CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

FERNANDO NUNES

AVALIAÇÃO DE TÉCNICAS E MECANISMOS PARA ENTRADA E SAÍDA DE INFORMAÇÕES EM DISPOSITIVOS MÓVEIS

MARÍLIA
2014

FERNANDO NUNES

AVALIAÇÃO DE TÉCNICAS E MECANISMOS PARA ENTRADA E SAÍDA DE INFORMAÇÕES EM DISPOSITIVOS MÓVEIS

Trabalho de Curso apresentado ao Curso de Bacharelado em Sistemas de Informação da Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípedes de Marília - UNIVEM, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador:
Prof. Ms. Ricardo José Sabatine

MARÍLIA
2014

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
BACHARELADO EM SISTEMAS DE INFORMAÇÂO

TRABALHO DE CONCLUSÃO DE CURSO – AVALIAÇÃO FINAL

Fernando Nunes

AVALIAÇÃO DE TÉCNICAS E MECANISMOS PARA ENTRADA E SAÍDA DE INFORMAÇÕES EM DISPOSITIVOS MÓVEIS

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Sistemas de Informação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Sistemas de Informação.

Nota: 8.5 (oito e meio)

Orientador: Ricardo José Sabatine

1°. Examinador: Elvis Fusco

2°. Examinador: Mauricio Duarte

Ricardo Sabatine

Marília, 03 de dezembro de 2014.

Dedico este trabalho primeiramente a Deus que me capacitou, aos meus pais por todo amor e carinho e a minha esposa Gabriela por toda paciência e incentivo nos momentos difíceis, *amo vocês!!!*

AGRADECIMENTOS

Agradeço a Deus por ter me dado forças ao longo de todo curso e ter meu auxiliado chegar até aqui.

Aos meus pais Alberto e Aparecida por todo incentivo e apoio nos momentos mais difíceis.

A minha esposa Gabriela pelo apoio e paciência e incentivo.

Aos amigos e profissionais Pedro Henrique, André Guedes e Sidnei Mendes Junior pelo apoio e o compartilhamento de experiências que contribuíram para o presente trabalho.

Ao meu orientador Ricardo José Sabatine, que através de sua experiência foi de uma colaboração impar para pesquisa desse trabalho.

“Tenha coragem de seguir seu coração e intuição. Eles já sabem o que você realmente deseja. Todo resto é secundário”.
(Steve Jobs)

NUNES, Fernando. **Avaliação de Técnicas e Mecanismos para Entrada e Saída de informações em Dispositivos Móveis**. 2014. 55 f. Trabalho de Curso (Bacharelado em Sistemas de Informação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2014.

RESUMO

Nos últimos anos, a procura por mobilidade aumentou consideravelmente. Consumidores desde usuários finais até empresas de pequeno a grande porte têm procurado tecnologias que atendam todas as suas necessidades diárias e que possam ser utilizadas em qualquer lugar e a qualquer momento. Desta forma, o mercado de dispositivos móveis vem a cada ano se tornando mais poderoso e promissor. Segundo a Gartner, empresa especializada em pesquisas, cerca de três bilhões de pessoas já possuem ao menos um dispositivo móvel, e este número aumenta a cada ano. Apesar das inúmeras vantagens que os dispositivos móveis oferecem, é importante levar em consideração que, por se tratar de uma tecnologia ainda considerada nova, muitos desafios precisam ser superados ao se utilizar um dispositivo como limitações de memória, bateria e interface por exemplo. Aplicações demasiadamente lentas, com péssimas estruturas e códigos complexos têm sido criados por parte dos desenvolvedores, o que afeta diretamente na integridade, segurança e confiabilidade com que os dados e informações estão sendo tratados. Sendo assim, a presente monografia apresenta técnicas de banco de dados e serviços web disponíveis para o auxílio na manipulação de dados e informações em dispositivos móveis a fim de sanar alguns dos problemas citados a cima evitando o excesso de uso dos dispositivos móveis e a perda de dados e informações

Palavras-Chave: Armazenamento de Informações, Boas Práticas, Dispositivos Móveis, Android.

NUNES, Fernando. **Avaliação de Técnicas e Mecanismos para Entrada e Saída de Informações em Dispositivos Móveis**. 2014. 55 f. Trabalho de Curso (Bacharelado em Sistemas de Informação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2014.

ABSTRACT

In recent years, the demand for increased mobility significantly. Consumers from end users to the small to large companies have sought technologies that meet all your daily needs and can be used anywhere and anytime. Thus, the mobile device market each year is becoming more powerful and promising. According to Gartner, enterprise specialized in research, about three billion people already own at least one mobile device and this number increases every year.
Despite the many advantages that mobile devices offer, it is important to consider that, because it is still considered new technology, many challenges need to be overcome when using a device such as memory limitations, battery and interface for example. Too slow applications with complex code and bad structures have been created by the developers, which directly affects the integrity, security and reliability with which data and information are being handled. Thus, this monograph presents techniques and best practices available to aid in the manipulation of data and information to mobile devices, remedy some of the problems mentioned above to avoid excessive use of mobile devices and the loss of data and information.

Keywords: Information Store, Best Practices, Mobile Devices, Android.

LISTA DE ILUSTRAÇÕES

Figura 1 - Estrutura iOS 15
Figura 2 - Estrutura Android..... 19
Figura 3 – Estrutura Sqlite..... 25
Figura 4 – ActiveAndroid..... 27
Figura 5 - Arquitetura interna do AlienDroid..... 28
Figura 6 - REST Web service design structure..... 30
Figura 7 – Diagrama Entidade Relacionamento..... 35
Figura 8 – Método OnCreate com as funções de Criação e Inserção dos dados no banco 36
Figura 9 – Classe CrudDao com os métodos do CRUD..... 37
Figura 10 - Classe Funcionario no ActiveAndroid..... 38
Figura 11 – Classe Setor no ActiveAndroid. 39
Figura 12 – Classe DAO com os métodos do CRUD no ActiveAndroid..... 40
Figura 13 - Classe Funcionario no AlienDroid..... 41
Figura 14 - Classe Setor no AlienDroid. 41
Figura 15 - Classe DAO com os métodos de Consulta no AlienDroid 42
Figura 16 - Classe CarregarImagemUrl utilizando AsyncTask..... 43
Figura 17 - Utilização da Biblioteca Picasso..... 44
Figura 18 - Estrutura do Json na Apiary.io 45
Figura 19 - Utilização do JSON 46
Figura 20 – Classe RestClient - Json..... 47
Figura 21 - Utilização API Volley 48
Figura 22 - Catalogo de Carros - Json..... 49

LISTA DE ABREVIATURAS E SIGLAS

OEM - Original Equipment Manufacturer

UI - User Interface

SO - Sistema Operacional

SMS - Short Message Service

MMS - Multimedia Messaging Service

API - Application Programming Interface

SQL - Structured Query Language

PHP - Personal Home Page

ANSI - American National Standards Institute

SGBD - Sistema de Gerenciamento de Banco de Dados

ORM - Object-relational mapping

CRUD - Create, Read, Update e Delete

DAO - Data Access Object

JSON - JavaScript Object Notation

XML - Extensible Markup Language

HTTP - HyperText Transfer Protocol

WSDL - Web Services Description Language

SOAP - Simple Object Acess Protocol

REST - Representational State Transfer

URL - UniformResourceLocator

E-R - Entidade Relacionamento

2D - Objetos e Entidades com duas dimensões

3D - Objetos e Entidades com três dimensões

USB - Universal Serial Bus

VGA - Video Graphics Array

H.264 - Padrão para compressão de vídeo

MPEG-4 - Padrão para compressão de dados digitais de áudio e vídeo

MP3 - Padrão de arquivos digitais de áudio

AAc - AdvancedAudioCoding

GPS - Global Positioning System

LISTA DE TABELAS

Tabela 1 – Resultados dos testes para a persistência em banco de dados. 52
Tabela 2 – Resultados dos testes para o carregamento de Imagens 53
Tabela 3 – Resultados dos testes de integração JSON com WebService 53

SUMÁRIO

INTRODUÇÃO 13
CAPÍTULO 1 – DISPOSITIVOS MÓVEIS 14
1.1 Principais Plataformas Móveis 14
1.1.1 iOS 14
1.1.2 Windows Phone 15
1.1.2.1 UI Windows Store 16
1.1.2.2 Live Tiles 16
1.1.3 Android 16
1.1.3.1 Estrutura do Android 18
1.1.3.2 Componentes do Android 20
1.2 Desafios do Android 21
CAPÍTULO 2 – TÉCNICAS E MECANISMOS DE ENTRADA E SAÍDA NO ANDROID 23
2.1 Mecanismos de Entrada e saída 23
2.1.1 SharedPreferences 23
2.1.2 InternalStorage 23
2.1.3 ExternalStorage 24
2.1.4 SQLiteDatabase 24
2.1.4.1 ActiveAndroid 25
2.1.4.2 AlienDroid 27
2.1.5 WebService 29
2.1.6 Volley 31
2.1.7 AsyncTask 31
2.1.8 JSON 33
CAPÍTULO 3 – IMPLEMENTAÇÃO E COMPARATIVOS ENTRE AS TÉCNICAS DE ENTRADA E SAIDA NO ANDROID 34
3.1 Técnicas de implementação para persistência em banco de dados 34
3.1.1Utilização de Componentes nativos 35
3.1.2 Utilização da Ferramenta ActiveAndroid 37
3.1.3 Utilização da Ferramenta AlienDroid 40
3.2 Técnicas de implementação para consumir WebService 42
3.2.1 Consumindo imagens (REST) 42
3.2.1.1 Utilização dos componentes nativos 42

3.2.1.2 Utilização da API PICASSO 44
3.2.2 Consumindo JSON (REST) 44
3.2.2.1 Implementação manual 45
3.2.2.2 Implementação utilizando a API Volley 48
CAPÍTULO 4 – RESULTADOS E ANÁLISES 50
4.1 Banco de Dados 50
4.2 WebServices 52
4.2.1 Carregando Imagens 52
4.2.2 Carregamento de JSON 53
4.3 Análises dos Resultados 54
4.3.1 Banco de Dados 54
4.3.2 WebService 54
CONCLUSÃO 55
REFERÊNCIAS 56

INTRODUÇÃO

Com a crescente demanda e utilização de dispositivos móveis pela sociedade, exige-se cada dia mais uma maior segurança e confiabilidade no armazenamento de dados e informações nos *smartphones*, porém apesar de proporcionar diversos benefícios, os *smartphones* ainda possuem algumas limitações de processamento e bateria, por exemplo, que agregados a aplicações pesadas podem contribuir na perda de informações e dados corrompidos. A escolha das técnicas e mecanismos de entrada e saída de dados no dispositivo móvel pode impactar diretamente quanto à integridade dos dados armazenados.

Diante deste contexto, este trabalho propõe apresentar técnicas para a persistência de dados em banco e *WebService*, disponíveis para o auxílio na manipulação de dados e informações em dispositivos móveis a fim de, sanar diversos problemas existentes atualmente quando se trata de trabalhar com dados, salvá-los, consultá-los e enviá-los por meio de técnicas não recomendadas e que acarretam em aplicações demasiadamente lentas e com péssimos códigos fonte, em que muitas instruções de acesso ao banco de dados e conexões a rede estão mescladas com o código fonte da aplicação.

Líder no seu segmento de mobilidade, o *Android* foi o sistema operacional escolhido como base para todo o desenvolvimento do trabalho, pois permitiu vários cenários de implementações e testes devido à grande diversidade de *smartphones* que o utilizam como sistema operacional, outra vantagem é que por ser *open source* permitirá também uma maior exploração de seus recursos nativos.

CAPÍTULO 1 – DISPOSITIVOS MÓVEIS

1.1 Principais Plataformas Móveis

Estão disponíveis diversas plataformas móveis para que o desenvolvedor crie seus aplicativos, porém grandes partes destas plataformas são incompatíveis entre si, o que não permite com que uma aplicação seja executada em duas plataformas diferentes. Outro ponto importante é que, todo dispositivo móvel suporte uma única plataforma, desta forma é extremamente necessário que o desenvolvedor escolha de forma cuidadosa e criteriosa a plataforma ideal a se utilizar na criação de suas aplicações a fim de, evitar problemas futuros e ampliar o alcance de suas aplicações. A seguir são apresentadas as principais plataformas para dispositivos móveis no mercado. (MARIA, 2010).

1.1.1 iOS

iOS é um sistema operacional projetado e criado com a tecnologia *multitouch*, ou seja, telas sensíveis que são utilizadas através do toque manual. Após o lançamento da sua quarta versão, o iOS transformou-se em um sistema multitarefa melhorando ainda mais seu desempenho. É possível também tornar o iOS em um sistema multiusuário, porém somente através da instalação de um aplicativo fornecido através da loja da *Apple*. Por ser um sistema operacional de código fechado e proprietário, o iOS permite sua instalação em dispositivos que não competem à empresa desenvolvedora. (MARIA, 2010).

Para o desenvolvimento de aplicações no iOS, o desenvolvedor conta com uma ferramenta chamada XCODE que roda no MAC, esta ferramenta se caracteriza por sua extensa capacidade e simplicidade na utilização. (GADELHA, 2012).

Para a produção e venda de aplicações nas plataformas *APPLE*, é necessário que o desenvolvedor faça o registro como *developer* na *Apple* e pague anualmente uma taxa, também é necessário pagar uma comissão sobre o valor de cada aplicativo vendido. (GADELHA, 2012).

Segundo GADELHA, 2012, a estrutura do sistema IOS é dividida em quatro camadas definidas como *Core OS*, *Core Services*, *Media* e *CocoaTouch*. A *CocoaTouch* é a de maior nível e as *Core OS* as de mais baixo nível.

CocoaTouch: Principal Camada para a criação de aplicações e eventos no iOS,

responsável por fornecer toda infraestrutura e ferramentas necessárias para a implementação.

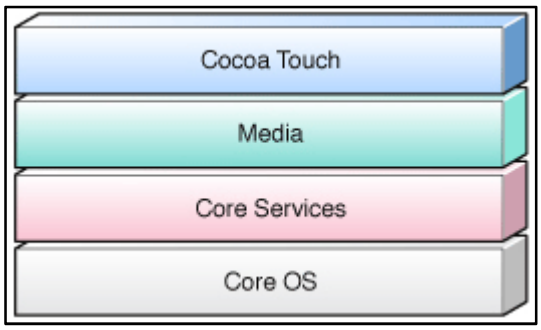
Media: Esta camada disponibiliza vários recursos multimídia para a criação das aplicações, tais como recursos de áudio e vídeo. A tecnologia fornecida por esta camada proporciona ao desenvolvedor excelentes experiências em multimídia.

Core Services: Camada responsável por fornecer os essenciais serviços do sistema operacional utilizados pelas aplicações, serviços tais como de Endereço, Localização, Rede, Segurança e Banco de dados.

Core OS: Camada que funciona como intermediária entre o *hardware* e as demais camadas do Sistema e que, permite o acesso a recursos do sistema por meio de um grupo de interfaces localizadas em bibliotecas. Nada mais é do que a camada que permite com que o telefone faça aquilo na qual ele foi projetado.

A figura 1ilustra a estrutura do Sistema Operacional iOS.

Figura 1 - Estrutura iOS



Fonte: http://sushiprogramador.files.wordpress.com/2011/03/

De forma objetiva, o iOS foi criado e projetado para ser um sistema que forneça segurança, estabilidade e confiança, fornecendo também aos usuários interfaces extremamente fáceis, divertidas e intuitivas de se utilizar. É um sistema operacional avançado para dispositivos móveis, sendo cada vez mais procurado e desejado pelos usuários. (APPLE, 2013).

1.1.2 Windows Phone

Windows Phone é um dos mais recentes sistemas operacionais móveis lançados cuja finalidade é disputar o mercado da mobilidade com o iOS da *Apple* e *Android* da *Google*. Lançado em 21 de outubro de 2010 pela *Microsoft*, o *Windows Phone* diferentemente do que

os usuários pensaram, não segue a linha do *Windows Mobile*, se trata de um sistema operacional totalmente modernizado, principalmente no quesito interface. (SAFATLI, 2013). Segundo SAFATLI, 2013, algumas das características do *Windows Phone* são:

- Foco no usuário final;
- Padronização de hardware junto às OEMs;
- Foco em interfaces ricas;
- Reuso de conhecimento;
- Experiências Integradas – HUBS;

A*Windows Store* é a loja de aplicativos do *Windows Phone* e possuí mais de 120.000 mil aplicativos e jogos autenticados. Para publicar um aplicativo dentro da *Windows Store* é necessária uma avaliação rigorosa feita pela *Microsoft* garantindo assim ao usuário que o aplicativo não causará nenhum dano em seu aparelho e ao desenvolvedor que a aplicação irá rodar em qualquer *Windows Phone* do mundo. (ZIEGLER,2010).

1.1.2.1 UI Windows Store

"UI Windows Store" é fundamentado nos princípios clássicos suíços de design gráfico, e é codinome da nova interface desenvolvida pela *Microsoft* para *Windows* 8. *UI Windows Store* tem uma linguagem de design baseado em tipografia, originalmente criado para uso no *Windows Phone*. (SAFATLI, 2013).

1.1.2.2 Live Tiles

Planos de cor *"Live Tiles"* foram inseridos na linguagem de design durante os estudos do *Windows Phone* primitivo. A *Microsoft* começou a unificar estes elementos da linguagem de design em seus outros produtos, com influência direta sendo visto nas últimas versões do *Windows Live Messenger*, *Live Mesh* e *Windows* 8. (SAFATLI, 2013).

1.1.3 Android

O *Android* é um sistema operacional para dispositivos móveis, desenvolvido e comercializado pela *Google* e que atualmente está entre os mais populares e utilizados

sistemas operacionais móveis do mundo. Suas características como *open source*, baixo custo e diversidade de aplicações, atraem cada vez mais usuários e desenvolvedores. (RASMUSSEN, 2011).

O sistema é baseado no núcleo do *Linux* dando base a criações de aplicações Java através de inúmeras bibliotecas e serviços oferecidos pela *Google*. (RASMUSSEN, 2011). Através de sua flexibilidade, o *Android* permite a troca de aplicativos padrões do sistema por aplicativos customizados onde poderão se comunicar com as demais aplicações padrões do sistema ou não. Segundo RASMUSSEN, 2011, algumas das características comuns do *Android* são:

- S.O totalmente *free*;
- Open Source;
- Pode ser utilizado em diferentes equipamentos e com hardware distintos;
- Utiliza *Kernel Linux*. Uma versão personalizada voltada para dispositivos móveis;
- Todas as aplicações são criadas sob a linguagem de Programação Java;
- Possui grande quantidade de aplicativos em sua loja virtual, tanto gratuitos como pagos;

As características específicas do *Android* são:

- *Handset Layouts*: A plataforma é adequada para equipamentos VGA maiores, gráficos 2D, bibliotecas gráficas 3D baseadas em *OpenGL* ES, especificação 2.0 e os layouts mais tradicionais de smartphones;
- Armazenamento: Utiliza o *SQLite* como principal ferramenta para armazenamento de dados;
- Mensagens: Utiliza o SMS e MMS como mecanismos para envio de mensagens;
- Navegador: O navegador usado é fundamentado no *framework open source* conhecido como *WebKit*;
- Máquina virtual *Dalvik*: Aplicações escritas em Java são compiladas e executadas usando a Máquina Virtual *Dalvik*;
- Multimídia: O sistema suporta formatos de áudio e vídeo como: MPEG-4, H.264, MP3 e AAC;

- Suporte Adicional de *Hardware*: O *Android* é totalmente capaz de utilizar câmeras de vídeo, *touchscreen*, GPS, e aceleração de gráficos 3D;

1.1.3.1 Estrutura do Android

O *Android* contém uma estrutura composta e dividida por camadas onde cada processo é controlado e monitorado por sua respectiva camada. Segundo SHANKLAND, 2007, as camadas são divididas em Aplicações, *Framework*, Bibliotecas, *AndroidRuntime* e *Kernel Linux*.

- **Aplicações**

Camada onde são desenvolvidos os códigos Java das aplicações e que contém de todos os aplicativos executados sobre o *Android*. Dentre os aplicativos, podemos destacar: e-mail, despertador, calendários, jogos, mapas, *browser* e internet.

- **Framework**

O objetivo da camada de *framework* é simplificar e facilitar a reutilização de procedimentos por parte do desenvolvedor ocultando toda a complexidade na criação das aplicações e também funcionando como uma conexão com a camada de bibliotecas do sistema operacional que serão utilizadas por meio de APIs localizadas no *framework*.

- **Bibliotecas**

A camada de bibliotecas como o próprio nome diz é a camada que contém todas as bibliotecas que são utilizadas para o funcionamento do sistema operacional, e, além disso, contém também bibliotecas que gerenciam recursos de multimídia, e funções para várias tarefas como: manipulação de navegadores web, manipulação de áudio e vídeo, aceleradores em nível de *hardware*, consultas de camadas 2D e 3D, acesso ao *SQLite*, banco de dados do *Android*, etc.

- **AndroidRuntime**

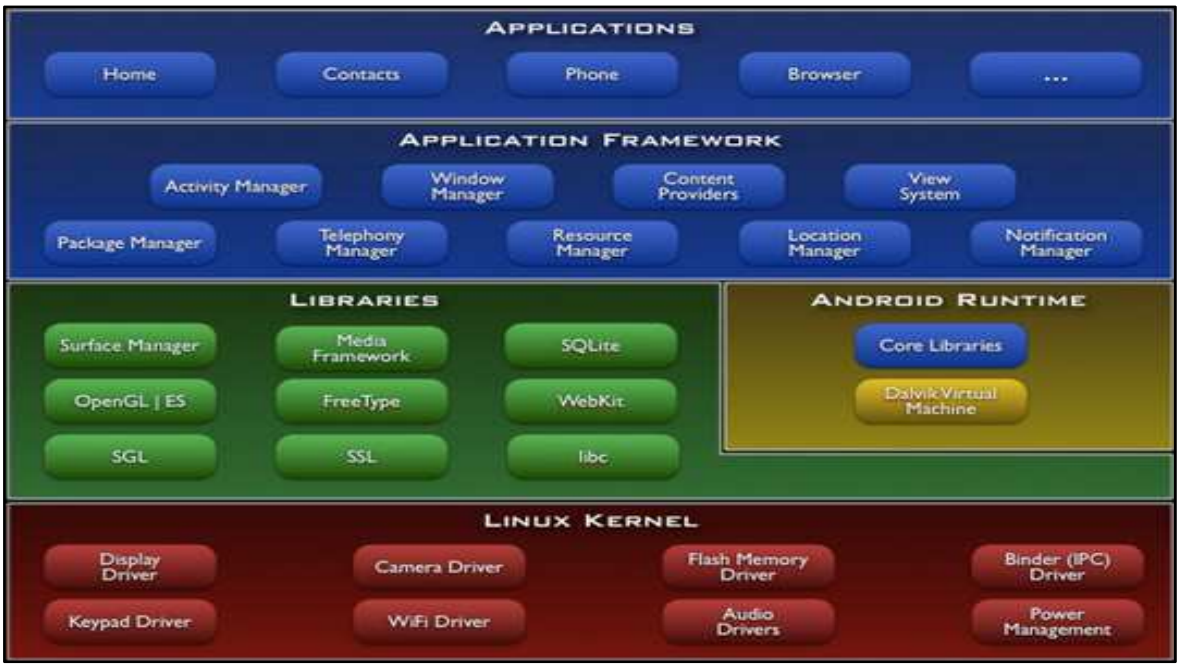
Apesar das aplicações para o *Android* serem desenvolvidas sob linguagem de programação Java, elas não são executadas por meio de uma tradicional máquina virtual Java e sim através da Máquina virtual *Dalvik* instanciada na camada *Runtime*. Para cada aplicação a ser executada no *Android* é gerada uma máquina virtual *Dalvik*, que tem por características seu excelente desempenho, compatibilidade com os novos *hardwares* lançados e, criada com a função de executar processos simultaneamente.

- **Kernel Linux**

A camada do *Kernel* pode ser considerada como a principal camada da arquitetura do *Android*. Responsável por grande parte dos serviços, a camada do *kernel* também tem como tarefa intermediar todos os aplicativos do *Android* com o *hardware* além de gerenciar e controlar o uso da memória e de *threads* por exemplo. O controle de processos, protocolos de rede, segurança dos arquivos e modelo de *drivers* também são tarefas realizadas por esta camada.

A figura 2 ilustra a estrutura do Sistema Operacional *Android*.

Figura 2 - Estrutura Android



Fonte: http://www.denis.net.br/android/images/android/

1.1.3.2 Componentes do Android

Toda aplicação criada para o *Android* é composta por componentes ou grupos de construção fundamentais na qual o sistema poderá se interagir. Cada componente existente contém suas próprias regras e identidade únicas na qual cooperam para definir o procedimento total de cada aplicação, porém alguns deles são dependentes um dos outros e nem sempre se caracterizam por serem pontos de entrada para usuários. (FRANÇA, 2007).

Existem quatro diferentes tipos de componentes de aplicação onde cada um deles é utilizado para uma finalidade diferente e cujo ciclo de vida de cada um determina como o componente é criado. (FRANÇA, 2007). De acordo com FRANÇA, 2007, os quatro tipos de componentes de aplicação são:

- **Atividade**

Uma atividade (*activity*) é o nome dado a uma interface de usuário, as chamadas "telas" de cada programa. Por exemplo, em um aplicativo de localização de clientes, será necessário uma *activity* para guardar os clientes, uma para cadastro de clientes e outra para alterações de clientes de forma individual. Embora as *activitys* trabalhem juntas em coerência e coesão nesse aplicativo de clientes, são independentes entre si. Dessa forma, pode-se iniciar qualquer uma dessas *activitys* por outras aplicações (Se a aplicação primária permitir). Por exemplo, uma aplicação de câmera pode iniciar a *activity* de cadastro de clientes, para inserir a imagem do cliente no banco de dados.

- **Serviço**

Diferente de uma *activity*, serviço (*service*) é um componente responsável por executar em segundo plano operações extremamente pesadas ou simplesmente processos remotos, e por si só, não apresenta interface de usuário. Por exemplo, um *service* é responsável por tocar uma lista de música enquanto o usuário navega na internet por outra aplicação, trabalhando em plano de fundo sem atrapalhar a interação do mesmo com a *activity* usada no momento.

- **Contentprovider**

Contentprovider é responsável por controlar grupos de dados compartilhados das aplicações e também realizar o armazenamento desses dados através de mecanismos de armazenamento existentes nos dispositivos móveis como, sistema de arquivos, banco de dados *SQlite* nativo do *Android*, *WebService*, ou qualquer forma persistente de armazenamento na qual as aplicações possam obter os acessos. Por meio do *contentprovider*, todas as aplicações que possuírem permissão podem realizar consultas e modificações nos dados de outras aplicações como, por exemplo, informações pessoais de usuários fornecidas pelo *Android* que podem ser consultadas através de parte do *contentprovider* para a leitura e escrita de informações complementares de usuários.

Contentproviders também são importantes quando se trata de leitura e escrita de dados restritos a própria aplicação e que não podem ser compartilhados, como por exemplo, o aplicativo *Note Pad* que usa um *contentprovider* para salvar notas.

- **Broadcast receivers**

Broadcast receiver é o componente que corresponde a anúncios e alertas que são apresentados para todo o sistema, como por exemplo, *broadcast* alertando a falta de bateria ou de atualizações disponíveis. Grande parte dos *broadcasts* são criados a partir do sistema operacional, porém algumas aplicações também possuem funções capazes de dar início a *broadcast* como por exemplo alertar a outras aplicações que dados estão disponíveis nos equipamentos para uso, um *broadcast receiver* não necessariamente precisa de uma interface para a execução dos alertas, elas podem realizar a criação de notificações no status bar do sistema para mostrar ao usuário que ocorreu algum evento *broadcast*, porém a função principal do *broadcast* é operar como um *gateway* intermediando entre os componentes e otimizar ao máximo o trabalho realizado.

1.2 Desafios do Android

Apesar das vastas vantagens que a mobilidade contém, é preciso evidenciar as imperfeições desses dispositivos, desde os recursos de *hardware* aos de *software* e interatividade. (RAMOS e DUARTE, 2007). De acordo com RAMOS e DUARTE, 2007, alguns dos desafios a serem vencidos de forma geral pelo sistema são:

- **Limitações de tela**

Outra limitação desses dispositivos é a dimensão de tela, como estes são pequenos, os desenvolvedores de aplicações têm de respeitar a pequena área de trabalho, não podendo ultrapassar as dimensões que ela exige em diferentes casos, já que existem muitos aparelhos que executam o mesmo sistema. Não dar a atenção devida a esses limites trará consigo o surgimento de barras de rolagem que tornarão a utilização do aplicativo nada ergonômica sendo às vezes até incômoda.

- **Limitações de software**

Já com relação ao *software* e suas funções, percebe-se que essas são limitadas às características funcionais do sistema operacional de cada aparelho móvel. Estes sistemas têm a responsabilidade de prover recursos multimídia, gráficos, manipulação de informações, dentre outras funções.

- **Expectativa de usuário**

Nesse universo de dispositivos móveis, é comum a execução dos aplicativos de forma extremamente rápida e isso acaba por gerar uma expectativa no usuário de que todo programa deve abrir instantaneamente, não importando quais sejam as funções do mesmo. Graças a essa intolerância por meio dos usuários, os desenvolvedores são influenciados consideravelmente muitas vezes tendo que optar por responder positivamente as expectativas do usuário sacrificando algumas das idéias primárias do seu projeto.

- **Limitações de bateria**

O sistema *Android* exige grande consumo de bateria, aparelhos celulares comuns que aguentavam até 72 horas sem alimentação de energia, hoje foram substituídos por *Smartphones* que raramente aguentam 24 horas sem alimentação. Isso traz problemas tanto aos usuários, quanto aos desenvolvedores que caso criem programas demasiadamente pesados, consumirão toda a energia do aparelho em pouco tempo, causando a frustração dos usuários.

CAPÍTULO 2 – TÉCNICAS E MECANISMOS DE ENTRADA E SAÍDA NO ANDROID

2.1 Mecanismos de Entrada e saída

O *Android* fornece aos desenvolvedores várias opções para se armazenar dados e informações de suas aplicações de forma que, tais soluções atentam às reais necessidades de armazenamento. A escolha do mecanismo de armazenamento é definida sob uma análise de cada aplicação visto que a mesma poderá ou não ter seus dados privados somente a sua aplicação e usuário. (GOOGLE, 2014).

Deve ser levado em conta também o espaço de armazenamento que os dados da aplicação irão precisar e se os mesmos serão temporários ou não. (GOOGLE, 2014).

2.1.1 SharedPreferences

SharedPreferences ou arquivo de Preferências é um mecanismo de armazenamento limitado a trabalhar com tipos de dados primitivos como boolean, int, long, string e float no formato de pares chave-valor onde todos os dados irão perdurar em todas as sessões da aplicação até mesmo após a mesma ser encerrada. (GOOGLE, 2014).

O uso de *sharedPreferences* é bastante viável e recomendável quando se devem salvar dados ou informações pequenas, únicas ou simples de cada aplicação como, por exemplo, o usuário e senha, configurações e preferências da aplicação.

Não é recomendado para guardar informações complexas e grandes que geralmente estão relacionadas ao uso de objetos, para isso é utilizado o *SQLite*. (GOOGLE, 2014).

2.1.2 InternalStorage

O *Android* permite que se armazenem dados e informações diretamente na memória interna do dispositivo, e por padrão, estes dados são restritos somente a sua própria aplicação não sendo possível o acesso e compartilhamento para as demais aplicações e usuários. Estes dados serão removidos da memória interna quando a aplicação for desinstalada do equipamento. (GOOGLE, 2014).

No *Android*, o desenvolvedor poderá também salvar seus dados de forma temporária

e não mais persistentemente com o uso de *Cache*. (GOOGLE, 2014).

Uma grande desvantagem na utilização dos *caches* é a segurança dos dados, isto porque no decorrer da utilização do dispositivo, caso o mesmo venha ficar com pouco espaço em memória interna, o *Android* poderá apagar os arquivos de *cachê* a fim de obter um maior espaço livre. (GOOGLE, 2014).

2.1.3 ExternalStorage

O armazenamento Externo no *Android* pode ser realizado através de mídia removível como o SDCard quanto uma mídia não removível localizada internamente no equipamento. Este armazenamento pode ser realizado com todo e qualquer tipo de equipamento compatível ao *Android* e as informações poderão ser compartilhadas entre si. (GOOGLE, 2014).

Os arquivos salvos no armazenamento externo não possuem restrição e podem ser lidos por qualquer aplicação, e também modificados pelo usuário do dispositivo ao ativar o armazenamento USB para realizar transferências de arquivos para um computador. (GOOGLE, 2014).

Uma grande restrição na utilização de Armazenamento Externo é que os arquivos salvos poderão ser a qualquer momento removido com a mídia externa e não existe bloqueio algum na remoção de mídia e também na segurança sobre os arquivos salvos. (GOOGLE, 2014).

Ao utilizar o armazenamento Externo, primeiramente é preciso verificar se a mídia está disponível através do método *getExternalStorageState().* Este método retorna o estado na qual a mídia se encontra, desta forma é possível saber se a mesma se encontra ausente, compartilhada ao computador ou até mesmo removida. (GOOGLE, 2014).

2.1.4 SQLiteDatabase

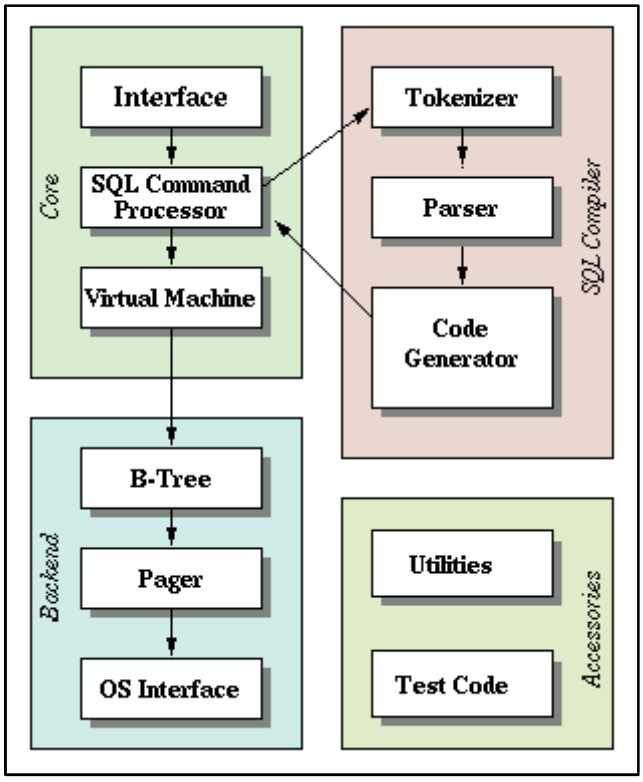
O *SQLite* é um mecanismo de banco de dados relacional leve, simples e eficiente para a persistência de dados que não necessita de processos separados no servidor. O *SQLite* permite com que os dados das aplicações possam ser gravados em tabelas e manipulados através de instruções SQL. Tais características tem tornado do *SQLite* uma ferramenta cada vez mais utilizada e desejada especialmente aos desenvolvedores de linguagens como PHP e C/C++. (SQLite, 2014).

O *SQLite* pode ser definido como uma biblioteca desenvolvida totalmente em C padrão (ANSI) podendo ser integrada a aplicações escritas em diversas linguagens de programação e inclusa ao PHP versão 5 podendo ser utilizada de forma muito simples a programas desenvolvidos em C e C++. (SQLite, 2014).

De forma prática e objetiva, o *SQLite* funciona como um “Mini SGBD” capaz de criar, ler e escrever um arquivo no disco do dispositivo móvel, este arquivo contém a extensão “db” e é capaz de manter uma grande quantidade de tabelas. (SQLite, 2014).

A criação e exclusão de tabelas no *SQLite* são feitas através dos comandos *CREATE TABLE* e *DROP TABLET* da linguagem SQL. Os dados das tabelas são manipulados através dos conhecidos comandos DML (*INSERT*, *UPDATE* e *DELETE*) e são consultados através do uso do comando *SELECT*. (SQLite, 2014). A figura 3 ilustra a estrutura do *SQLite.*

Figura 3– Estrutura Sqlite



Fonte: http://eupodiatamatando.com/2007/08/13/sqlite-resolvendo-problemas-simples-com-um-banco-simples/

2.1.4.1 ActiveAndroid

O *ActiveAndroid* é um framework fundamentado no padrão de projeto *ActiveRecord* apesar de possuir algumas divergências com relação a este padrão. Definido como uma

solução aberta, o *ActiveAndroid* tem como característica separar em módulos as funções de consulta. (PARDO, 2014).

Assim como um ORM, o *ActiveAndroid* tem como objetivo abstrair o acesso ao banco de dados *SQLite* contribuindo para um código fonte mais fácil e otimizado, permitindo a utilização de todas as funções do CRUD sem a necessidade de muita escrita de instruções SQL. Cada tabela do banco de dados é representada por classes e os campos são representados por variáveis de forma ordenada podendo ser manipulados através das funções *save()* e *delete()*. (PARDO, 2014).

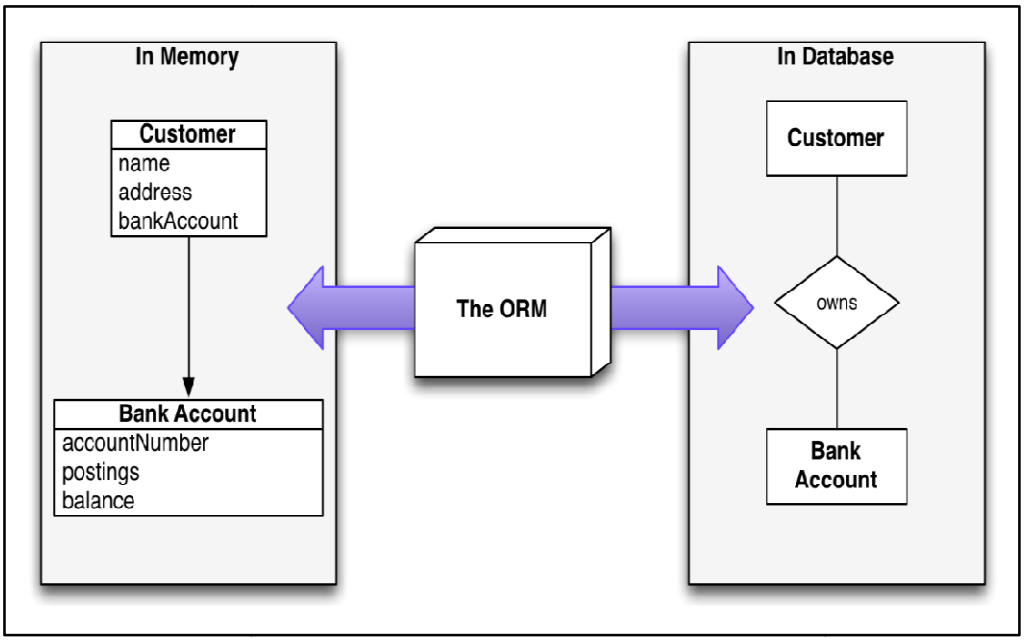
O *ActiveAndroid* é de fácil utilização e pode ser encontrado em repositórios públicos como o *GuitRub*. Como característica negativa, é possível afirmar que o *ActiveAndroid* utiliza mecanismos de anotações para definir informações como por exemplo nomes de campos e tabelas do banco na qual será realizado o mapeamento da classe, este recurso poderia ser feito de forma automática não sendo necessário a intervenção por parte do desenvolvedor. (PARDO, 2014).

Segundo PARDO, 2014, algumas das principais características do *ActiveAndroid* são:

- Configuração rápida e fácil;
- Fácil manutenção;
- Separação entre modelos de dados e objetos DAO;
- Pode ser utilizado em combinação com Json;
- Banco de dados pré-preenchidos;

Na figura 4 é ilustrado a arquitetura interna do *ActiveAndroid.*

Figura 4– ActiveAndroid



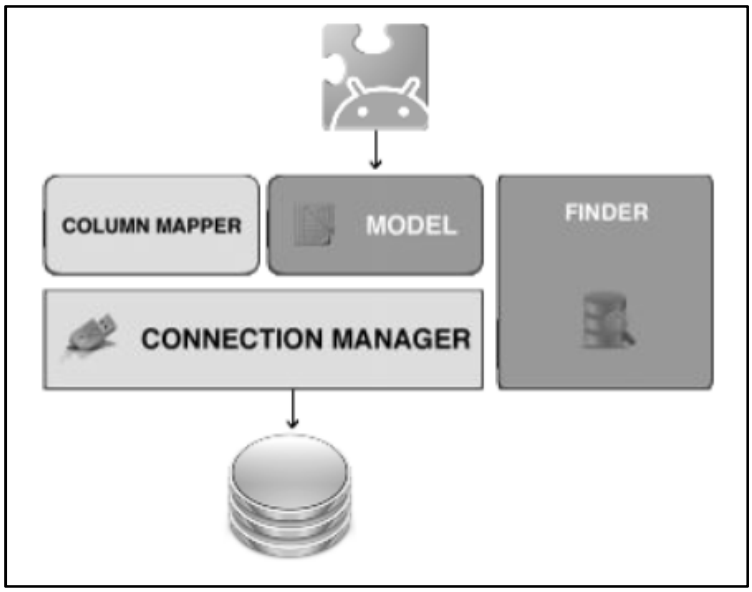
Fonte: https://github.com/codepath/android_guides/wiki/ActiveAndroid-Guide

2.1.4.2 AlienDroid

AlienDroid, lançado em 2013, tem seu funcionamento baseado também no *ActiveRecord*, trazendo a incorporação de leis que ajudem os objetos a permanecer nas classes de domínio do aplicativo. (SILVA, 2013). Essa visão, apesar de possuir uma forte oposição com relação à funcionalidade em um SO de grande porte, mostra-se gigantescamente útil quando o assunto é aplicativo para *Android*. Sabe-se que esses aplicativos tendem a ser compactos e com o menor escopo possível. Olhando por esse ângulo, o padrão *ActiveRecord* toma a frente pois não utiliza funções desnecessariamente complexas, que muitas vezes são indesejáveis. (SILVA, 2013).

Na figura 5 é ilustrado a Arquitetura interna do *AlienDroid.*

Figura 5- Arquitetura interna do AlienDroid



Fonte: http://www.lbd.dcc.ufmg.br/colecoes/sbsi/2013/0034.pdf

Observando a Figura 5, nota-se que o componente no caso chamado *Model*, será responsável por toda a comunicação desenvolvida entre *AlienDroid* e as funções que o iniciarão. Não necessariamente os programadores deverão ter grande noção acerca do projeto interno, apenas a interface encontrada através desse componente que uni todos os dados relevantes para executar o mapeamento de elementos para um modelo de mesmo peso no *SQLite*. (SILVA, 2013).

Nesta arquitetura, diferencia-se principalmente por sua simplicidade. Como citado acima, guarda-se no *Model* a maior parte do código determinado a interagir com o *SQLite*. Essa simplicidade é causada propositalmente com o objetivo de causar menos sobrecarga para os programas que se basearão no *AlienDroid*. Pode-se enfatizar, também, que o *framework* utiliza o recurso de reflexão apenas em alguns momentos, apenas para caracterizar os números representantes dos atributos dos elementos. Mesmo assim, apesar da grande funcionalidade a favor de grande parte dos *frameworks*, a reflexão atrapalha o desempenho de programas escritos nesta linguagem. (SILVA, 2013).

A função de notas também não foi inclusa no *AlienDroid*. Seu uso, apesar de facilitar o ato de codificar, acarreta na sobrecarga extra para o *framework* consequente da reflexão, que para ter êxito na leitura dos dados obtidos pelas notas, torna-se preciso recorrer a esta característica da linguagem. (SILVA, 2013).

2.1.5 WebService

O *WebService* foi projetado e criado com a finalidade de estabelecer a comunicação entre aplicações criadas por linguagens de programação distintas umas das outras através de serviços que são apresentados afim de que outros sistemas consigam acessá-los, de forma simplificada, *WebServices* tem por principal objetivo a comunicação entre aplicações por meio da internet. (SADAGI, 2013).

Com a utilização de *WebServices*, o desenvolvimento de aplicações móveis tem se tornado mais eficiente e simples, pois, toda a lógica complexa de desenvolvimento fica a cargo do serviço web. Esta técnica é de extrema importância, pois, se tratando de dispositivos móveis que apesar de toda a evolução ainda possuem limitações no quesito recursos de *hardware*, é possível diminuir execuções de serviços por parte dos dispositivos e adicioná-los aos serviços web como mencionado acima tornando as aplicações mais leves e capazes de interagir com demais aplicações. (SADAGI, 2013).

- **SOAP**

O Soap fundamenta-se em uma requisição remota de uma função e para isso precisa definir o diretório do componente, a denominação do método e os contextos para esse método. As informações são organizadas em XML com regras específicas e transmitidas normalmente via HTTP. (SADAGI, 2013).

O Soap não estabelece como regra algum tipo de semântica tanto para padrão de programação quanto para implementação. Esta característica é de grande importância, pois aceita com que os serviços ou clientes constituam de aplicações criadas por distintas linguagens de programação. (SADAGI, 2013).

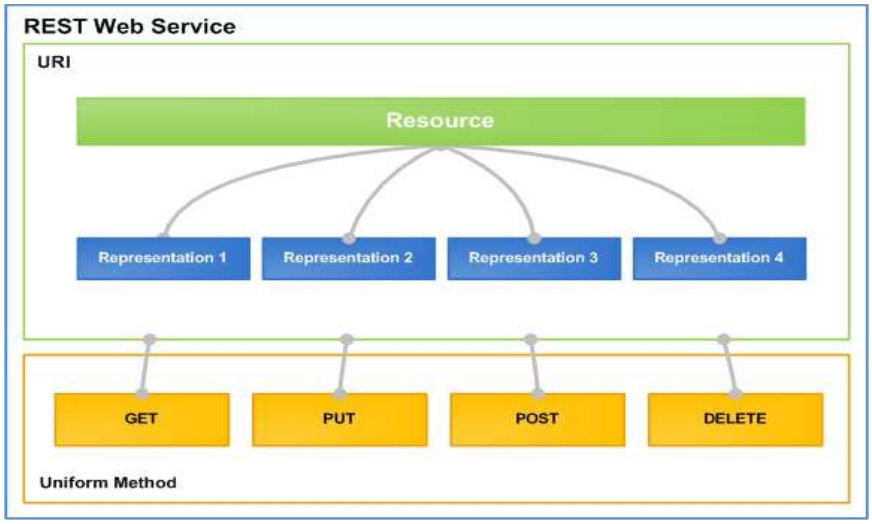
O WSDL apresenta os serviços oferecidos à rede por meio de uma semântica XML e fornece o manual necessário para a chamada do sistema distribuído e a metodologia necessária para o estabelecimento da comunicação. O SOAP estabelece a comunicação do cliente com o servidor e o WSDL apresenta os serviços disponibilizados. (SADAGI, 2013).

- **REST**

REST é um método de construção de *software* para sistemas operacionais desenvolvidos por Roy Fielding. O objetivo do desenvolvimento do REST é tornar as funções simples e abertas a qualquer pessoa que queira usá-las. E esta pessoa poderá utilizar qualquer plataforma disponível, tornando mais fácil consequentemente o trabalho dos programadores, já que utilizarão de uma linguagem mais simples, possibilitando a implantação em qualquer servidor que suportem HTTP ou HTTPS. (SADAGI, 2013).

É também uma forma de acoplar um formato de arquitetura de dificuldade elevada para codificar um *software* em que os usuários possam realizar requisições de funções. Sua finalidade é que, ao contrário da utilização de métodos mais complicados como, por exemplo, o SOAP, utilizado para estabelecer a comunicação entre usuário e servidor, seja usado o método HTTP, que por sua vez é bem mais simples para executar essas requisições. A solicitação de uma função REST pode ser executada de forma síncrona ou assíncrona. SOs pautados nas características do método REST apresentam-se como RESTful. É fundamental que se evidencie que em um *WebService* RESTful o primordial são as URLs da função e os *resources* (recurso, entidade, dados apresentados através de XML). De forma gera, a URL utilizada para ter acesso a essa função nunca se alterará, contudo se for editado o método HTTP (*GET*, *POST*, *PUT* e *DELETE*) o retorno da solicitação será divergente do esperado. (SADAGI, 2013). Na figura 6 é ilustrada a estrutura do REST.

Figura 6 - REST Web service design structure



Fonte: http://wink.apache.org/1.0/html/1%20Introduction%20to%20Apache%20Wink.html

2.1.6 Volley

Volley é o nome dado a uma biblioteca que trabalha com a execução e armazenamento em memória *(cache)* de recorrências a rede, poupando os programadores de ter de repetir o mesmo código em todas as funções do projeto, reduzindo as chances de erros cometidos pelo programador decorrentes de falta de atenção, exaustão, etc. Baseando-se nisso, a *Google* desenvolveu *Volley*. (KIRKPATRICK, 2013).

Essa biblioteca auxilia a realização de solicitações, o tratamento de respostas, o trabalho com filas de requisições, assuntos prioritários e também traz consigo memória disponível para guardar comandos e dados disponibilizando-os de forma mais rápida e eficiente nas aplicações. (KIRKPATRICK, 2013).

Na *Volley*, toda a interação da web e até mesmo o uso de memória é a partir de solicitações, e o mais importante meio de fazer isso é utilizando uma fila de requisições *(RequestQueue).* Utilizada em todos os momentos do aplicativo para trazer ao centro do processamento toda a linha de comunicação, essa fila permite também até a priorização de elementos e requisições dentro dela mesma. (KIRKPATRICK, 2013). Ainda sobre *Volley*, segundo KIRKPATRICK, 2013, é possível citar como principais vantagens:

- ✓ Gerência automaticamente as requisições;
- ✓ Memória em disco (Cache);
- ✓ Possibilidade de cancelar uma requisição;
- ✓ Customização;
- ✓ Comunicação paralela com possibilidade de priorização;
- ✓ Fornece ferramentas de depuração e rastreamento;

2.1.7 AsyncTask

O *AsyncTask* é uma classe abstrata objetivada na realização de requisições em uma *Thread* não ligada ao *background* do *Android* permitindo que a principal *Thread* permaneça disponível para continuar funcionando suas aplicações. Assim, faz-se possível a realização de requisições sem oferecer qualquer risco negativo ao desempenho do SO. O uso do *AsyncTask* tem como situações alvo, funções rápidas, que levam poucos segundos para serem processadas. Se, por algum motivo, for preciso realizar funções mais complexas e consequentemente mais longas com relação ao tempo, aconselha-se o uso das opções

APIsdisponíveis no pacote *"Java.util.concurrent"*. (CARVALHO, 2014).

No *AsyncTask,* os principais métodos são:

- **onPreExecute()**

Disponível como primeiro método requisitado assim que executa-se o *execute(Params...).* Este método é usado, junto a outras funções, para configurar processos e/ou simplesmente acionar algum *StatusLoader* acusando ao usuário que há um processo acontecendo no momento. (CARVALHO, 2014).

- **doInBackground(Params..)**

Disponível como segundo método requisitado logo após o método *onPreExecute()* ser finalizado. É onde a função será feita em uma *Thread* separada da *Thread* Principal. Podemos também aqui requisitar a função *publishProgress(Progress...)* passando o número de progresso do seu processo, que quando solicitado, executará o*onProgressUpdate(Progress...)* transmitindo esse valor passado no *publishProgress(Progress...).* Aconselha-se também que, na função, torne-se válido sem exceção o uso do método *isCancelled()* (Esta função devolverá *TRUE* se requisitada a função *cancel()*) para que, se ocorrer de retornar *TRUE,* possibilite ao usuário parar o processamento da função. (CARVALHO, 2014).

- **onProgressUpdate(Progress..)**

Já este método será solicitado assim que a função *publishProgress(Progress...)* for ativada. Tem como objetivo trazer um status do progresso do processo. Nesse momento, é possível atualizar o status de algum tipo de medidor de progresso, por exemplo. Essa função é também independente de timeout, já que o método *doInBackground(Params...)* se mantém processando em outra *Thread*, não sofrendo interferência pelo processamento da função *onProgressUpdate(Progress...).* (CARVALHO, 2014).

- **onPostExecute(Result)**

Essa função é solicitada ao sistema assim que é realizado o encerramento do método

onInBackground(Params...). Ela recebe o resultado da *onInBackground(Params...)* através do tipo organizado como resultado na configuração da Classe em questão. (CARVALHO, 2014).

2.1.8 JSON

JSON pode ser definido como um padrão em comunicação e armazenamento de dados via texto e tem sido muito usado por aplicações web devido a sua disposição de organizar dados e informações de uma maneira compacta sendo melhor até do que o padrão XML, o que torna mais eficiente e rápida a análise dessas informações. Tais qualidades comprovam o motivo do JSON ter se tornado o padrão utilizado por empresas conceituadas como *Google* e *Yahoo* que possuem grandes quantidades de dados a serem transmitidos por suas aplicações.

A representação das informações no JSON é feita de forma objetiva e fácil, onde é atribuído um nome para todo valor representado que apresenta seu significado. Essa Técnica é proveniente da técnica usada pelo *JavaScript* na representação de seus dados.

CAPÍTULO 3 – IMPLEMENTAÇÃO E COMPARATIVOS ENTRE AS TÉCNICAS DE ENTRADA E SAIDA NO ANDROID

Considerando a crescente demanda por aplicativos para dispositivos móveis e a forte necessidade de desenvolvê-los em tempo hábil e com boa qualidade, diversas bibliotecas e *frameworks* foram propostos aos desenvolvedores a fim de facilitar no desenvolvimento das aplicações, otimizar códigos fontes sem a mescla de regras de negócio com acesso ao banco de dados e também aumentar a produtividade. Com isto tem sido possível a criação de aplicações eficazes e com códigos fáceis de serem compreendidos. (RASMUSSEN, 2011).

Diante do exposto, o objetivo deste capítulo é propor boas práticas para o processo de entrada e saída de dados em aplicações *Android* através de ferramentas que visam facilitar e melhorar o desenvolvimento das aplicações.

3.1 Técnicas de implementação para persistência em banco de dados

A seguir serão apresentadas algumas ferramentas que tem por finalidade auxiliar os desenvolvedores a trabalharem com persistência de dados em banco de dados. Tais ferramentas podem ser definidas como frameworks ORM para o *Android*.

ORM – Mapeamento Objeto-Relacional

Um framework ORM tem como finalidade estabelecer o acesso banco de dados por meio da orientação a objetos, pois estabelecem uma conexão entre o modelo Relacional que é definido como banco de dados e orientado a objetos, definidos como classes da aplicação. A implementação de um *framework* ORM contribui na redução de escritas SQL criadas para a manipulação dos dados no banco, pois todas as consultas são criadas de forma automática através do *framework*, acarretando em aplicações com códigos otimizados e fáceis de serem compreendidos contribuindo para futuras manutenções. (YURI, 2010).

O uso de um *Framework* ORM para manter os dados de um aplicativo em um banco de informações torna mais fácil o desenvolvimento e ajuda a diminuir o tempo de fabricação de uma aplicação, devido ao fato de todo o trabalho de transformar os dados armazenados do banco em classes do servidor do aplicativo é executado pelo *Framework*. Dessa forma, é possível a criação de uma aplicação completamente orientada a objetos, não tendo demasiado

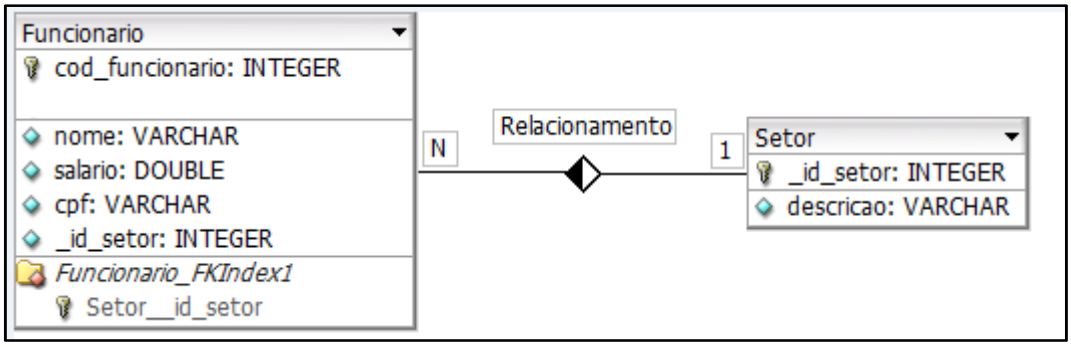
esforço, já que os desenvolvedores trabalham com dados, utilizando o mesmo formato de linguagem usado para o desenvolvimento da lógica de negócio do aplicativo. (YURI, 2010).

Diagrama E-R (Entidade-Relacionamento)

O Diagrama E-R é um padrão diagramático que apresenta o modelo de dados de um sistema com alto nível de abstração. Caracteriza-se por ser a principal representação gráfica do Modelo de Entidades e Relacionamentos. (EDUARDO, 2011).

Para as técnicas de entrada e saída no *Android* que serão apresentadas neste capítulo, foi utilizado o seguinte Diagrama E-R conforme ilustrado na figura 7:

Figura 7– Diagrama Entidade Relacionamento



Fonte: Próprio autor

3.1.1 Utilização de Componentes nativos

Na primeira técnica de entrada e saída dos dados no *Android* foram utilizados os componentes nativos para a criação das tabelas e inserção dos dados com base no Diagrama apresentado na figura 7.

Na figura 8 é ilustrada a forma de criação das tabelas de funcionário e setor no banco de dados e, também a forma de inserção dos primeiros dados nas tabelas através de instruções SQL dentro das classes da aplicação.

Figura 8– Método OnCreate com as funções de Criação e Inserção dos dados no banco

ListaOpenHelper.java

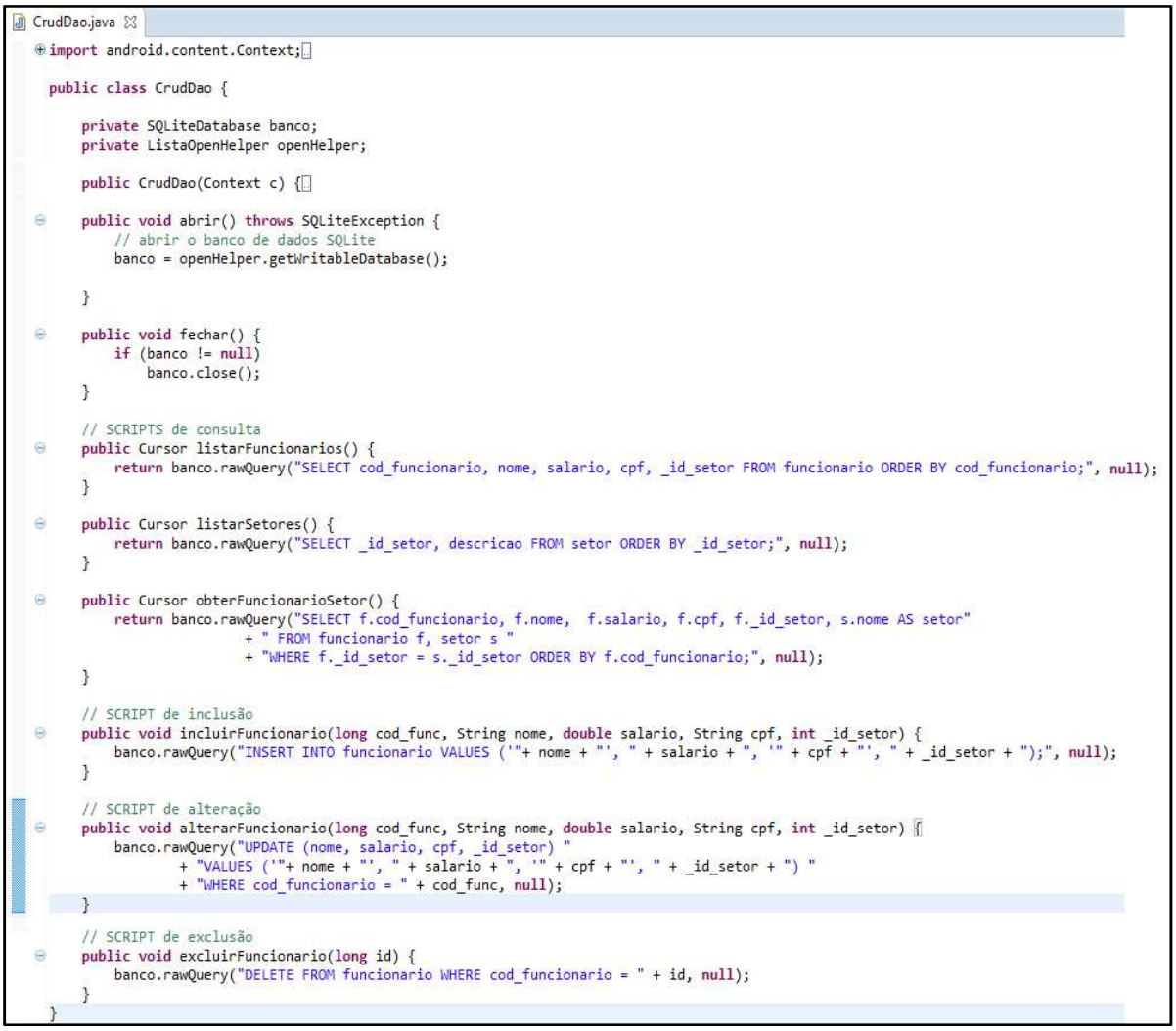
```
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.database.sqlite.SQLiteOpenHelper;

public class OpenHelper extends SQLiteOpenHelper {

    public OpenHelper(Context context, String name, CursorFactory factory,
            int version) {
        super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase banco) {
        banco.execSQL("CREATE TABLE funcionario (cod_funcionario INTEGER PRIMARY KEY AUTOINCREMENT, " +
                                                "nome TEXT, " +
                                                "salario NUMERIC, " +
                                                "cpf TEXT, " +
                                                "_id_setor INTEGER);");

        banco.execSQL("CREATE TABLE setor (_id_setor INTEGER PRIMARY KEY, descricao TEXT);");

        banco.execSQL("INSERT INTO funcionario(nome, salario, cpf, _id_setor) VALUES ('Joao', 2000.00, '5428754123', 3);");

        banco.execSQL("INSERT INTO setor(descricao, _id_setor) VALUES ('RH', 1);");
        banco.execSQL("INSERT INTO setor(descricao, _id_setor) VALUES ('Financeiro', 2);");
        banco.execSQL("INSERT INTO setor(descricao, _id_setor) VALUES ('Producao', 3);");
    }

    @Override
    public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2) {

    }
}
```

Fonte: Próprio autor

Para a entrada e saída dos dados foi criado a classe CrudDao. Nesta classe são apresentadas as formas básicas do CRUD para se utilizar e manipular os dados através de funções que contém as instruções SQL de consulta, atualização, inserção e exclusão. Assim como na figura 8, a classe CrudDao contém a mescla de código JAVA com instruções de acesso ao banco de dados, o que não é aconselhável, dificultando o entendimento e futuramente a manutenção da aplicação.

Na figura 9 é ilustrado a classe CrudDao.class com os métodos de manipulação do banco de dados.

Figura 9– Classe CrudDao com os métodos do CRUD.



```
CrudDao.java
import android.content.Context;

public class CrudDao {

    private SQLiteDatabase banco;
    private ListaOpenHelper openHelper;

    public CrudDao(Context c) {

    public void abrir() throws SQLiteException {
        // abrir o banco de dados SQLite
        banco = openHelper.getWritableDatabase();

    }

    public void fechar() {
        if (banco != null)
            banco.close();
    }

    // SCRIPTS de consulta
    public Cursor listarFuncionarios() {
        return banco.rawQuery("SELECT cod_funcionario, nome, salario, cpf, _id_setor FROM funcionario ORDER BY cod_funcionario;", null);
    }

    public Cursor listarSetores() {
        return banco.rawQuery("SELECT _id_setor, descricao FROM setor ORDER BY _id_setor;", null);
    }

    public Cursor obterFuncionarioSetor() {
        return banco.rawQuery("SELECT f.cod_funcionario, f.nome,  f.salario, f.cpf, f._id_setor, s.nome AS setor"
                        + " FROM funcionario f, setor s "
                        + "WHERE f._id_setor = s._id_setor ORDER BY f.cod_funcionario;", null);
    }

    // SCRIPT de inclusão
    public void incluirFuncionario(long cod_func, String nome, double salario, String cpf, int _id_setor) {
        banco.rawQuery("INSERT INTO funcionario VALUES ('"+ nome + "', " + salario + ", '" + cpf + "', " + _id_setor + ");", null);
    }

    // SCRIPT de alteração
    public void alterarFuncionario(long cod_func, String nome, double salario, String cpf, int _id_setor) {
        banco.rawQuery("UPDATE (nome, salario, cpf, _id_setor) "
                + "VALUES ('"+ nome + "', " + salario + ", '" + cpf + "', " + _id_setor + ") "
                + "WHERE cod_funcionario = " + cod_func, null);
    }

    // SCRIPT de exclusão
    public void excluirFuncionario(long id) {
        banco.rawQuery("DELETE FROM funcionario WHERE cod_funcionario = " + id, null);
    }
}
```

Fonte: Próprio autor

3.1.2 Utilização da Ferramenta ActiveAndroid

Dentre as principais e mais eficientes técnicas para gerenciamento de entrada e saída dos dados no *Android*, o *ActiveAndroid* se mostrou uma ferramenta excelente nos quesitos *performance*, otimização de código-fonte, redução da mescla de instruções SQL com regra de negócio e também de fácil utilização. Para o diagrama apresentado na figura 7 foram criadas as classes Funcionario.class e Setor.class que são representações das suas respectivas tabelas funcionario e setor no banco de dados.

Na figura 10 é ilustrado a classe Funcionario.class no *ActiveAndroid*.

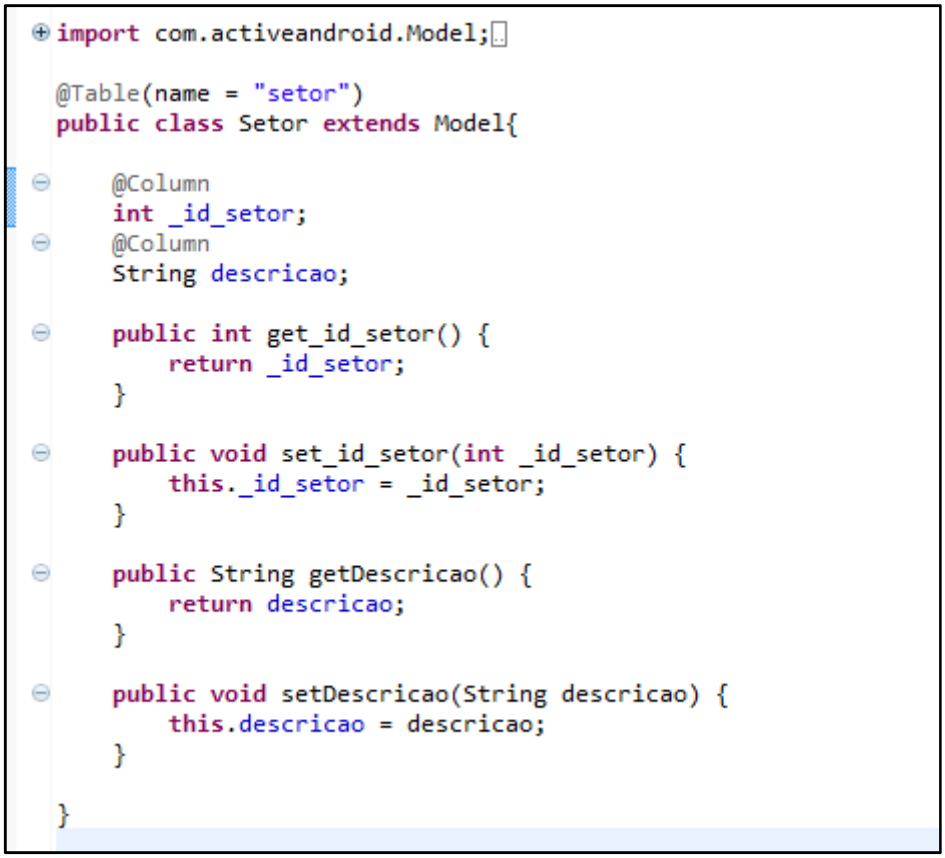
Figura 10- Classe Funcionario no ActiveAndroid.

```
@Table(name = "funcionario")
public class Funcionario extends Model {

    @Column(index = true)
    int cod_funcionario;
    @Column
    String nome;
    @Column
    double salario;
    @Column
    String cpf;
    @Column
    int _id_setor;

    public Funcionario() {
    }
    public Funcionario(int cod_funcionario, String nome, double salario, String cpf,
            int _id_setor) {
        this.cod_funcionario = cod_funcionario; this.nome = nome; this.salario = salario; this.cpf = cpf; this._id_setor = _id_setor;
    }

    public int getCod_funcionario() {   return cod_funcionario;     }

    public void setCod_funcionario(int cod_funcionario) {   this.cod_funcionario = cod_funcionario;     }

    public String getNome() {   return nome;    }

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

    public double getSalario() {    return salario;     }

    public void setSalario(double salario) {    this.salario = salario;     }

    public String getCpf() {    return cpf;     }

    public void setCpf(String cpf) {    this.cpf = cpf;     }

    public int get_id_setor() {     return _id_setor;   }

    public void set_id_setor(int _id_setor) {   this._id_setor = _id_setor;     }

    @Override
    public String toString() {
        return "Nome: " + nome + "\nSalário: " + salario + "\nCPF: " + cpf  + "\nCódigo do Setor: " + _id_setor;
    }
}
```

Fonte: Próprio autor.

Na figura 11 é ilustrado a classe Setor.class no *ActiveAndroid*.

Figura 11– Classe Setor no ActiveAndroid.



```
import com.activeandroid.Model;

@Table(name = "setor")
public class Setor extends Model{

    @Column
    int _id_setor;
    @Column
    String descricao;

    public int get_id_setor() {
        return _id_setor;
    }

    public void set_id_setor(int _id_setor) {
        this._id_setor = _id_setor;
    }

    public String getDescricao() {
        return descricao;
    }

    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

}
```

Fonte: Próprio autor.

Para criação do modelo de banco de dados no *ActiveAndroid*, é preciso apenas serem criadas classes denominadas com o nome das tabelas do banco de dados e variáveis que definem os campos para cada uma das colunas.

Alguns pré-requisitos que devem ser levados em conta na utilização do *ActiveAndroid* são que, as classes devem estender a classe *Model* e seus membros devem ser anotados usando *Column***.** O *ActiveAndroid* lida com tipos de dados primitivos, bem como as relações com outras tabelas e classes de data.

Uma coisa importante a salientar é que *ActiveAndroid* cria um campo id para as tabelas. Este campo é uma chave primária auto incremento.

Na figura 12 é ilustrado a classe ListaDao.class com os métodos para a manipulação dos dados através do CRUD utilizando o *ActiveAndroid*.

Figura 12– Classe DAO com os métodos do CRUD no ActiveAndroid.

```
import com.activeandroid.query.Select;

public class ListaDao {

    // SCRIPTS de consulta
    public static List<Funcionario> listarFuncionarios() {
        return new Select().from(Funcionario.class).orderBy("cod_funcionario")
                .execute();
    }

    public static List<Setor> listarSetores() {
        return new Select().from(Setor.class).orderBy("_id_setor").execute();
    }

    // Utilizando relacionamento
    public Funcionario obterFuncionarioSetor() {
        return (Funcionario) new Select().from(Funcionario.class).join(Setor.class)
                .on("Funcionario._id_setor = Setor._id_setor").execute();
    }

    // SCRIPT de inclusão
    public void incluirFuncionario(int cod_func, String nome, double salario,
            String cpf, int _id_setor) {
        Funcionario funcionario = new Funcionario(cod_func, nome, salario, cpf,
                _id_setor);
        funcionario.save();
    }

    // SCRIPT de alteração
    public void alterarFuncionario(int cod_func, String nome, double salario,
            String cpf, int _id_setor) {
        Funcionario funcionario = Funcionario.load(Funcionario.class, cod_func);
        funcionario.cod_funcionario = cod_func;
        funcionario.nome = nome;
        funcionario.salario = salario;
        funcionario.cpf = cpf;
        funcionario._id_setor = _id_setor;
        funcionario.save();
    }

    // SCRIPT de exclusão
    public void excluirFuncionario(int cod_func) {
        Funcionario funcionario = Funcionario.load(Funcionario.class, cod_func);
        funcionario.delete();
    }
}
```

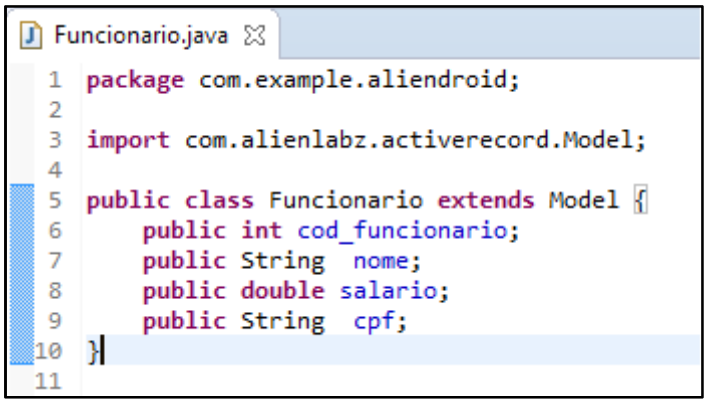
Fonte: Próprio autor

3.1.3 Utilização da Ferramenta AlienDroid

Para o diagrama apresentado na figura 7 foram criadas as classes Funcionario.class e Setor.class que são representações das suas respectivas tabelas funcionario e setor no banco de dados.

Na figura 13 é ilustrado a criação da classe Funcionado.class utilizando o *AlienDroid*.

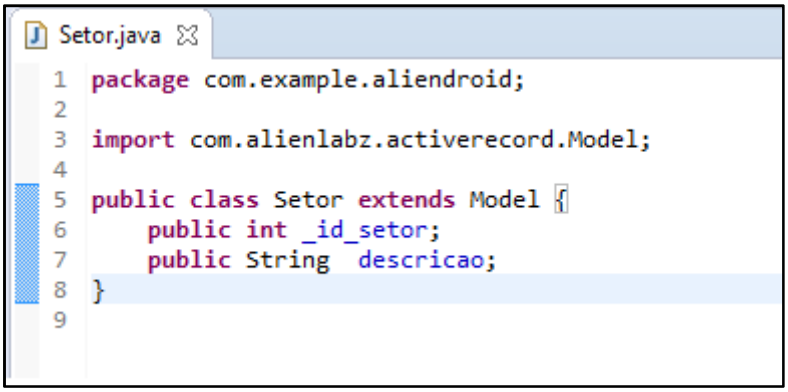
Figura 13- Classe Funcionario no AlienDroid.



Fonte: Próprio autor

Na figura 14 é ilustrado a criação da classe Setor.class utilizando o *AlienDroid*.

Figura 14- Classe Setor no AlienDroid.



Fonte: Próprio autor

O *AlienDroid* demonstrou ser uma ferramenta extremamente simples e prática na modelagem do banco de dados. Assim como no *ActiveAndroid*, as tabelas do banco de dados são representadas por classes, porém como diferencial, no *AlienDroid* não é preciso a criação de anotações para a definição dos campos e tabelas do banco na qual será realizado o mapeamento da classe, tornando a codificação simples e elegante.

Com a utilização do *AlienDroid*, foi possível verificar a carência da ferramenta na tentativa de manipular os dados utilizando os métodos do CRUD, a ferramenta ofereceu pouquíssimos recursos na tentativa de realizar relacionamentos entre as tabelas do banco.

Na figura 15 é ilustrado a classe CrudDao.class para a consulta dos dados utilizando o *AlienDroid*.

Figura 15- Classe DAO com os métodos de Consulta no AlienDroid

CrudDao.java

```
1  package com.example.aliendroid;
2
3  import java.util.List;
6
7  public class CrudDao {
8
9      // Busca por todos os funcionários
10     public static List<Funcionario> listarFuncionarios() {
11
12         return Model.findAll(Funcionario.class);
13     }
14
15     // Busca por um funcionário específico
16     public List<Funcionario> buscaFuncionario(final int codFunc) {
17
18         return Model.where(Funcionario.class, "cod_funcionario = ", new String[] {String.valueOf(codFunc)});
19     }
20
21     // Busca por todos os Setores
22     public static List<Setor> listarSetores() {
23
24         return Model.findAll(Setor.class);
25     }
26
27     // Busca por um setor específico
28     public List<Setor> buscaSetor(final int idSetor) {
29
30         return Model.where(Setor.class, "_id_setor = ", new String[] {String.valueOf(idSetor)});
31     }
32 }
33
```

Fonte: Próprio autor

3.2 Técnicas de implementação para consumir WebService

Neste tópico foram desenvolvidos dois cenários comparando duas formas de implementação, sendo que, no primeiro cenário foi implementado somente carregamento de imagens, e no segundo cenário foi implementado o carregamento de arquivos JSON.

3.2.1 Consumindo imagens (REST)

A seguir serão apresentadas técnicas para o carregamento de imagens nos dispositivos móveis por meio de implementação manual através de componentes nativos do *Android* e através da biblioteca *Picasso* que será detalhada na sequência do capítulo.

3.2.1.1 Utilização dos componentes nativos

Para o carregamento de imagens através dos componentes nativos do *Android*, foram utilizados os métodos do *AsyncTask*. Na figura 16 é ilustrado a classe CarregarImagemUrl.class utilizando *AsyncTask*.

Figura 16- Classe CarregarImagemUrl utilizando AsyncTask

CarregarImagemUrl.java

```
1  package com.example.imageviewandroid;
2
3  import java.io.InputStream;
18
19 public class CarregarImagemUrl extends Activity {
20     private EditText edtUrl;
21
22     @Override
23     protected void onCreate(final Bundle savedInstanceState) {
24         super.onCreate(savedInstanceState);
25         setContentView(R.layout.tela_url_img);
26         edtUrl = (EditText)findViewById(R.id.editText1);
27
28         // Obtendo a ultima URL digitada
29         final SharedPreferences preferencias = getSharedPreferences( "configuracao", MODE_PRIVATE);
30         final String url = preferencias.getString("url_imagem", "http://");
31         edtUrl.setText(url);
32     }
33
34     @Override
35     protected void onDestroy() {
36         super.onDestroy(); // Salva a URL para utiliza-la quando essa tela for re-aberta
37         final SharedPreferences preferencias = getSharedPreferences( "configuracao", MODE_PRIVATE);
38         final SharedPreferences.Editor editor = preferencias.edit();
39         editor.putString("url_imagem", edtUrl.getText().toString());
40         editor.commit();
41     }
42
43     public void baixarImagemClick(final View v){
44         new DownloadImagemAsyncTask().execute( edtUrl.getText().toString());
45     }
46
47     class DownloadImagemAsyncTask extends AsyncTask<String, Void, Bitmap>{
48         ProgressDialog dialog;
49
50         @Override protected void onPreExecute() {
51             super.onPreExecute();
52             dialog = ProgressDialog.show( CarregarImagemUrl.this, "Aguarde", "Carregando a imagem...");
53         }
54
55         @Override
56         protected Bitmap doInBackground(final String... params) {
57             final String urlString = params[0];
58             try {
59                 final URL url = new URL(urlString);
60                 final HttpURLConnection conexao = (HttpURLConnection) url.openConnection();
61                 conexao.setRequestMethod("GET");
62                 conexao.setDoInput(true);
63                 conexao.connect();
64                 final InputStream is = conexao.getInputStream();
65                 final Bitmap imagem = BitmapFactory.decodeStream(is);
66                 return imagem;
67             }
68             catch (final Exception e) {
69                 e.printStackTrace();
70             } return null;
71         }
72
73         @Override
74         protected void onPostExecute(final Bitmap result)
75         {
76             super.onPostExecute(result);
77             dialog.dismiss();
78             if (result != null){
79                 final ImageView img = (ImageView)findViewById(R.id.imageView1);
80                 img.setImageBitmap(result);
81             }
82             else {
83                 final AlertDialog.Builder builder =
84                         new AlertDialog.Builder(CarregarImagemUrl.this).setTitle("Erro").
85                         setMessage("Não foi possivel carregar imagem, tente novamente mais tarde!").
86                         setPositiveButton("OK", null); builder.create().show();
87             }
88         }
89     }
90 }
```

Fonte: Próprio autor

3.2.1.2 Utilização da API PICASSO

Quando se trata de *download* de Imagens, a Biblioteca *Picasso* se torna uma excelente alternativa para o desenvolvedor, pois, realiza todo o gerenciamento e controle de download e exibição da imagem, evitando problemas que comumente os desenvolvedores encontram ao implementar esses tipos de recursos. (PICASSO, 2014).

Na figura 17 é ilustrado a classe MainActivity.class utilizando a *Picasso*.

Figura 17- Utilização da Biblioteca Picasso

MainActivity.java

```
1  package br.picasso;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5  import android.widget.ImageView;
6  import com.example.picasso.R;
7  import com.squareup.picasso.Picasso;
8
9  public class MainActivity extends Activity {
10
11     ImageView imagem;
12
13     @Override
14     protected void onCreate(final Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.main);
17
18         imagem = (ImageView)findViewById(R.id.imagem);
19
20         final String url = "http://www.papeisdeparedehd.com/walls/carro_negro_pela_noite-wide.jpg";
21
22         Picasso.with(getApplicationContext()).load(url)
23         .placeholder(R.drawable.ic_launcher)
24         .error(R.drawable.ic_launcher).into(imagem);
25
26     }
27 }
28
```

Fonte: Próprio autor

3.2.2 Consumindo JSON (REST)

Para a utilização do Padrão JSON, foi utilizado uma ferramenta *apiary.io* que permite a criação da estrutura JSON a ser utilizada para testes. Não será apresentado no presente trabalho o conceito de ferramentas que auxiliam testes dos comportamentos de objetos, pois não é relevante para o mesmo.

Na figura 18 é ilustrado um exemplo de estrutura JSON no *apiary.io.*

Figura 18- Estrutura do Json na Apiary.io

Fonte: Próprio autor

3.2.2.1 Implementação manual

Para consumir a estrutura JSON definida no exemplo da figura 18, foi necessário a criação da classe DownloadCarros.class e da classe RestClient.class responsável por estabelecer a conexão com o *WebService*.

Na figura 19 é ilustrado a classe DownloadCarros.class utilizando JSON para consumir o *WebService*.

Figura 19- Utilização do JSON

```
class DownloadCarros extends AsyncTask<String, String, String> {

    @SuppressWarnings("deprecation")
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        showDialog(progress_bar);
    }

    @Override
    protected String doInBackground(final String... args) {
        final String jsonRes = RestClient.get(args[0]);
        try {
            final JSONObject jsonObject = new JSONObject(jsonRes);
            final JSONArray jsonResposta = jsonObject
                    .getJSONArray("carros_exemplo");

            _listaCarros = new ArrayList<Carro>();

            for (int i = 0; i < jsonResposta.length(); i++) {
                final JSONObject carroObj = jsonResposta.getJSONObject(i);

                final Carro carro = new Carro();
                carro.setNome(carroObj.getString("nome"));
                carro.setImagem(carroObj.getString("imagem"));
                carro.setDescricao(carroObj.getString("descricao"));
                carro.setValor(carroObj.getString("valor"));

                _listaCarros.add(carro);
            }
        } catch (final JSONException e) {
            e.printStackTrace();
        } catch (final ParseException e) {
            e.printStackTrace();
        } catch (final Exception e) {
            e.printStackTrace();
        }
        return jsonRes;
    }

    @SuppressWarnings("deprecation")
    @Override
    protected void onPostExecute(final String file_url) {
        dismissDialog(progress_bar);

        _listaAdapter = new ListaCarrosAdapter(ListaCarrosActivity.this, _listaCarros);
        _lvCarros.setAdapter(_listaAdapter);
    }
}
```

Fonte: Próprio autor

Na figura 20 é ilustrado a classe RestClient.class responsável pela comunicação com o *WebService*.

Figura 20– Classe RestClient - Json

RestClient.java

```
1  package br.com.catalogocarro.service;
2
3  import java.io.BufferedReader;
22
23 @SuppressLint("NewApi")
24 public class RestClient {
25
26
27     public static String get(String url) {
28         StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder()
29                 .permitAll().build();
30         StrictMode.setThreadPolicy(policy);
31         StringBuilder builder = new StringBuilder();
32         HttpClient client = new DefaultHttpClient();
33         HttpGet httpGet = new HttpGet(url);
34         try {
35             client.getParams().setParameter(
36                     CoreConnectionPNames.CONNECTION_TIMEOUT, 100000);
37             HttpResponse response = client.execute(httpGet);
38             StatusLine statusLine = response.getStatusLine();
39             int statusCode = statusLine.getStatusCode();
40             if (statusCode == 200) {
41                 HttpEntity entity = response.getEntity();
42                 InputStream content = entity.getContent();
43                 BufferedReader reader = new BufferedReader(
44                         new InputStreamReader(content));
45                 String line;
46                 while ((line = reader.readLine()) != null) {
47                     builder.append(line);
48                 }
49             }
50         } catch (ConnectTimeoutException e) {
51             e.printStackTrace();
52         } catch (ClientProtocolException e) {
53             e.printStackTrace();
54         } catch (IOException e) {
55             e.printStackTrace();
56         } catch (Exception e) {
57             e.printStackTrace();
58         }
59         return builder.toString();
60     }
61
62     public static boolean checkConnection(Context context) {
63         boolean conectado;
64         ConnectivityManager conectivtyManager = (ConnectivityManager) context
65                 .getSystemService(Context.CONNECTIVITY_SERVICE);
66         if (conectivtyManager.getActiveNetworkInfo() != null
67                 && conectivtyManager.getActiveNetworkInfo().isAvailable()
68                 && conectivtyManager.getActiveNetworkInfo().isConnected()) {
69             conectado = true;
70         } else {
71             conectado = false;
72         }
73         return conectado;
74     }
75 }
```

Fonte: Próprio autor

3.2.2.2 Implementação utilizando a API Volley

Na figura 21 é ilustrado a classe ListaCarrosActivity.class utilizando a API *Volley*.

Figura 21- Utilização API Volley

ListaCarrosActivity.java

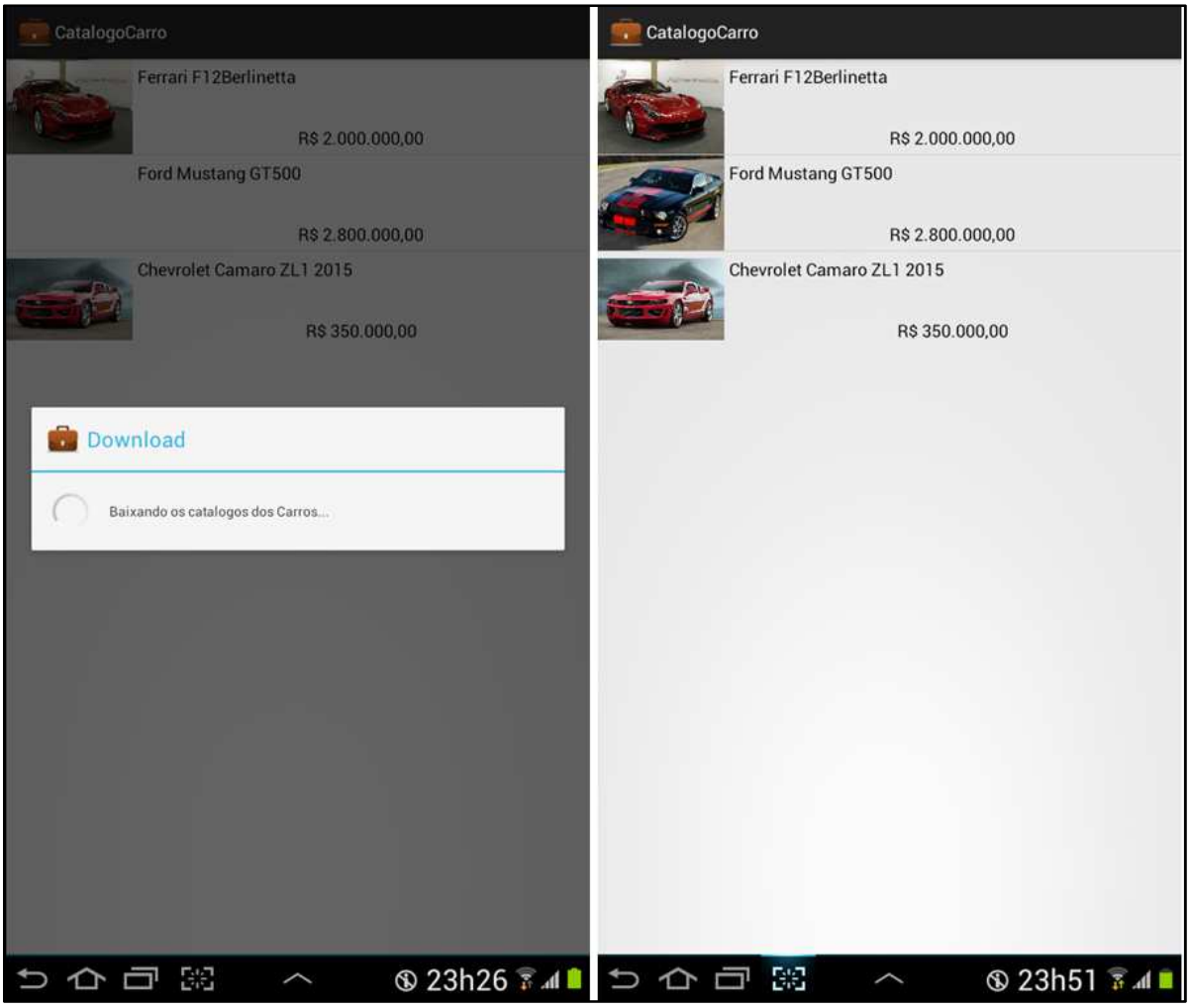
```
1  package br.com.catalogocarros;
2  import java.util.ArrayList;
19
20 public class ListaCarrosActivity extends Activity {
21
22     private ProgressDialog progressDialog;
23     public static final int progress_bar = 0;
24     private List<Carro> _listaCarros;
25     private ListaCarrosAdapter _listaAdapter;
26
27     ListView _lvCarros;
28     Resources res;
29
31     protected void onCreate(final Bundle savedInstanceState) {
38
40     protected Dialog onCreateDialog(final int id) {
55
56     class DownloadCarros extends AsyncTask<String, String, String> {
57
58         @SuppressWarnings("deprecation")
59         @Override
60         protected void onPreExecute() {
61             super.onPreExecute();
62             showDialog(progress_bar);
63         }
64
65         @Override
66         protected String doInBackground(final String... args) {
67
68             JsonObjectRequest jsonObjReq = new JsonObjectRequest(Method.GET,
69                     args[0], null, new Response.Listener<JSONObject>() {
70
71                         @Override
72                         public void onResponse(JSONObject response) {
73
74                             try {
75                                 JSONArray jsonResposta = response
76                                         .getJSONArray("carros_exemplo");
78                                 _listaCarros = new ArrayList<Carro>();
79
80                                 for (int i = 0; i < jsonResposta.length(); i++) {
81                                     JSONObject carroObj = jsonResposta.getJSONObject(i);
82                                     Carro carro = new Carro();
83                                     carro.setNome(carroObj.getString("nome"));
84                                     carro.setImagem(carroObj.getString("imagem"));
85                                     carro.setDescricao(carroObj.getString("descricao"));
86                                     carro.setValor(carroObj.getString("valor"));
87                                     _listaCarros.add(carro);
88                                 }
89                             } catch (JSONException e) {
90                                 e.printStackTrace();
91                                 Toast.makeText(getApplicationContext(),
92                                         "Error: " + e.getMessage(),Toast.LENGTH_LONG).show();
93                             }
94                         }
95                     }, new Response.ErrorListener() {
96                         @Override
97                         public void onErrorResponse(VolleyError error) {
98                             Toast.makeText(getApplicationContext(),
99                             error.getMessage(), Toast.LENGTH_SHORT).show();
100                        }
101                    });
102            return jsonObjReq.toString();
103        }
104
105        @SuppressWarnings("deprecation")
106        @Override
107        protected void onPostExecute(final String file_url) {
108            dismissDialog(progress_bar);
109            _listaAdapter = new ListaCarrosAdapter(ListaCarrosActivity.this,
110                    _listaCarros);
111            _lvCarros.setAdapter(_listaAdapter);
112        }
113    }
114 }
```

Fonte: Próprio autor

Como se pode observar, a grande vantagem na utilização da API *Volley* foi a ausência da classe RestClient.class, isto porque a *Volley* já realiza internamente toda a comunicação com o *WebService*, tornando o desenvolvimento mais simples e com código mais elegante.

Na figura 22 é ilustrado o resultado das aplicações que consumiram o JSON criado na ferramenta *apiary.io.*

Figura 22- Catalogo de Carros - Json



Fonte: Próprio autor

CAPÍTULO 4 – RESULTADOS E ANÁLISES

4.1 PERSISTÊNCIA DOS DADOS EM BANCO

Para realizar um comparativo entre os modelos de desenvolvimento propostos, para cada requisito estabelecido abaixo foram especificadas algumas métricas de avaliação para cada modelo, definidos em Bom, Médio e Ruim.

Criação da base de dados: Nesta métrica foi apresentada a forma como é realizado a criação de toda a base de dados em cada modelo de desenvolvimento apresentado no trabalho.

Padrão de Projeto utilizado: Nesta métrica foi apresentado o padrão de Projeto que cada modelo de desenvolvimento apresentado utiliza.

Independência de SQL: Nesta métrica foi avaliada a independência da utilização de SQL em cada modelo apresentado, visando a otimização de códigos fontes.

- Componentes nativos: Ruim. Extremamente dependente de SQL.
- *ActiveAndroid*: Bom. Praticamente não foi preciso utilizar SQL na sua implementação.
- *AlienDroid*: Médio. Foi necessário a utilização de SQL para realizar relacionamentos entre os objetos do banco.

Suporte e Documentação comunitária: Nesta métrica foi avaliado o nível de cada modelo apresentado referente ao fornecimento de documentações e também o suporte a cada ferramenta.

- Componentes nativos: Bom. Possuí boa documentação e de fácil acesso.
- *ActiveAndroid*: Bom. Possuí boa documentação e de fácil acesso.
- *AlienDroid*: Bom. Possuí boa documentação e de fácil acesso.

Atuação: Nesta métrica foi avaliado no nível de atuação de cada modelo apresentado no ponto de vista do desenvolvedor, desde consultas dos dados até a utilização dos métodos do CRUD.

- Componentes nativos: Bom. Foi possível realizar toda a persistência dos dados.
- *ActiveAndroid*: Bom. Foi possível realizar toda a persistência dos dados.

- *AlienDroid*: Ruim. Não provê formas de realizar relacionamento entre os objetos do banco.

Simplicidade: Nesta métrica foi avaliado a simplicidade de cada modelo apresentado, levando em conta o tempo e facilidade de implementação, e também a necessidade de treinamento para uma melhor compreensão do código final.

- Componentes nativos: Média. Implementação fácil, porém custosa. A necessidade de SQL torna este modelo mais lento e de maior esforço para o entendimento.
- *ActiveAndroid*: Bom. Implementação fácil e rápida, sem necessidade de grandes esforços. Código Simples e de fácil entendimento.
- *AlienDroid*: Média. Implementação fácil, porém custosa. A necessidade de SQL torna este modelo mais lento e de maior esforço para o entendimento.

Desempenho: Nesta métrica foi avaliado o desempenho de cada modelo apresentado em nível de tempo de execução e utilização de recursos do Dispositivo móvel como memória e processamento.

- Componentes nativos: Média. O tempo de resposta para realizar a persistência no banco de dados foi concluído com uma diferença de 0.673 segundos comparado ao *ActiveAndroid.*
- *ActiveAndroid*: Bom. O tempo de resposta para realizar a persistência no banco de dados foi concluído de forma quase que instantânea. A utilização da memória pode ser considerada baixa.
- *AlienDroid*: Média. O tempo de resposta para realizar a persistência no banco de dados foi concluído de forma quase que instantânea, porém não foi possível avaliar seu desempenho para relacionamentos entre os objetos. A utilização da memória pode ser considerada baixa.

Tabela 1 - Resultados dos testes para a persistência em banco de dados

Resultados dos testes com modelos para a persistência em Banco de dados			
Critério	**Componentes Nativos**	**ActiveAndroid**	**AlienDroid**
Criação da base de dados	Feita por SQL	Mecanismo de Anotação	Utilização da linguagem
Padrão de projeto Utilizado	Livre	ActiveRecord	ActiveRecord
Independência de SQL	Ruim	Bom	Média
Suporte e Documentação	Bom	Bom	Bom
Atuação	Bom	Bom	Ruim
Simplicidade	Média	Bom	Média
Desempenho	Média	Bom	Média

Fonte. Próprio autor

Após os testes realizados, foi possível identificar que a utilização do *ActiveAndroid* com relação aos componentes nativos não teve grandes variações nos resultados, porém os dois modelos se mostraram mais completos e viáveis comparados com o modelo *AlienDroid*.

4.2 PERSISTÊNCIA DOS DADOS EM WEBSERVICE

Para realizar o comparativo entre os modelos de desenvolvimento propostos na utilização de *WebService*, foram dividas duas partes sendo que a primeira é tratado simplesmente o carregamento de imagens, e a segunda o carregamento de imagens consumindo o *WebService* através de JSON e através da API *Volley*.

4.2.1 Carregando Imagens

Para este trabalho, o teste de carregamento de imagens foi realizado através de implementação utilizando componentes nativos e a API Picasso conforme já demonstrado no decorrer do trabalho.

A tabela 2 apresenta o resultado do desempenho das implementações com base em diferentes dimensões de imagens.

Tabela 2 – Resultados dos testes para o carregamento de Imagens

Resultados de desempenho nas implementações para carregamento de Imagens		
Dimensão da imagem	**Componentes Nativos**	**API Picasso**
620X465 pixels	2.143 Segundos	2.093 Segundos
1280X790 pixels	6.274 Segundos	5.145 Segundos
2275X1160 pixels	13.765 Segundos	8.342 Segundos
3211X1861 pixels	15.321 Segundos	13.698 Segundos
7200X4051 pixels	21.643 Segundos	18.342 Segundos

Fonte. Próprio autor

Após os testes realizados foi possível identificar que, com a utilização da API *Picasso* o carregamento das imagens se mostrou mais rápido. Não houve grandes variações nos resultados, porém é válido ressaltar que o teste foi realizado apenas com uma imagem para cada dimensão apresentada, aplicando os mesmos testes com uma quantidade maior de imagens as variações podem aumentar consideravelmente, tornando assim a API *Picasso* uma excelente ferramenta.

4.2.2 Carregamento de JSON

Para este trabalho, o teste de integração com a API *WebService* através de JSON foi implementada utilizando a API *Volley* e através de componentes nativos.

A tabela 3 apresenta o resultado do desempenho das implementações com base em diferentes quantidades de dados.

Tabela 3 – Resultados dos testes de integração JSON com WebService

Resultados de desempenho nas implementações de integração do API WebService		
Número de Imagens	**Componentes Nativos**	**API Volley**
3 Imagens	7.256 Segundos	7.184 Segundos
20 Imagens	19.942 Segundos	18.493 Segundos
50 Imagens	43.283 Segundos	41.834 Segundos
100 Imagens	01.49 Minutos	01.15 Minutos
500 Imagens	06.15 Minutos	05.37 Minutos

Fonte. Próprio autor

Assim como no teste de carregamento de imagens, não houve grandes impactos na utilização da API *Volley* para integração com *WebService* comparado com a utilização de

componentes nativos. O fator satisfatório na utilização da *Volley* foi a otimização de código fonte.

4.3 Análises dos Resultados

4.3.1 Banco de Dados

A utilização das técnicas e ferramentas apresentadas no decorrer do trabalho para a persistência de dados no banco foi extremamente satisfatória e proporcionou ótimos resultados comparado ao modelo tradicional, onde foi possível observar a redução de instruções SQL juntamente com códigos da linguagem de programação, facilitando o entendimento da aplicação e tornando o código mais elegante e otimizado, e também foi possível identificar que através dessas ferramentas, o desenvolvimento ficou mais simples e rápido, fatores preponderantes no desenvolvimento de sistemas.

Na utilização das ferramentas, como ponto negativo pode se destacar a carência de algumas delas quando é necessário realizar relacionamentos entre as tabelas do banco, o que acarreta na necessidade de mais instruções SQL no código fonte.

4.3.2 WebService

Para as técnicas apresentadas no carregamento de imagens e consumo de *WebService*, foi possível observar que com a utilização das ferramentas, assim como os testes realizados com o banco de dados, os resultados foram satisfatórios comparados a utilização dos componentes nativos. Em relação a desempenho não houve muitas variantes dentre as técnicas apresentadas, porém quando analisado a otimização de código fonte e facilidade de implementação, as ferramentas se destacaram por sua simplicidade e eficiência. Com a utilização das APIs *Volley* e *Picasso* a implementação reduziu para mais que a metade, e a qualidade e desempenho das aplicações melhoram.

CONCLUSÃO

Tendo em vista a importância da mobilidade para a sociedade e o aumento na utilização dos dispositivos móveis, o objetivo deste trabalho foi contribuir com os desenvolvedores de aplicativos móveis através de avaliações entre técnicas de mecanismos de entrada e saída de informações em dispositivos móveis.

Foram apresentadas diversas técnicas tanto para a persistência de dados no banco quanto para o carregamento de imagens e consumo de *WebServices*, tais técnicas que proporcionaram resultados satisfatórios e que poderão ser utilizados tanto para o uso pessoal quando para o uso corporativo.

Após as implementações das ferramentas de persistência dos dados conclui-se que, quando se trata de redução de implementações e redução de mescla de instruções SQL com códigos fontes, as ferramentas *ActiveAndoid* e *AllienDroid* tem uma grande superioridade com relação aos modelos tradicionais de implementação, porém ao contrário do *ActiveAndroid*, o *AllienDroid* apresentou carências no quesito relacionamentos entre os objetos do banco.

Conclui-se também que, para o carregamento de imagens e integração com *WebService*, as APIs *Volley* e *Picasso* mostraram uma superioridade muito grande comparado aos componentes nativos, as técnicas se destacaram por simplicidade de implementação, melhor desempenho e também boa documentação em repositórios públicos, o que contribuiu para as implementações no decorrer do trabalho.

Com o desenvolvimento do trabalho, foi possível concluir que através da utilização de boas práticas e técnicas de armazenamento dos dados eficazes, é possível a criação de aplicações móveis melhores e mais eficientes, contornado muitas vezes as limitações que os dispositivos móveis ainda apresentam, e também problemas diários vivenciados pelos desenvolvedores, de forma que tais soluções atendam as suas reais necessidades.

REFERÊNCIAS

APPLE**. iOSDev Center**, 2013. Disponivel em: <https://developer.apple.com/library/ios/#referencelibrary/GettingStarted/RoadMapiOS/chapters/Introduction.html>. Acesso em: 01 Junho 2014.

BASIURA, Russ et al. **Professional ASP.NET Web Services:** De programador para programador. São Paulo: Pearson Education, 2003. 712 p.

BORGES JUNIOR, Mauricio Pereira. **Desenvolvendo WebServices**: Guia rápido usando Visual Studio .NET com Banco de Dados SQL Server. 1. ed. Rio de Janeiro: Editora Ciencia Moderna Ltda., 2005. 144 p.

B'FAR, Reza. **Mobile computingprinciples: designinganddeveloping mobile applicationswith UML and XML**. Cambridge University Press, 2004.

CARVALHO, Luiz. **Android: AsyncTasks e Handlers**. 2014. Disponívelem: <http://www.redrails.com.br/2014/03/01/android-asynctasks-e-handlers/>. Acesso em: 27 out. 2014.

EDUARDO, Carlos. **Armazenamento de dados em Sistemas Gerenciadores de banco de dados Relacionais (SGDBR's)**.2011. Disponível em: <http://www.fatecsp.br/dti/tcc/tcc0007.pdf>. Acesso em: 23 Nov. 2014.

FRANÇA, Juliana. **Plataformas de Desenvolvimento para Dispositivos Móveis**. 2007. Disponível em: <http://www-di.inf.puc-rio.br/~endler/courses/Mobile/Monografias/07/Android-Juliana-Mono.pdf>. Acesso em: 15 ago. 2014

GADELHA, A. et. Al. **Sistema Operacional IOS**. 2012. Disponível em: <http://files.duvidascomputacao.webnode.com.br/20000044608cac09c4a/IOS%20noite%20OK.pdf>. Acesso em: 25 ago. 2014

GONÇALVES, Eduardo Corrêa. **Introdução ao formato JSON**. Disponível em: <http://www.devmedia.com.br/introducao-ao-formato-json/25275>. Acesso em: 25 out. 2014.

GOOGLE. **StorageOptions**. Disponível em: <http://developer.android.com/guide/topics/data/data-storage.html>. Acesso em 06 de junho de 2014

JSON.ORG. **Introdução ao JSON**. Disponível em: <http://www.json.org/json-pt.html>. Acesso em: 25 out. 2014.

KIRKPATRICK, Fícus. **Volley**.2013. Disponível em: http://commondatastorage.googleapis.com/io-2013/presentations/110%20-%20Volley-%20Easy,%20Fast%20Networking%20for%20Android.pdf> Acesso em: 20 Nov. 2014.

MARIA, Rafaela. **Desenvolvimento de Aplicativos para Dispositivos Móveis**. 2010. Disponível em: < http://fateczl.edu.br/TCC/2010-2/TCC-009.pdf>. Acesso em: 30 ago. 2014

MENDONÇA, Aderval. **Mobilidade em Análise**. Disponível em: <http://www.devmedia.com.br/mobilidade-em-analise/3309>. Acesso em: 14 ago. 2014.

MONTEBUGNOLI, Thiago C. M. **Android – Gerenciadores de Base de Dados SQLite**. Disponível em: <http://www.theclub.com.br/restrito/revistas/201306/andr1306.aspx>. Acesso em: 06 de junho de 2014.

PARDO, Michael. **ActiveAndroid**. Disponível em: <http://www.activeandroid.com/>. Acesso em 10 Set. de 2014.

PICASSO. **Picasso**. Disponível em: <http://square.github.io/picasso/>. Acesso em 20 Out. de 2014.

RAMOS, Leandro O; DUARTE, Roseclea M. **Desenvolvimento de objetos de aprendizagem para dispositivos móveis: uma nova abordagem que contribui para a educação**. 2007. Disponível em: < http://www.cinted.ufrgs.br/ciclo9/artigos/4aLeandro.pdf>. Acesso em: 20 ago. 2014

RASMUSSEN, Bruna. **Diário de um Android,** 2011**.** Disponível em: <http://www.tecmundo.com.br/infografico/9010-android-o-sistema-operacional-movelque-conquistou-o-mundo.htm> Acesso em 10de Março de 2014.

SADAGI, I. et. Al. **Analise de bibliotecas para WebServices no Desenvolvimento em Smartphones baseado no sistema operacional Microsoft Windows Phone**. 2013. Disponível em: <http://lyceumonline.usf.edu.br/salavirtual/documentos/2455.pdf>. Acesso em: 26 out. 2014

SAFATLI, Nabil. **Introdução ao Desenvolvimento em Windows Phone**. Disponível em: <http://www.devmedia.com.br/introducao-ao-desenvolvimento-em-windows-phone/26642>. Acesso em: 27 ago. 2014.

SILVA, Marlon. **AlienDroid – Abstração do Acesso a Dados em Android**. 2013. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/sbsi/2013/0034.pdf>. Acesso em 15 Set. de 2014.

SQLite. **AboutSQLite**. Disponível em: <http://www.sqlite.org/about.html>. Acessoem 15 Julho de 2014.

SHANKLAND, S. Google's Android parts ways with Java industry group. **CNET**, 2007. Disponivel em: <http://www.cnet.com/news/googles-android-parts-ways-with-java-industry-group/>. Acessoem: 24 Março 2014.

WILSON, M. T-Mobile G1: Full Details of the HTC Dream Android Phone. **gizmodo**, 2008. Disponivel em: <http://gizmodo.com/5053264/t+mobile-g1-full-details-of-the-htc-dream-android-phone>. Acesso em: 24 Abril 2014.

YURI, Samuel. **Framework para Mapeamento Objeto-Relacional em Delphi**.2010. Disponível em: http://deschamps.blog.com/files/2011/03/Framework-ORM-Delphi-Monografia.pdf>. Acessoem: 22 Nov. 2014.

ZIEGLER, C. Microsoft talks Windows Phone 7 Series development ahead of GDC: Silverlight, XNA, and no backward compatibility. **Endgadget**, 2010. Disponivel em: <http://www.engadget.com/2010/03/04/microsoft-talks-windows-phone-7-series-development-ahead-of-gdc/>. Acesso em: 29 Abril 2014.