS DE USO

Um diagrama de Caso de Uso descreve as principais funcionalidades do sistema e a interação destas funcionalidades com os usuários do mesmo sistema. A figura abaixo representa um diagrama de Casos de Uso do Sistema de Vistoria Veicular – *VistoCar Mobile*.

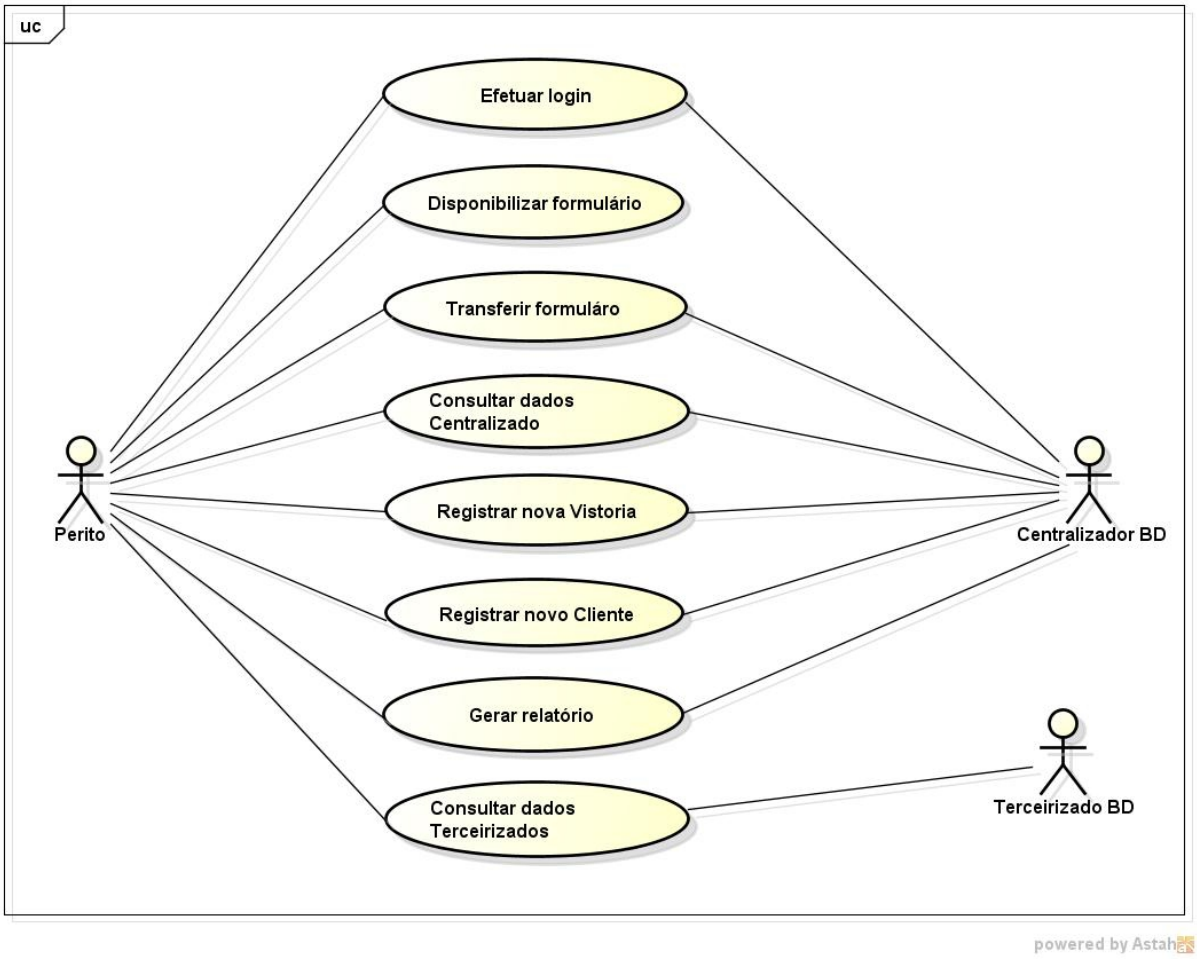


Figura 7: Casos de Uso do aplicativo móvel VistoCar Mobile.

3.5. FLUXO DE EVENTOS

Autenticar usuário	
1. Perito inicia o aplicativo móvel. 2. Aplicativo abre com a tela de login. 3. Perito preenche informações de usuário e senha e pressiona o botão “Entrar”. 4. Se autenticação for bem-sucedida. 4.1. Redireciona para tela *Atividade* que contém opção para as telas de *Cliente*, *Vistoria* e *Relatório*. 4.2. Senão inicia o passa 3.	
Pré condições	*Smartphone* deve ter aplicação *Vistocar Mobile* instalada, possuir conexão de rede e Perito deve ser registrado na base de dados do servidor Centralizador.
Pós condições	Perito deve estar autenticado no sistema.

Disponibilizar formulário	
1. Apresenta a tela *Atividade* e recebe a opção escolhida. 2. Se opção for *Vistoria*. 2.1. Abre a tela de formulário de vistoria. 2.2. Recebe os dados coletados da vistoria.	
Pré condições	Perito deve estar autenticado no sistema.

Consulta no servidor Centralizador	
1. Apresenta a tela *Atividade* e recebe a opção escolhida. 2. Se opção for *Vistoria*. 2.1. Abre a tela de formulário de vistoria. 2.2. Recebe o número da placa do veículo. 2.3. Busca os dados do veículo no banco de dados do servidor *Centralizador*. 2.4. Se encontrou dados do veículo. 2.4.1. Apresenta os dados atuais. 2.5. Recebe os dados novos.	
Pré condições	Perito deve estar autenticado no sistema. *Smartphone* deve possuir conexão de rede e banco de dados do servidor *Centralizador* disponível.

Consulta no servidor Terceirizado	
1. Apresenta a tela *Atividade* e recebe a opção escolhida. 2. Se opção for *Vistoria*. 2.1. Abre a tela de formulário de vistoria. 2.2. Recebe o número da placa do veículo. 2.3. Verifica se tem registro no servidor *Centralizador.* 2.4. Se tem registro. 2.4.1. Verifica a necessidade de novas consultas. 2.5. Se não tem registro. 2.5.1. Busca os dados do veículo no banco de dados do servidor *Terceirizado*. 2.5.2. Apresenta os dados atuais.	
Pré condições	Perito deve estar autenticado no sistema. *Smartphone* deve possuir conexão de rede e banco de dados do servidor *Terceirizado* disponível.

Transferir formulário	
1. Apresentar tela conforme descrito no fluxo de evento *Disponibilizar Formulário*. 2. Utilizar o fluxo de evento *Consultar no servidor Centralizador* a partir do passo 2.2. 3. Utilizar o fluxo de evento *Consultar no servidor Terceirizado* a partir do passo 2.2. 4. Verificar se tem alguma conexão de rede disponível. 4.1. Se tiver: 4.1.1. Enviar formulário. 4.2. Se não tiver: 4.2.1. Voltar ao passo 4.	
Pré condições	Perito deve estar autenticado no sistema. *Smartphone* deve possuir conexão de rede disponível.

Registrar nova vistoria
1. Apresentar tela conforme descrito no fluxo de evento *Transferir Formulário*. 2. Gravar os dados no banco de dados do servidor Centralizador.

3. Se gravação foi bem-sucedida. 3.1. Apresentar uma mensagem de confirmação. 4. Se gravação não foi bem-sucedida voltar ao passo 2.

Pré condições	Perito deve estar autenticado no sistema. *Smartphone* deve possuir conexão de rede e banco de dados do servidor *Centralizador* disponível.

Registrar novo cliente
1. Apresenta a tela principal e recebe a opção escolhida. 2. Se opção for C*liente*. 2.1. Abre o formulário de registro de cliente com opção para consulta. 2.2. Recebe o nome do cliente. 2.3. Verifica se tem registro no servidor *Centralizador*: 2.3.1. Se tem registro. 2.3.1.1. Apresenta os dados atuais. 2.4. Recebe os dados novos. 2.5. Gravar os dados no banco de dados do servidor Centralizador.

Pré condições	Perito deve estar autenticado no sistema.

Gerar relatório
1. Apresenta a tela principal e recebe a opção escolhida. 2. Se opção for *Relatório*. 2.1. Abre a tela de consulta. 2.2. Recebe o número da placa do veículo. 2.3. Verifica se tem registro no servidor *Centralizador*: 2.4. Se tem registro. 2.4.1. Gerar relatório de vistoria. 2.5. Se não tem registro, voltar ao passo 2.2 ou finalizar consulta.

Pré condições	Perito deve estar autenticado no sistema. *Smartphone* deve possuir conexão de rede e banco de dados do servidor *Centralizador* disponível.

3.6. DIAGRAMA DE SEQUÊNCIA

Representa a sequência de mensagens passadas entre os componentes do sistema, demonstra as interações entre os componentes do sistema.

3.6.1. AUTENTICAR USUÁRIO

O controle do sistema exibe a tela de autenticação e fica aguardando o retorno que pode ser: fechar a aplicação ou enviar dados para o SBD Cent. Se os dados forem autenticados o controle ativa a tela de menu conforme figura 8.

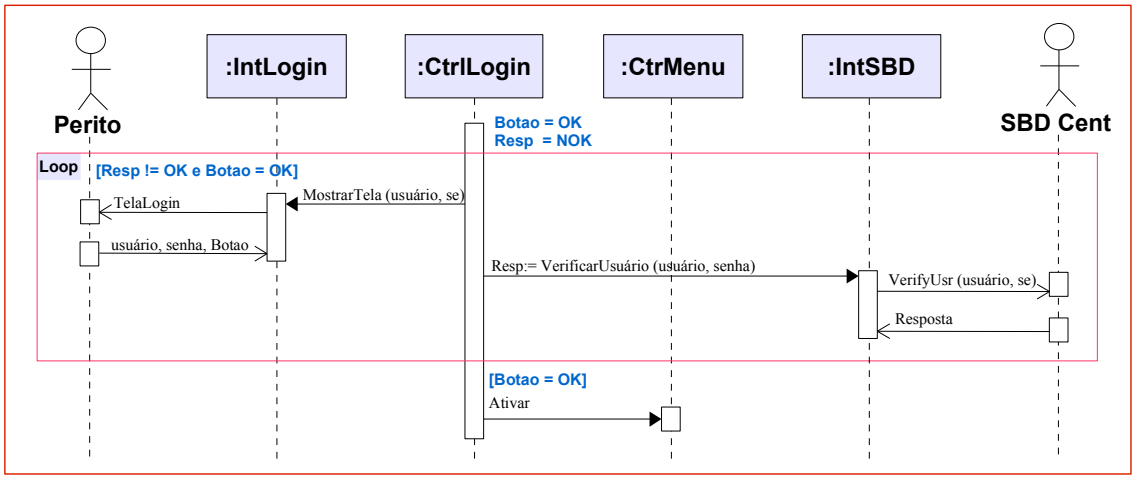


Figura 8: Diagrama de sequência autenticar usuário.

3.6.2. APRESENTAR MENU

O controle do sistema exibe uma tela de menu e fica aguardando o retorno podendo disponibilizar formulário de vistoria conforme a figura 9.

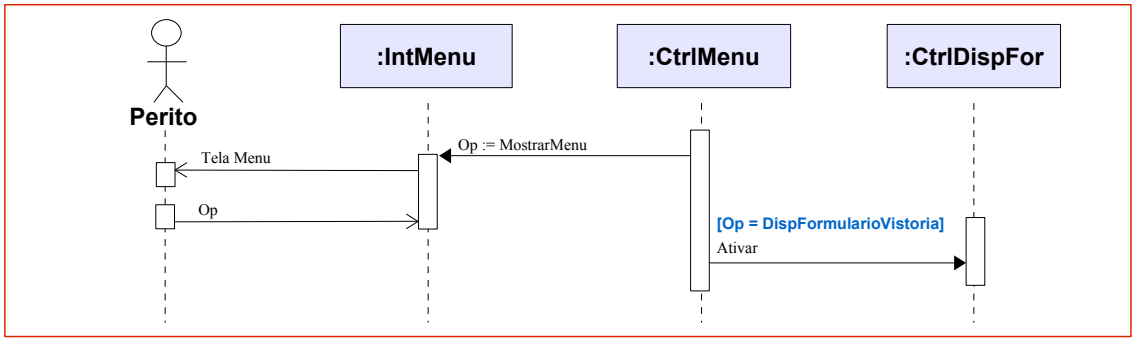


Figura 9: Diagrama de sequência apresentar menu.

3.6.3. VISTORIA VEICULAR

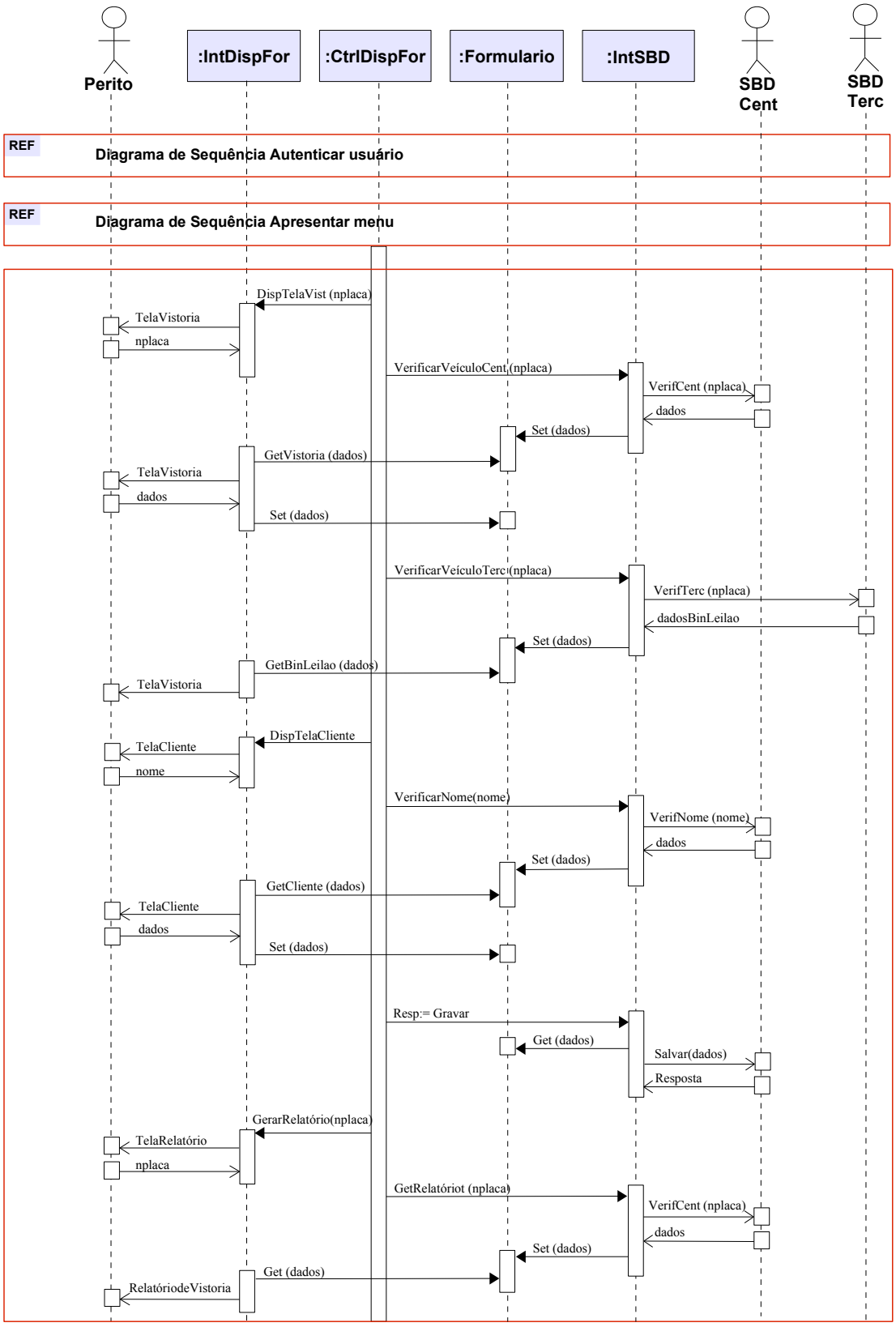


Figura 10: Diagrama de sequência vistoria veicular.

3.7. IMPLEMENTAÇÃO

3.7.1. SERVIDOR

3.7.1.1. Diagrama de Classes

O diagrama de classes tem por objetivo demonstrar as principais classes utilizadas na camada servidor e as relações entre elas no sistema. Elas provém os serviços necessários para as operações do sistema.

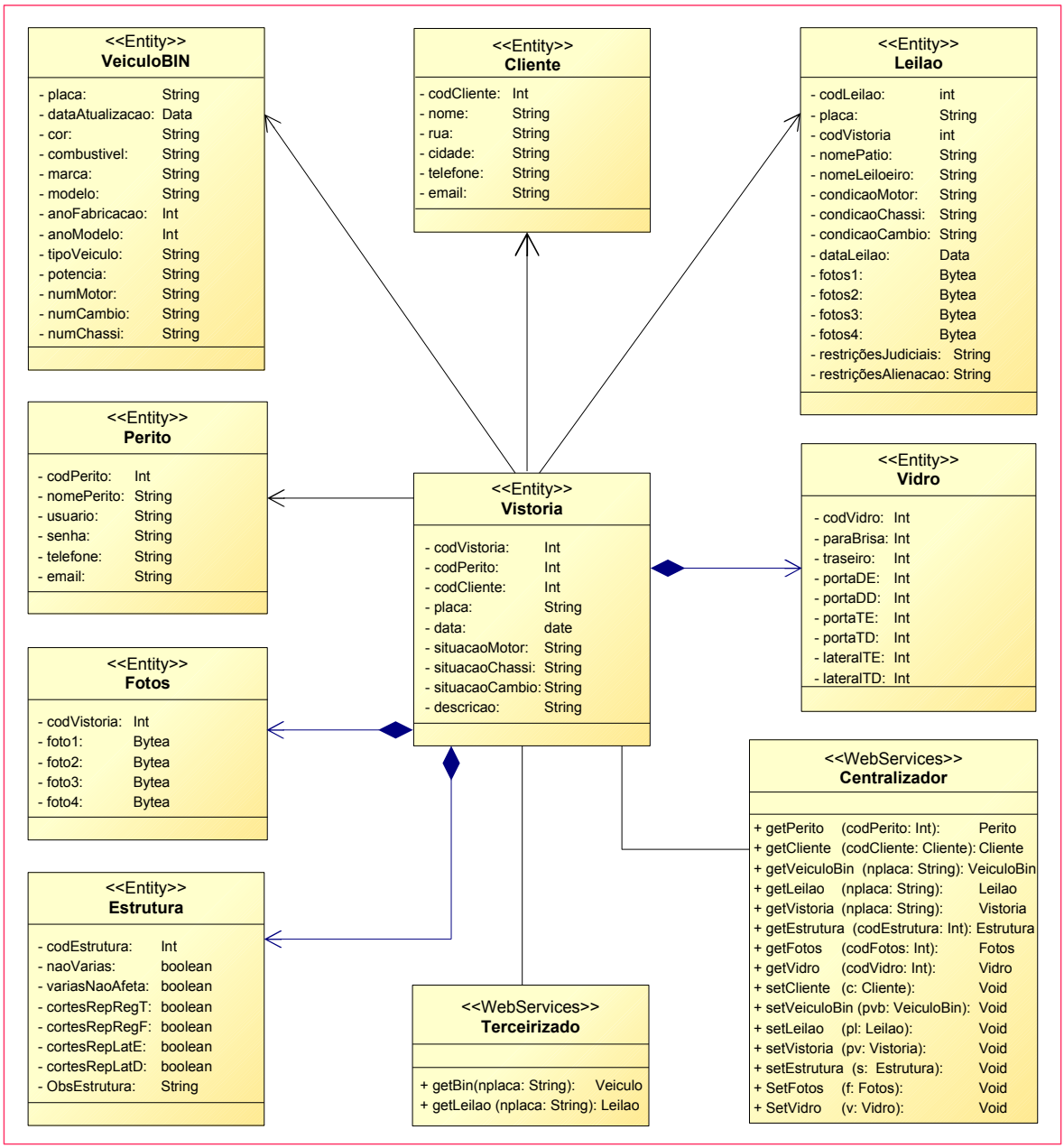


Figura 11: Diagrama de classes do servidor.

3.7.1.2. Entidades

Representam as classes que são mapeadas para persistência na base de dados. Para o desenvolvimento do protótipo foram criadas 8 entidades.

Entidades	**Descrição**
VeículoBin	Representa os dados originais do veículo, alguns são usados para comparação de itens de segurança.
Leilão	Contém o histórico do veículo quando leiloado, inclusive com fotos.
Perito	Contém os dados do usuário que tem acesso e gerenciamento do sistema, possibilita registro, alteração e geração de relatório.
Cliente	Descrição das informações do cliente que solicitou a vistoria e do local onde será realizado o procedimento.
Vistoria	Contém informações da vistoria realizada pelo perito, possibilita acesso as demais entidades. Através dela obtemos o histórico completo do veículo para geração do relatório.
Estrutura	Representa o suporte e o estado externo do veículo.
Fotos	Representa visualmente o estado do veículo quando vistoriado.
Vidro	Contém informações de originalidade dos vidros do veículo.

3.7.1.3. Diagrama de Estados

Permite especificar a dinâmica de um sistema, é uma descrição global do comportamento dos objetos de uma classe em todo o sistema. Demostra as transições de estados possíveis entre as entidades de um sistema. Diagrama de estado da entidade Vistoria apresenta 2 cenários:

1) Cenário de Pesquisa
    1. *WerbServices* aguardando serviço *(Aguardando chamada de serviço)*;
    2. Recebendo serviço *(Recebendo serviço)*.
    3. Selecionando serviço para execução *(Executando processo)*;
    4. Realizando pesquisa de vistoria nas entidades *(Pesquisando)*;
    5. Retornando dados da pequisa e encerrando processo *(Encerrando)*.

2) Cenário de Inserção

1. *WerbServices* aguardando serviço *(Aguardando chamada de serviço)*;
2. Recebendo serviço *(Recebendo serviço)*.
3. Selecionando serviço para execução *(Executando processo)*;
4. Inserindo dados de vistoria nas entidades *(Inserindo)*.
5. Encerra processo *(Encerrando).*

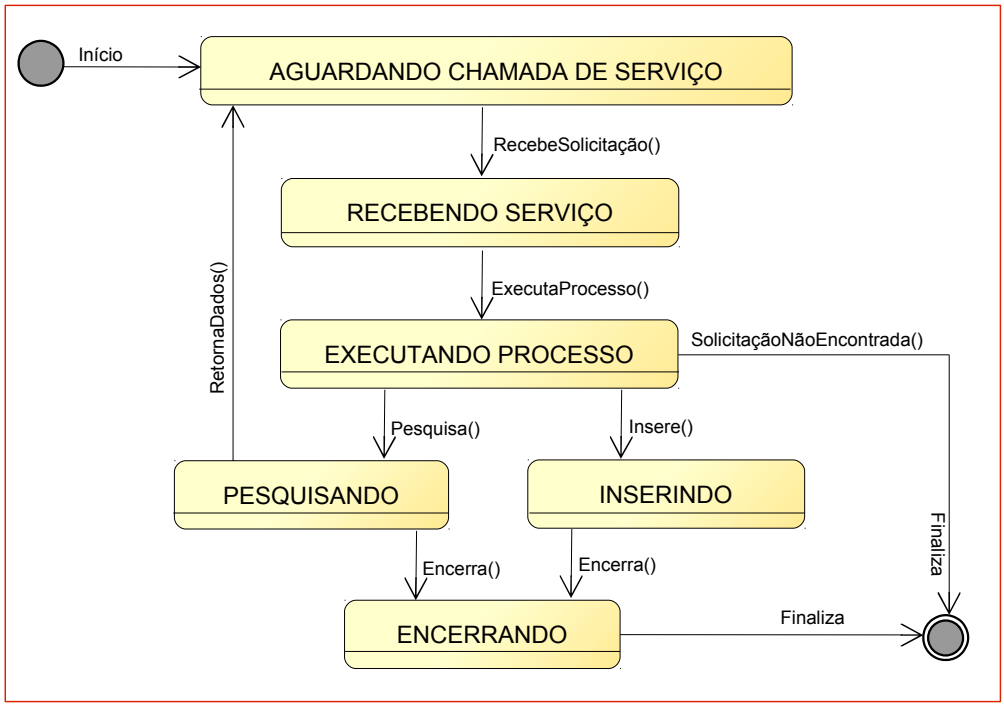


Figura 12: Diagrama de Estado.

3.7.1.4. WebServices

Para o desenvolvimento do protótipo foi utilizado a tecnologia *WebServices* que apresenta esquema de funcionamento onde o cliente invoca o serviço para o servidor *Web,* o qual retorna a resposta para o cliente, tal comunicação é realizada através dos protocolos *HTTP* e *SOAP* pela rede de comunicação sem fio.

Foi utilizado esta tecnologia para solução do requisito do sistema que é a comunicação do aplicativo móvel através de rede sem fio com um servidor externo.

O protótipo foi criado com *WebServices Centralizador* com 15 operações e com *WebServices Terceirizado* com 2 operações.

As classes do tipo *WebServices* expõem as operações utilizadas pelo aplicativo móvel de forma de serviços. São classes fornecedoras de serviços.

3.7.2. SISTEMA MÓVEL

3.7.2.1. Diagrama de Atividades

O objetivo do diagrama de atividades é mostrar o fluxo de atividades de um sistema. O diagrama apresenta como uma atividade depende da outra. A figura abaixo mostra as atividades realizadas pelo usuário da aplicação *VistoCar*, desde o login até o fechamento. Os retângulos azuis representam, de uma maneira geral, a fonte de dados usada para busca e inserção nos campos dos formulários.

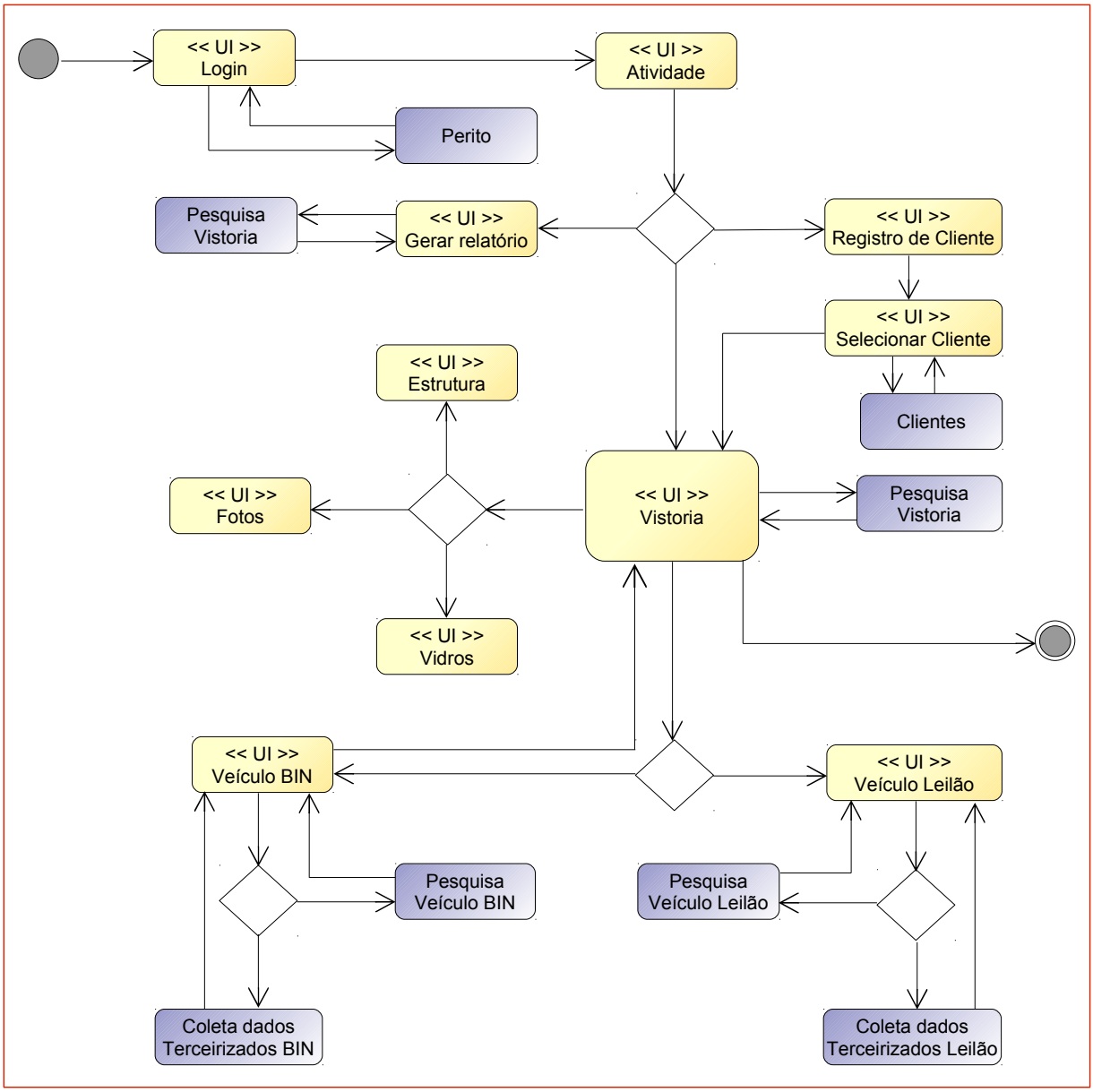


Figura 13: Diagrama de Atividades.

3.7.2.2. Diagrama de Classes

O diagrama de classes tem por objetivo demonstrar as principais classes utilizadas no aplicativo e as relações entre elas no sistema. Elas provêm os serviços necessários para as operações do sistema.

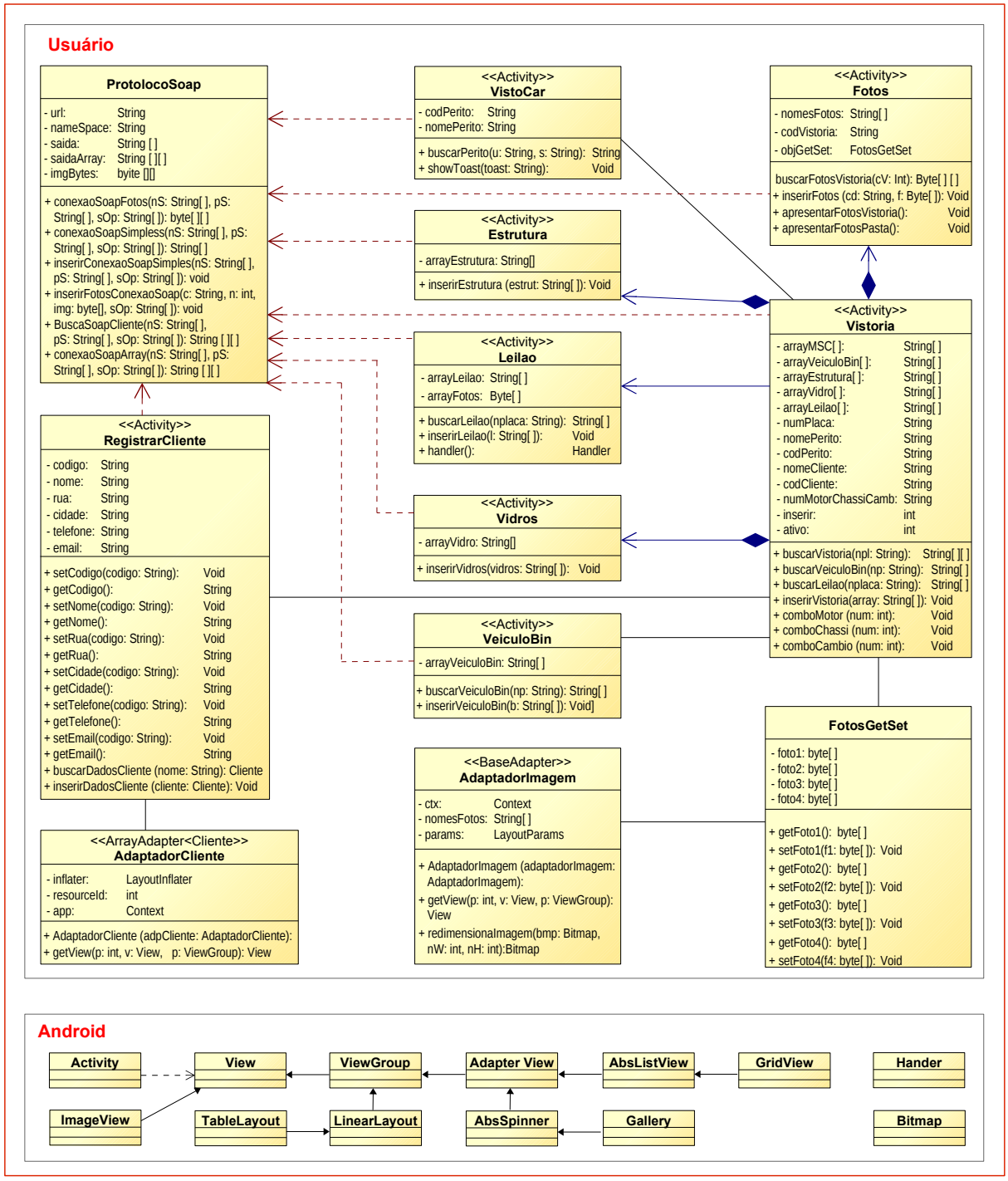


Figura 14: Diagrama de classes do aplicativo móvel.

3.7.2.3. Requisitos de acesso ao servidor

Para acesso ao método remoto no servidor é utilizado a biblioteca *Ksoap* através de *WebServices* contendo o nome de identificação e atributos. O resultado do acesso é um objeto genérico *SoapObject* contendo dados do tipo:

a) String

O *array* de *String* receberá os atributos do *SoapObject* e pode ser:

- Nulo, quando não encontrado dados da pesquisa.
- Não nulo, são dados que pode ser usado para campos de edição de texto, novos registros e constatação da veracidade da entrada de dados digitados.

b) byte

O *array* de byte receberá os atributos do *SoapObject* e pode ser:

- Nulo, quando não encontrado dados da pesquisa.
- Não nulo, são bytes que representam imagem de fotos do veículo. São usadas para exibição da imagem e novos registros.

3.7.2.4. Tela Login

Figura 15: Tela de Login

O aplicativo móvel inicia com a *Activity VistoCar* e disponibiliza a tela com os campos usuário e senha que são verificados no servidor conforme descrito no item 3.7.2.3. letra a.

Se o usuário for autenticado na base de dados do servidor, é retornado o código, o usuário é redirecionado para tela de *Atividade.* É enviado para nova tela o código e nome do usuário.

3.7.2.5. Tela Atividades

Figura 16: Tela de Atividades

A *Activity* de *Atividades* apresenta 3 opções: a primeira é a tela de *Cliente* que contém os serviços de registros, alteração e pequisa, a segunda é tela de *Vistoria* que permite acesso as telas de estrutura, fotos, vidros, leilão, cliente e dados veiculares e a terceira é para geração do relatório de vistoria. A primeira e segunda opção recebe os atributos código e nome do usuário.

3.7.2.6. Tela Cliente

A *Activity* de *Cliente* apresenta uma tela de cadastro, onde pode ser registrado novos clientes preenchendo os campos do formulário e pressionando o botão *"Salvar"*. Porém se o usuário desejar buscar o cliente para alteração de dados deve utilizar o campo nome para entrada de dados e pressionar o botão *"Pesquisar".*

Figura 17: Tela de Cliente

Botão Salvar - Envia para o servidor os dados inseridos nos campos do formulário de registro, inclusive o código do cliente quando o registro for selecionado para edição. Após esta operação o usuário é redirecionado para tela de *Atividades*.

Botão Pesquisar - Busca no servidor o conteúdo do campo nome e apresenta os dados retornados em outra tela, numa lista customizada, com a opção de alteração de registro exibida no botão *"Alterar"*. Se o usuário clicar no botão *"Alterar"* os dados da lista preenchem o formulário de registro de cliente onde podem ser reeditados. Caso seja pressionado algum item da lista será redirecionado para tela de *Vistoria*.

A tela escolhida para próxima atividade recebe os atributos código e nome do usuário e do cliente.

Para o envio e busca de dados é usada a tecnologia descrita no item 3.7.2.3. letra a.

3.7.2.7. Tela Vistoria

A *Activity* de *Vistoria* é a principal do aplicativo, contém botões que possibilitam o relacionamento eficiente entre as telas da aplicação.

O campo Cliente e Perito não são editados, seus dados são provenientes das telas anteriores. Se o campo cliente apresentar vazio, pode ser adicionado pressionando o botão *"Pesquisar"* que será redirecionado para tela *Cliente* conforme item 3.7.2.6.

O conteúdo do campo Motor, Chassi e Câmbio pode ser selecionado de uma lista de itens, conforme apresentado na figura 18a.

O campo Descrição permite inserção de dados, porém seu conteúdo pode ser proveniente conforme descrito no item 3.7.2.11 *"botão comparar"*.

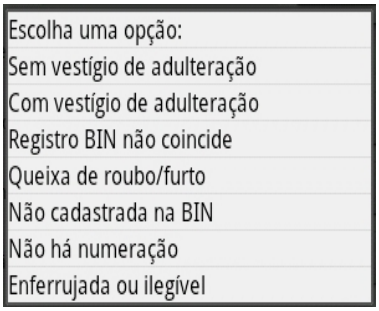


Figura 18a: Dados contidos na combo Motor, Chassi e Câmbio.

Figura 18: Tela de Vistoria

Botão Salvar - Envia para o servidor os dados inseridos nos campos do formulário, inclusive o código do usuário e do cliente, quando houver. No servidor é criado um registro na tabela de Vistoria gerando sequencialmente um valor numérico para o código do registro. São criado também registros nas tabelas Estrutura, Fotos e Vidros sendo que o código gerado anteriormente será adicionado nestas tabelas para relacionamento. Este valor será único para as tabelas citadas *"Primary Key"*.

Botão Pesquisar - Os botões apresentados nesta tela, exceção *"Pesquisar"* do campo Cliente, somente possibilitam alguma ação, após o conteúdo do campo Placa ser analisado no servidor.
Se a pesquisa é encontrada retorna dados das telas Vistoria, Estrutura, Vidros, Usuário, Cliente, Leilão e Veículo, se eles existirem. Caso deseja realizar nova vistoria do mesmo veículo, pode pressionar o botão *"Salvar"*.
Se a pesquisa não é encontrada é preciso preencher o formulário de vistoria e salvá-lo para depois realizar outras ações.

Para o envio e busca de dados é usada a tecnologia descrita no item 3.7.2.3. letra a.

3.7.2.8. Tela Estrutura

A *Activity* de *Estrutura* representa o suporte e o estado externo do veículo, "lataria", contém campos que possibilitam marcações de defeitos encontrados numa vistoria.

Ao iniciar a tela Estrutura seus campos são preenchidos automaticamente, caso seja encontrado registro da vistoria pesquisada.

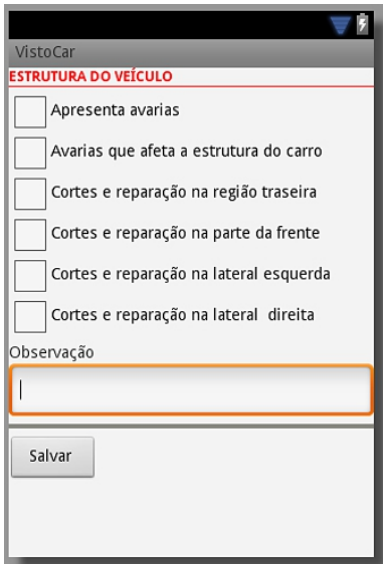


Figura 19: Tela de Estrutura.

Botão Salvar - Envia para o servidor os dados apresentados no formulário, sendo os campos ticados como *true* e os não ticados como *false.* O código da estrutura criado na *Activity* de *Vistoria* é enviado para localização do registro e alteração dos dados.

Para o envio e busca de dados é usada a tecnologia descrita no item 3.7.2.3. letra a.

3.7.2.9. Tela Vidros

A *Activity* de *Vidros* representa a situação dos vidros que compõem o veículo, contém campos que possibilitam marcação de originalidade ou inconsistências encontradas numa vistoria. As legendas apresentadas de 1 a 6 são opções de escolhas para inserção nos campos.

Ao iniciar a tela Vidros seus campos são preenchidos automaticamente, caso seja encontrado registro da vistoria pesquisada.

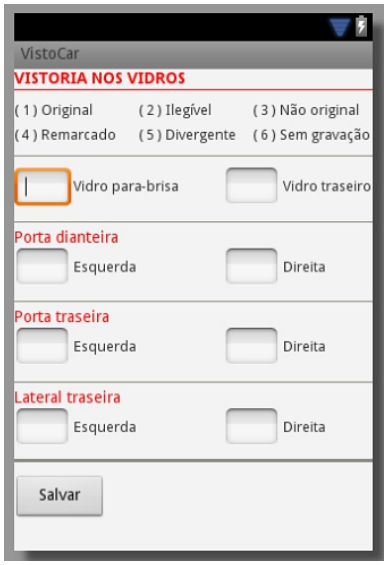


Figura 20: Tela de Vidros.

Botão Salvar - Envia para o servidor os dados apresentados no formulário. O código da estrutura criado na *Activity* de *Vistoria* é enviado para localização do registro e alteração dos dados.

Para o envio e busca de dados é usada a tecnologia descrita no item 3.7.2.3. letra a.

3.7.2.10. Tela Fotos

A *Activity* de *Fotos* representa imagens visuais para comprovação visual da integridade do veículo no momento da vistoria. O componente *Gallery* exibe as fotos provindas da vistoria ou da pasta do *Smartphone*.

A tela Fotos ao ser iniciada recebe o código que é utilizado para inserção de novas imagens, tendo como limite 4 fotos, ou para acesso as fotos da vistoria.

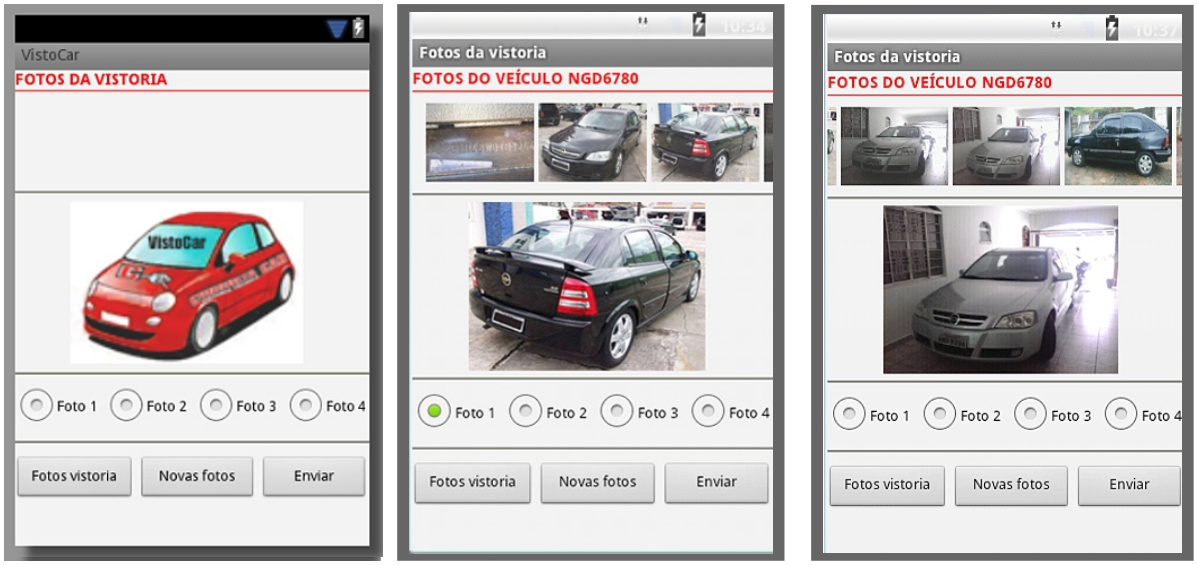


Figura 21: Apresenta a tela 1 sem vistoria, a tela 2 com fotos transmitidas por *WebServices* e a tela 3 com fotos da pasta do *Smartphone*.

Botão Fotos Vistoria - Busca as fotos no servidor pelo código da vistoria e apresenta as imagens quando existirem, em forma de galeria com limite de 4 fotos. O tamanho da foto é redimensionado para agilizar o processo de busca no servidor.

Ao clicar numa figura um dos botões de opções referente a ela é marcado e a imagem é exibida em formato maior para melhor visualização.

Botão Novas Fotos - Busca no *Smartphone* as imagens da pasta DCIM, e apresenta na galeria. Ao clicar na imagem é exibida em formato maior para melhor visualização.

Botão Enviar - Envia para o servidor o código da vistoria e a imagem exibida em formato maior do componente central.
Só é permitido o envio da imagem quando um dos botões de opções é marcado.
Este processo é usado tanto para inserção de novas fotos no servidor como para substituição das existentes.

Para o envio e busca de dados é usada a tecnologia descrita no item 3.7.2.3. letra b.

3.7.2.11. Tela VeículoBin

A *Activity* de *VeiculoBin* representa os dados veiculares fornecidos por empresas conveniadas por órgãos governamentais conforme descrito no item 2.1.1.

Os dados originais exibidos neste formulário não podem ser editados. Somente é permitida a edição do campo Motor, Chassi e Câmbio referente a dados coletados.

Figura 22: Tela de dados da BIN.

Ao iniciar a tela *VeiculoBin* seus campos são preenchidos automaticamente, caso seja encontrado registro da vistoria pesquisada. Não encontrando registro, o campo Placa será preenchido pelo número provindo da tela de *Vistoria*.

Botão Pesquisa BIN - Busca os dados no servidor Terceirizado pelo conteúdo do campo placa e apresenta no formulário.

Botão Comparar - O usuário poderá inserir dados para comparação de originalidade com os dados fornecidos pela empresa conveniada. Pressionando o botão *Comparar* a tela é redirecionada para a *Activity* de *Vistoria* e a resposta da comparação é exibida no campo Descrição.

Botão Salvar - Envia os dados inseridos nos campos pelo servidor Terceirizado para o servidor Central. Este procedimento faz uma réplica dos dados, evitando assim novas consultas no servidor Terceirizado.

Para o envio e busca de dados é usada a tecnologia descrita no item 3.7.2.3. letra a.

3.7.2.12. Tela Leilão

Figura 23: Apresenta a tela 1 sem registro na base leilão e a tela 2 com fotos transmitidas por *WebServices* da base leilão.

A *Activity* de *Leilão* representa os dados veiculares fornecidos por empresas conveniadas por órgãos governamentais conforme descrito no item 2.1.1.

Os dados exibidos neste formulário não podem ser editados.

Ao iniciar a tela *Leilão* seus campos são preenchidos automaticamente, inclusive com fotos, caso seja encontrado registro da vistoria pesquisada. Quando esta tela é iniciada recebe o número da placa e o código da vistoria da tela anterior.

Botão Pesquisa Leilão - Busca os dados no servidor Terceirizado pelo número da placa e apresenta no formulário.

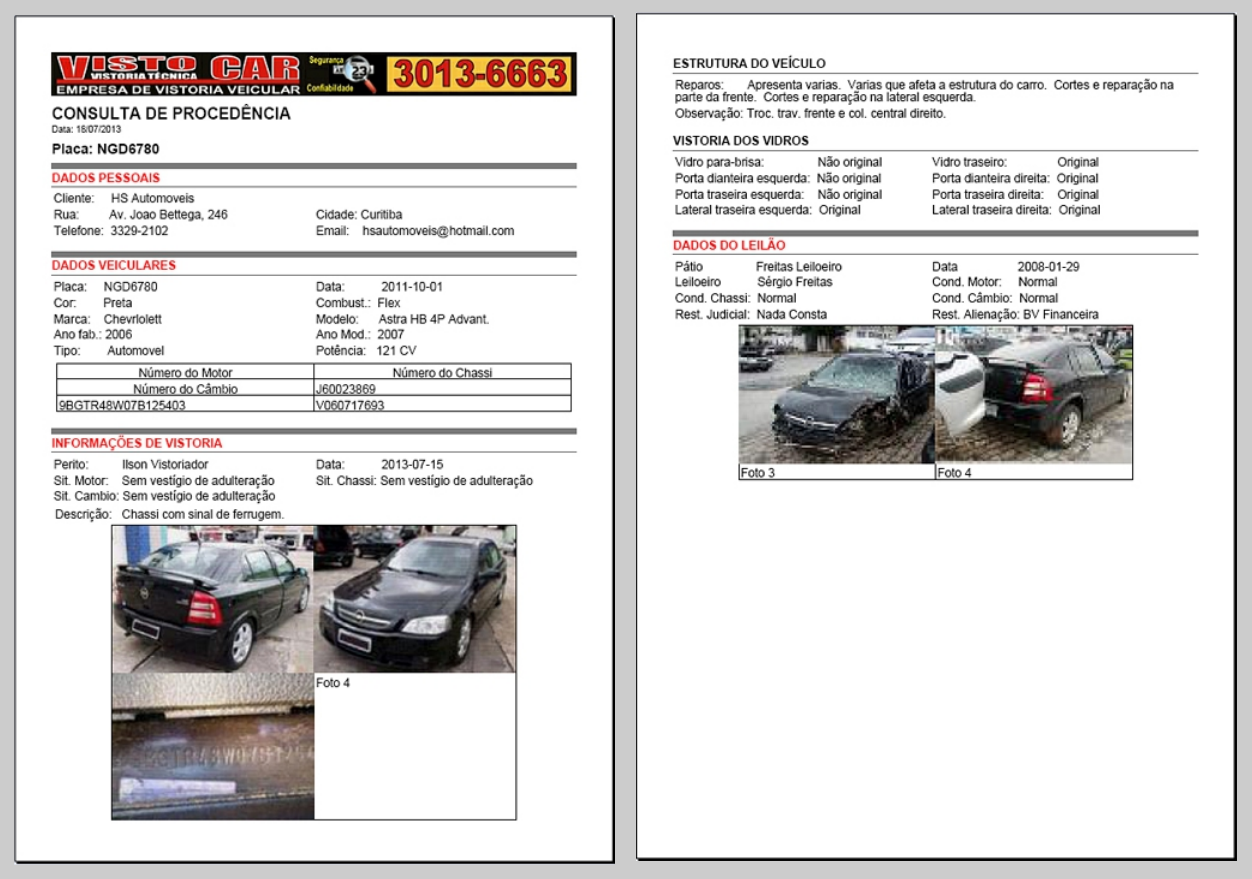
Botão Salvar - Envia o código da vistoria e os dados inseridos nos campos pelo servidor Terceirizado para o servidor Central. Este procedimento faz uma réplica dos dados, evitando assim novas consultas no servidor Terceirizado num período de 6 meses.

Para o envio e busca de dados é usada a tecnologia descrita no item 3.7.2.3. letras a e b.

3.7.2.13. Tela Relatório

Figura 24: Tela de Relatório.

Apresenta a tela de entrada de dados para pesquisa de vistoria. Os dados retornados são usados para elaboração do relatório de consulta de procedência e vistoria. O arquivo é criado em formato PDF e armazenado no *smartphone* na pasta PDF tendo como nome do arquivo a placa do veículo.


VISTO CAR
VISTORIA TÉCNICA
EMPRESA DE VISTORIA VEICULAR
3013-6663

CONSULTA DE PROCEDÊNCIA
Data: 18/07/2013
Placa: NGD6780

DADOS PESSOAIS
Cliente: HS Automoveis
Rua: Av. Joao Bettega, 246 Cidade: Curitiba
Telefone: 3329-2102 Email: hsautomoveis@hotmail.com

DADOS VEICULARES
Placa: NGD6780 Data: 2011-10-01
Cor: Preta Combust.: Flex
Marca: Chevrlolett Modelo: Astra HB 4P Advant.
Ano fab.: 2006 Ano Mod.: 2007
Tipo: Automovel Potência: 121 CV

Número do Motor	Número do Chassi
Número do Câmbio	J60023869
9BGTR48W07B125403	V060717693

INFORMAÇÕES DE VISTORIA
Perito: Ilson Vistoriador Data: 2013-07-15
Sit. Motor: Sem vestígio de adulteração Sit. Chassi: Sem vestígio de adulteração
Sit. Cambio: Sem vestígio de adulteração
Descrição: Chassi com sinal de ferrugem.

Foto 4

ESTRUTURA DO VEÍCULO
Reparos: Apresenta varias. Varias que afeta a estrutura do carro. Cortes e reparação na parte da frente. Cortes e reparação na lateral esquerda.
Observação: Troc. trav. frente e col. central direito.

VISTORIA DOS VIDROS
Vidro para-brisa: Não original Vidro traseiro: Original
Porta dianteira esquerda: Não original Porta dianteira direita: Original
Porta traseira esquerda: Não original Porta traseira direita: Original
Lateral traseira esquerda: Original Lateral traseira direita: Original

DADOS DO LEILÃO
Pátio Freitas Leiloeiro Data 2008-01-29
Leiloeiro Sérgio Freitas Cond. Motor: Normal
Cond. Chassi: Normal Cond. Câmbio: Normal
Rest. Judicial: Nada Consta Rest. Alienação: BV Financeira

Foto 3 Foto 4

Figura 25: Arquivo PDF gerado pela vistoria do veículo placa NGD6780 contendo duas páginas.

Botão Relatório - Busca uma vistoria no servidor pelo conteúdo do campo placa e apresenta os dados retornados no formato de texto e imagem que são usados para composição do arquivo PDF.

Na busca de dados é usada a tecnologia descrita no item 3.7.2.3. letra a e b.

4. CONSIDERAÇÕES FINAIS

Os conceitos de *Android* e *WebServices* estão em pauta atualmente, estudá-los foi bastante proveitoso. No decorrer do curso foram apresentadas várias disciplinas que foram aplicadas no trabalho contribuindo para fixação e maior aproveitamento de tudo que foi apreendido.

A plataforma *Android* foi escolhida para o trabalho por ser um projeto de código aberto, bastante popular, além da excelente qualidade das ferramentas disponíveis (IDE, Emulador) e documentações.

O trabalho foi adequado para aprofundamento dos conhecimentos sobre as tecnologias *Android*, *WebServices* e Banco de Dados. Durante o desenvolvimento do protótipo, pela falta de experiência, houve alguns imprevistos e obstáculos como transmissão e obtenção de dados, principalmente imagens, por *WebServices*. Superá-los foi um grande desafio que permitiu a criação de uma aplicação útil para vistoria que pode ser expandida posteriormente.

O trabalho apresentado cumpriu o objetivo proposto de desenvolvimento do protótipo de vistoria veicular preventiva para dispositivo móvel. Foi feita a automatização apresentando formulário de fácil preenchimento com itens de vistoria que são armazenados na base centralizada. Foi realizada a integração de dados provindos de base terceirizada e inserida na base centralizada. Estas atividades otimizam o tempo e o gasto operacional da realização de uma vistoria, além de possibilitar a redução de custo em novas vistorias do mesmo veículo. Foi implantada a tecnologia *WebServices* para transferência de dados na comunicação com o servidor.

O protótipo desenvolvido espera contribuir com as empresas que realizam serviços de vistoria veicular preventiva, oferecendo às mesmas a possibilidade do uso da tecnologia móvel para efetuarem suas atividades. A mobilidade têm muito a oferecer para as empresas e podem ajudar a otimizar os serviços e reduzir custos.

4. 1. PROJETOS FUTUROS

A aplicação desenvolvida neste projeto é uma versão simplificada de vistoria preventiva e poderá ser expandida adicionando módulos e implementando melhorias como:

- Agendamento

  Agendar uma vistoria por data e horário desejado, classificando-as por ordem de atendimento. Receber automaticamente mensagem de notificação 3 horas antes do horário agendado.

- Localização

  Oferecer serviços de posição geográfica que apresente um mapa com indicação da localização do cliente através dos dados do usuário.

- Administrativo

  Implantar um módulo administrativo para liberar acesso ao aplicativo para os peritos.

- Consultas

  Apresentar relatório de vistorias feitas por dia por um perito.

  Apresentar relatório da quantidade de consultas feitas na base Leilão e VeículoBIN.

- Usar o REST *(Representational State Transfer)*

  O trabalho foi desenvolvido utilizando o protocolo SOAP, porém o REST é uma abordagem proposta para *WebServices* utilizando o protocolo HTTP. O REST por ser mais leve reduziria a carga de transferência de dados na comunicação com o servidor.

5. REFERÊNCIAS

CONSULTAAUTO. **Consulta de Veículos**. Disponível em http://www.consultaauto.com.br/consulta-veiculo. Acesso em: Agosto/2013.

DENATRAN**. Resoluções do Contran**. Disponível em: http://www.denatran.gov.br/download/Resolucoes/RESOLUCAO_CONTRAN_282.pdf. Acesso em: março/2013.

HENDRICKS, Mack; GALBRAITH, Bem; IRANI, Romin; MILBERNY, James; MODI, Tarak; TOST, André; TOUSSAINT, Alex; BASHA, S. Jeelani, CABLE, Scott. **Profissional Java Web Services**. Rio de Janeiro: Editora Alta Books, 2002.

KUHN, Clayton Eduardo. **Elaboração de um Protótipo de Aplicativo para Acompanhamento de Requisições de Táxi.** Monografia de Especialização do curso de Tecnologia Java, UTFPR, 2012.

LECHETA, Ricardo R. **Google Android: aprenda a criar aplicações para dispositivos móveis com o Android SDK**. São Paulo: Novatec Editora, 2010.

MEDEIROS, Higor. **Utilizando XML para Solicitação e Respostas**. Disponível em: http://www.linhadecodigo.com.br/artigo/3622/utilizando-xml-para-solicitacao-e-respostas.aspx. Acesso em: Abril/2013.

OLHO VIVO VISTORIAS AUTOMOTIVAS. **Vistoria Cautelar**. Disponível em http://www.olhovivovistorias.com.br/vistorias/vistoria-cautelar.html. Acesso em: Agosto/2013.

OLIVEIRA, Eric C. **Projeto Eclipse for Java**. Disponível em: http://www.linhadecodigo.com.br/artigo/677/projeto-eclipse-for-java.aspx. Acesso em: Abril/2013.

PEREIRA, Ana Paula. **Apache Tomcat 7.0.39. O Tomcat é o Servidor Web que pode ser usado também para tecnologia Java sendo mais leve do que a maioria, Baixaki,** março 2013. Disponível em: http://www.baixaki.com.br/download/apache-tomcat.htm. Acesso em abril/2013

RECKZIEGEL, Maurício. **Entendendo os WebServices**. Disponível em: http://imasters.com.br/artigo/4245/web-services/entendendo-os-webservices. Acesso em: Abril/2013.

RIBEIRO, Gabriella Fonseca. **Chamada remota de procedimentos .net**, agosto 2011. Disponível em: http://www.slideshare.net/gabii_fonseca/chamada-remota-de-procedimentos-net. Acesso em: Abril/2013.

SINALIZA.NET. **Consultas Veiculares**. Disponível em http://www.sinaliza.net/SINSITPRD00101.aspx. Acesso em: Agosto/2013.

SOBRE PostgreSql. **Sobre o PostgreSQL**. Disponível em: http://www.postgresql.org.br/sobre. Acesso em: Abril/2013.

SPINOLA, Eduardo Oliveira. **Desenvolvendo Web Services utilizando JAX-WS**. Disponível em: http://www.devmedia.com.br/desenvolvendo-web-services-utilizando-jax-ws/2374. Acesso em: Abril/2013.

TOSIN, Carlos. **Conhecendo o Android**. Disponível em: http://www.softblue.com.br/blog/home/postid/11/CONHECENDO+O+ANDROID Acesso em: Abril/2013.