UNIVERSIDADE FEDERAL DOS VALES DO JEQUITINHONHA E MUCURI
FACULDADE DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE SISTEMAS DE INFORMAÇÃO

SISTEMA MÓVEL DE MONITORAMENTO DE ESPÉCIES PARA UNIDADE DE CONSERVAÇÃO

Mateus de Oliveira Andrade

Diamantina
2014

UNIVERSIDADE FEDERAL DOS VALES DO JEQUITINHONHA E MUCURI
FACULDADE DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE SISTEMAS DE INFORMAÇÃO

SISTEMA MÓVEL DE MONITORAMENTO DE ESPÉCIES PARA UNIDADE DE CONSERVAÇÃO

Mateus de Oliveira Andrade

Orientador:

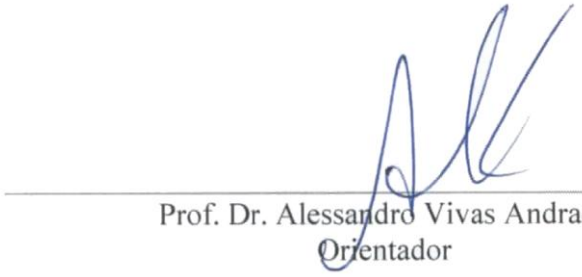
Prof. Dr. Alessandro Vivas Andrade

Trabalho de Conclusão de Curso apresentado ao Curso de Sistemas de Informação, como parte dos requisitos exigidos para a conclusão do curso.

Diamantina
2014

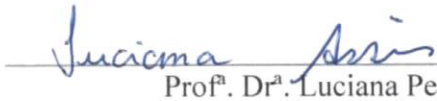
Monografia de projeto final de graduação sob o título “Sistema Móvel de Monitoramento de Espécies para Unidades de Conservação”, defendida por Mateus De Oliveira Andrade e aprovada em 15 de julho de 2014, em Diamantina, Minas Gerais.

Banca Examinadora:



Prof. Dr. Alessandro Vivas Andrade
Orientador

Prof. Dr. Rafael Santin



Profª. Drª. Luciana Pereira de Assis

À todos meus amigos e familiares...

AGRADECIMENTOS

Desejo expressar minha sincera gratidão a todos que contribuíram para o sucesso deste trabalho. Agradeço primeiramente à entidade criadora do universo (Deus) que concedeu a nós, seres humanos, a capacidade de criar e compartilhar conhecimentos. Agradeço aos meus pais Herivelto e Maristela, pelo apoio e conselhos concedidos nos momentos em que eu mais precisei. Agradeço ao meu irmão Júnior pelas conversas que sempre me guiaram para o melhor. Agradeço a minha irmã Daniela por todas as palavras que sempre me motivaram. Agradeço à minha amada família, tios e avó, pelos ensinamentos. Um agradecimento especial ao meu amigo Mateus Felipe que sempre esteve do meu lado nas horas boas e ruins e vivenciou grande parte da minha trajetória. Também agradeço ao meu amigo Bruno Fraga que acrescentou significativamente no meu aprendizado em tão pouco tempo que nos conhecemos. Agradeço também ao meu orientador Alessandro Vivas que sempre demonstrou compromisso e paciência comigo durante toda a elaboração do trabalho.

“A vida é uma sequência
de encontros inéditos com o mundo
e portanto, ela não se deixa traduzir
em fórmula de nenhuma espécie.”

Clóvis de Barros Filho

RESUMO

O desenvolvimento de aplicativos para dispositivos móveis envolve conhecimentos sobre as linguagens de programação que serão utilizadas e principalmente sobre qual plataforma o sistema será projetado. Depois de estabelecidos estes pressupostos, deve-se realizar uma análise sobre quais os requisitos que o software deve possuir. Este trabalho teve como foco o desenvolvimento de uma ferramenta para auxiliar os pesquisadores do Parque Nacional das Sempre-Vivas no trabalho de registro das espécies de plantas encontradas no parque. Esta versão 1.0 do software ModuloParque permite cadastrar espécies com o nome, nome científico, um texto explicativo, uma imagem e a localização(longitude, longitude). Utilizando esta ferramenta os pesquisadores poderão realizar suas coletas de dados de e posteriormente analisar o mapa de espécies que será gerado. Com este mapa pronto será possível mapear locais com maior ou menor incidência de uma determinada espécie.

Palavras-chave: Android. Java. Eclipse. XML. Smartphone. Tecnologia. Geolocalização.

ABSTRACT

The development of applications for mobile devices involves knowledge of programming languages to be used and especially on what platform the system is designed. Once established these assumptions, we should perform an analysis on what requirements the software must possess. This work focused on the development of a tool to assist researchers in the Parque Nacional das Sempre-Vivas in the work of registration of plant species found in the park. This version 1.0 of the software ModuloParque allows registering the species with name, scientific name, an explanatory text, a picture and the location (lontitude, longitude). Using this tool researchers can perform their collections data and then analyze the map of species that will be generated. With this finished map will be possible to map areas with higher or lower incidence of a species.

Keywords: Android. Java. Eclipse. XML. Smartphone. Tech. Geolocation.

SUMÁRIO

1 INTRODUÇÃO

A presente monografia aborda o projeto de desenvolvimento de um software capaz de auxiliar no monitoramento de espécies de plantas em unidades de conservação. A priori, será apresentado o objeto de estudo que referencia a temática principal do software (Seção 1.1), os objetivos a serem alcançados com o trabalho (Seção 1.2) e por fim a estrutura do trabalho (Seção 1.3).

1.1 Apresentação

O Brasil possui 313 unidades de Conservação Federais que estão sob a gestão do Instituto Chico Mendes de Conservação da Biodiversidade (ICMBIO). Unidades de Conservação são popularmente conhecidas como parques e reservas e estão divididas em dois grandes grupos: Grupo de Proteção Integral e grupo de Uso Sustentável.

O Grupo de Proteção Integral é dividido em: Estação Ecológica (ESEC), Reserva Biológica (REBIO), Parque Nacional (PARNA), Monumento Natural (MN) e Refúgio de Vida Silvestre (REVIS).

O Grupo de Uso Sustentável é dividido e: Área de Proteção Ambiental (APA), Área de Relevante Interesse Ecológico (ARIE), Floresta Nacional (FLONA), Reserva Extrativista (RESEX), Reserva de Fauna (REFAU), Reserva de Desenvolvimento Sustentável (RDS) e Reserva Particular do Patrimônio Natural (RPPN).

As unidades de conservação estão espalhadas em todos os biomas brasileiros: Amazônia, Caatinga, Cerrado, Mata Atlântica, Pampa, Pantanal e Marinho.

O Parque Nacional das Sempre-Vivas foi criado em 13 de dezembro de 2002 e tem aproximadamente 124.154,47 hectares. O principal bioma do parque é o cerrado. Ele está localizado nas proximidades dos municípios de Olhos d'Água, Bocaiúva, Buenópolis e Diamantina. A Figura 01 apresenta os limites do Parque das Sempre-Vivas.

Figura 1.1 - Limites do Parque das Sempre-Vivas

Fonte: Parque Nacional das Sempre Vivas no Google Maps[1].

O parque tem importância estratégica na conservação de várias espécies ameaçadas como o Lobo-guará, Gato-maracajá, Tamanduá-bandeira e Tatu-canastra.

1.2 Objetivos

Este trabalho tem como objetivo desenvolver de um sistema/software para auxiliar os pesquisadores do parque a cadastrarem e registrarem as espécies do parque. Através deste aplicativo o pesquisador poderá fazer de maneira off-line o cadastro de espécies, inserir a coordenada geográfica da ocorrência da espécie, inserir foto e fazer buscas no banco de dados da aplicação.

O sistema permitirá capacidades básicas de localização e facilitará a navegação do pesquisador no parque. Este aplicativo será compatível com dispositivos da plataforma Android como celulares e Tablets.

[1] Disponível em: < https://www.google.com.br/maps/place/Parque+Nacional+das+Sempre+Vivas/@-17.7847532,-43.7599186,11z/data=!4m2!3m1!1s0x0:0x6944909c4d62c8d3> Acesso em mai 2014.

1.3 Estrutura do trabalho

No Capítulo 2 é realizado um breve histórico dos telefones celulares e redes móveis, para um melhor entendimento e contextualização.

O Capítulo 3 apresenta a estrutura do sistema operacional Android e suas especificações expondo um referencial teórico do trabalho. Também é exposto o histórico de versões deste sistema e a sua evolução com o tempo.

No Capítulo 4 são apresentadas as linguagens de programação utilizadas no desenvolvimento do projeto e suas características singulares.

O Capítulo 5 apresenta as ferramentas utilizadas para o desenvolvimento da aplicação e demonstra as maneiras de instalação e configuração.

O Capítulo 6 aborda os passos do desenvolvimento da aplicação ModuloParque e apresentação do produto final.

O Capítulo 7 apresenta as conclusões e sugestões para trabalhos futuros.

2 COMUNICAÇÃO E TECNOLOGIAS

Os dispositivos móveis utilizados para a comunicação passaram por muitas evoluções no decorrer dos últimos anos. Desenvolvido pela Motorola em 1983, o modelo DynaTac 8000X foi o primeiro a ser comercializado. Ele tinha 33 centímetros de comprimento, 8,9 centímetros de largura e 4,4 centímetros de profundidade. Pesava 790 gramas, sua bateria durava menos de meia hora e custava $ 3995 dólares somados com o plano da operadora. Ele foi apelidado pelos usuários de tijolo devido ao seu formato e peso. Este telefone permitia apenas a comunicação por voz e possuía poucos botões em seu teclado. O telefone Motorola DynaTac 8000X é apresentado na Figura 2.1.

Figura 2.1 - Motorola DynaTac 8000X

Fonte: Site content.time.com[2].

Esta foi conhecida como a primeira geração de celulares na qual muitos deles eram produzidos para serem instalados no interior de veículos devido ao peso e ao tamanho destes aparelhos.

Com a chegada da segunda geração de celulares em meados da década de 90, os celulares diminuíram consideravelmente o seu tamanho e peso como pode ser observado no modelo Nokia 3310, um representante deste período, apresentado na Figura 2.2. Dentre as novidades apresentadas desta geração podemos citar a

[2] Disponível em: <http://content.time.com/time/specials/packages/article/0,28804,2023689_2023708_2023656,00.html> Acesso em mai 2014.

tecnologia por envio de mensagens de texto, os toques ringtones monofônicos e polifônicos e os primeiros celulares com cores que possibilitaram o uso de uma nova tecnologia para envio de mensagens com músicas, imagens, vídeos.

Figura 2.2 - NOKIA 3310

Fonte: Site Windowsphonebrasil[3].

De forma paralela à evolução dos celulares, os tipos de redes para transmissão de dados também foram aprimorados. A primeira que surgiu foi a rede utilizando a tecnologia 1G. Nesta tecnologia o sinal era analógico e foi principalmente usada para transmissão de voz, apesar de que poderia ser utilizada para transmissão de dados a uma velocidade semelhante à tecnologia discada, ou seja 54kbps (kilobits por segundo).

A Internet no celular também chegou na segunda geração, com o advento da tecnologia 2G, todavia muitos websites ainda não estavam adaptados para tal uso, visto que os estes eram demandavam muita largura de banda e os primeiros dispositivos e operadoras não ofereciam alta qualidade de serviço. Então surgiu o 2,5G que aumentou consideravelmente a velocidade da internet e oferecia até 114kbps de download e 48kbps de upload.

O sistema operacional Android foi incorporado aos celulares da 3ª geração, os também chamados Smartphones. O fator marcante desta geração é a tela sensível ao toque, ou seja, o touch screen. Esta tecnologia foi criada a mais de

[3] Disponível em: <http://windowsphonebrasil.com.br/nokia-foi-longe-demais-na-brincadeira-de-1o-de-abril-anunciando-o-nokia-3310-com-pureview-de-41mpx/> Acesso em mai 2014.

uma década pela IBM com o aparelho chamado Simon, em 1993. Mas na época o celular não possuía os diversos recursos que temos hoje. A Figura 2.3 ilustra o aparelho IBM Simon.

Figura 2.3 - IBM Simon

Fonte: Site www.icg.tugraz.at[4].

Esta forma de interação homem-máquina possibilitou uma melhor experiência com o usuário e a Apple aproveitou esta tecnologia e lançou o iPhone, que praticamente ditou tendências de mercado para os próximos anos. Com os celulares desta 3ª geração, também surgiu o 3G, que permitiu vídeo chamadas, download de aplicativos e uso de uma internet de alta velocidade chegando até 21Mbps(Megabit por segundo), entretanto no Brasil grande parte dos planos oferece apenas 1Mbps.

Com a grande sucesso da Apple, o Google percebeu que deveria entrar neste mercado em ascensão e em 2005, o Google comprou a Android, INc, uma empresa com apenas 22 meses de existência que trabalhava com o desenvolvimento e softwares para telefones móveis.

Em 2007, um grupo de 34 empresas, incluindo a Google, criaram o Open Handset Alliance com o objetivo de acelerar a inovação em sistemas móveis e oferecer aos consumidores telefones celulares mais avançados.

Ao final de 2008 o primeiro telefone com o sistema Android foi lançado e o mercado de smartphones avançou cada vez mais nos últimos anos umas vez que o celular passou a ser um item indispensável para muitos.

[4] Disponível em: <https://www.icg.tugraz.at/~daniel/HistoryOfMobileAR/> Acesso em mai 2014.

3 SISTEMA OPERACIONAL ANDROID

Este capítulo aborda o sistema operacional Android, para o qual a aplicação foi desenvolvida. Também será apresentado o histórico de versões deste sistema e a sua evolução de funcionalidades com o tempo.

3.1 Introdução

Um sistema operacional é o software responsável por fazer o uso conveniente do hardware. Segundo Tanenbaum(2000, p. 18), um sistema operacional pode ser definido de acordo com dois pontos de vista. Um sistema operacional como máquina estendida ou como um gerenciador de recursos. No primeiro o sistema operacional é uma abstração do hardware, no qual são simplificados procedimentos complexos de baixo nível. Já no segundo o sistema operacional é um gestor que aloca de forma ordenada e controlada os recursos de uma máquina entre os vários programas que competem por eles. O kernel é a parte do sistema operacional que é responsável pelo acesso aos recursos do sistema. Ele permite o compartilhamento de todos os recursos do computador ou dispositivo. O kernel roda diretamente no hardware do processador e faz todo o trabalho de baixo nível necessário para que o processador consiga realizar suas tarefas.

O projeto do Android consiste no desenvolvimento de um sistema operacional aberto e gratuito baseado no kernel do sistema operacional Linux.

O sistema Android possui uma plataforma de código aberto sob a licença Apache de Software Versão 2.0. Segundo o Google o projeto Android foca na liberdade e escolha. O objetivo é promover a abertura do sistema no mundo móvel, encorajando todos a fazer o mesmo, ou seja, sistemas e aplicações abertas e modificáveis.

Atualmente o mercado de dispositivos móveis cresce a cada ano. Os usuários de tecnologia em geral perceberam que poderiam executar algumas de suas tarefas rotineiras como verificar uma caixa de e-mail através do seu próprio celular. O uso desta e de outras comodidades fez crescer em 78% o tráfego a partir de dispositivos móveis durante o primeiro trimestre de 2013, em relação ao mesmo período de 2012, e cerca de 109% desde 2011 (NANJI, 2013).

Dentre os sistemas operacionais que mais se destacam nas vendas temos o Android e o iOS da Apple. Segundo Müller(2014): “Somando o número de vendas de aparelhos com o sistema operacional da Google e da Apple, temos nada menos que 93,8% de todos os dispositivos comercializados em 2013. Somente no último trimestre do ano passado, os dois SOs dominaram 95,7% das vendas”. Deste percentual de vendas temos que 78.1% são dispositivos com sistema operacional Android o que justifica ser uma plataforma buscada por grande parte dos usuários.

3.2 Histórico e Versões

O Android começou sua história em 2005, quando o Google comprou a empresa que desenvolvia um pequeno sistema para dispositivos móveis. Durante dois anos o Google manteve segredo sob o desenvolvimento do sistema e em 2007 foi anunciado o nascimento do Android como plataforma. Segundo Lecheta (2010), o Android causou grande impacto quando foi anunciado, atraindo a atenção de muita gente. E devido a estes fatores o Google fez uma parceria na época com outras 33 empresas líderes no mercado de telefonia para criar a Open Handset Alliance(OHA). Em janeiro de 2014 este grupo contava com a participação de 84 empresas que desempenham o papel de inovação no mercado móvel e que juntos desenvolvem o Android.

Em Fevereiro de 2009, foi lançada a versão 1.1 do Android. Esta versão possuía suporte a diversas funcionalidades, como: Alarmes, calculadora, câmera, contatos, mensagem, música, entre outras configurações básicas.

Em maio de 2009 surgiu o Android 1.5 apelidado de Cupcake. Nesta versão o sistema ganhou suporte a bluetooth e gravação de vídeos.

Em setembro de 2009, foi lançado o 1.6 apelidado de Donut que revolucionou as buscas oferecendo um sistema de buscar por voz e gestos. Também nesta versão foi criada uma interface integrada para câmera, gravação de vídeos e a galeria, permitindo o uso de multi-seleção para editar ou deletar vários vídeos simultaneamente.

Lançado em tempo recorde, em outubro de 2009 a versão Android 2.0/2.1 Eclair ganhou o mercado pois tinha um suporte otimizado para o hardware e possuía suporte a Live wallpapers, que são os papéis de parede animados e iterativos.

Já em maio de 2010 surgiu o Android 2.2 chamado de FroYo. Nesta versão os usuários poderiam salvar aplicativos diretamente no cartão SD e suporte à tecnologia Flash.

No final do ano de 2010 foi lançado o Android 2.3 Gingerbread que trouxe um gerenciador de downloads, teclado por padrão multi-touch e compatibilidade com a tecnologia NFC.

Em fevereiro de 2011 surgiu o Android 3.0 Honeycomb. Uma versão do Android feita exclusivamente para Tablets com suporte a bate-papo em vídeo via Google Talk e aprimoramento da plataforma para trabalhar com processadores de múltiplos núcleos.

O Android 4.0, Ice CreamSandwich, veio em outubro de 2011 e foi um divisor de águas quando se fala em experiência do usuário, pois permitiu uma maior customização do sistema, recurso de deslizar para fechar aplicativos e captura de tela do dispositivo.

Em Junho de 2012 foram lançadas as versões 4.1, 4.2 e 4.3. Todas estes versões ganharam o nome de Jelly Bean e o diferencial delas foi um Bluetooth com baixo consumo de energia e suporte a fotos panorâmicas em 360 graus com o Photo Sphere.

E por fim em Outubro de 2013 apareceu o Android 4.4 chamado de Kitkat. Esta versão possui melhorias na discagem por voz na qual o usuário basta dizer "Ok Google" para que o sistema identifique um comando. Outra importante mudança foi tornar o sistema mais suave e responsivo para smartphones de custo baixo, com 512 de memória RAM.

A investida do Google em desenvolver um sistema que dê suporte para celulares de baixo custo deve-se ao fato de que permita aos fabricantes disponibilizar smartphones com preço reduzido, possibilitando assim o acesso de outras classes a estes dispositivos. Assim o Google explora outro nicho de mercado, possibilitando que o Android em tempos futuros cresça ainda mais como plataforma para dispositivos móveis, principalmente para mercados de países emergentes como: Brasil, China, Índia, México, entre outros.

Uma comparação entre as versões ativas do sistema Android em dispositivos pode ser observada na Tabela 3.1.

Tabela 3.1 - Número relativo de dispositivos que possuem Android de acordo com a versão da plataforma.

Version	Codename	API	Distribution
2.2	Froyo	8	0.7%
2.3.3 - 2.3.7	Gingerbread	10	13.5%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	11.4%
4.1.x	Jelly Bean	16	27.8%
4.2.x		17	19.7%
4.3		18	9.0%
4.4	KitKat	19	17.9%

Fonte: Google Dashboards, 2014[5].

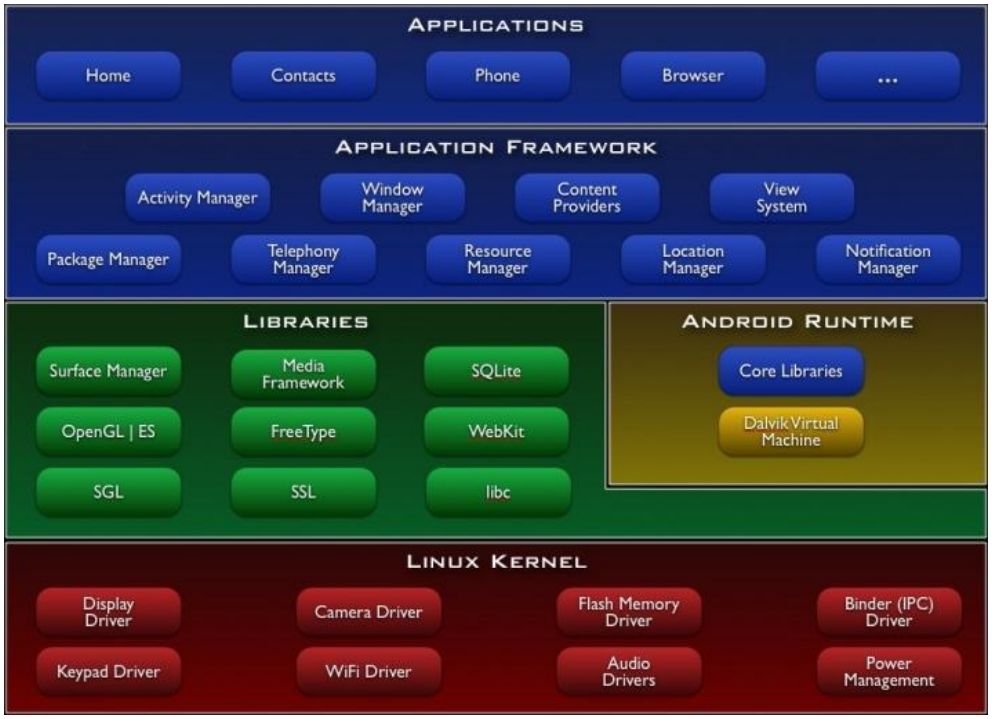
Percebe-se pela tabela que O Android 4.x mais de 56% do mercado. Isto deve-se ao fato de que muitos fabricantes lançaram atualizações para os seus dispositivos que antes possuíam a versão Gingerbread. Por enquanto a versão mais atual do sistema disponível aos usuários, o Android 4.4 Kitkat começa a crescer de forma moderada na popularidade entre os usuários.

3.3 Arquitetura

O Google faz referência ao Android como um sistema constituído de camadas e componentes de softwares no qual cada camada oferece funções e serviços que serão utilizados pela camada acima. Temos a camada base composta pelo kernel, em seguida a camada composta pelas bibliotecas nativas e Android Runtime, depois a camada dos frameworks de aplicação e por fim a camada de aplicação. A Figura 3.1 exemplifica a arquitetura do sistema Android e seus componentes.

[5] Disponível em: <http://developer.android.com/about/dashboards/index.html?utm_content=buffer07ca2&utm_source=buffer&utm_medium=twitter&utm_campaign=Buffer> Acesso em jul 2014.

Figura 3.1 - Arquitetura Android em Níveis



Fonte: Site Mobiltec[6].

3.3.1 Linux Kernel

É uma camada composta por um kernel do Linux modificado. Foi usado na construção do Android a versão 2.6 do kernel do Linux, que conta com programas para um eficiente gerenciamento de memória e energia e boas configurações de segurança. Além disso, no kernel está incluso diversos drivers de componentes que podem ser utilizados pelos mais variados tipos de hardware.

Segundo a Microsoft(2014): “Um driver é um software que permite que o computador se comunique com o hardware ou com os dispositivos.”, ou seja, o driver é um controlador que fornece uma interface de extrema importância para que seja possível a troca de informações entra o sistema operacional, e um componente físico do sistema.

3.3.2 Bibliotecas Nativas

Neste nível estão as bibliotecas, que em sua maioria foram escritas em C e

[6] Disponível em: <http://www.mobiltec.com.br/blog/index.php/desenvolvimento-mobile-nas-plataformas-android-e-ios/> Acesso em mai 2014.

C++, fornecem suporte básico ao sistema como operações de banco de dados, suporte para a parte gráfica, certificado SSL, entre outros.

Esta preferência em utilizar a linguagem C nas bibliotecas nativas é devido um fator preponderante da própria linguagem. Segundo Deitel (2014, pag. 56):

> Como C é uma linguagem padronizada, independente de hardware e amplamente disponível, aplicativos escritos em C podem ser frequentemente executados com pouca ou nenhuma modificação em uma ampla variedade de sistemas de computação diferentes.

Uma biblioteca neste nível que trabalha, por exemplo, sensores de temperatura darão suporte a esta funcionalidade para o programador. O Android usa a biblioteca SQLite para gerenciamento de base de dados.

O SQLite é um banco autocontido, compacto, com suporte nativo no Android e sem necessidade de configuração ou instalação. Isto torna-o a escolha natural para um ambiente em que devemos prezar por desempenho, disponibilidade de memória e praticidade de uso (LUZZI, 2014).

3.3.3 Android Runtime

O Android Runtime é uma aplicação de software que comporta como um dispositivo independente. É uma máquina virtual modificada e adaptada para dispositivos móveis chamada de Dalvik virtual machine. Segundo Laureano (2006, pag. 17) uma máquina virtual é uma duplicata eficiente e isolada de uma máquina real. Isto quer dizer que o papel de uma máquina virtual é compatibilizar diferentes plataformas oferendo às aplicações, uma duplicação isolada dos recursos de hardware do aparelho.

A máquina virtual Dalvik é baseada em registradores e otimizada para requerer pouca memória. Esta máquina virtual permite múltiplas instâncias e assim cada aplicação possui o seu próprio processo, retirando quaisquer dependências entre aplicações que estão rodando. Por isso uma aplicação Android em caso de falha não afeta os outros processos em andamento. O gerenciamento de memória acaba por ser simplificado por estes fatores.

3.3.4 Android Framework

É uma camada que possui diversos frameworks que trabalham com as funções básicas do smartphone. Com as ferramentas disponíveis neste nível é possível acessar a interface gráfica, localização, armazenamento no cartão SD, entre outros.

É neste nível que o programador Android trabalha e possui acesso total ao sistema. São ferramentas básicas que quando manuseadas pelo desenvolvedor, podem tornar-se operações complexas.

3.3.5 Aplicação

No nível mais acima de camadas temos o nível da Aplicação. Neste nível que as aplicações rodam, como por exemplo, fazer chamadas telefônicas, listar os contatos do dispositivo ou então acessar o navegador Web. O usuário final manipula e executa operações nesta camada.

3.4 Ciclo de Vida de uma Atividade

O ciclo de vida de uma atividade consiste nos estados em que uma aplicação ou parte dela por estar em determinados momentos. Comparando com computadores desktop, os smartphones possuem recursos bem mais limitados e isto acarreta na adoção de estratégias para utilizar estes recursos de melhor forma e tornar a experiência do usuário satisfatória.

O Android transparece que executa vários processamentos em paralelo. O que de fato acontece, é que existe uma pilha de atividades e o sistema coloca ou retira atividades desta pilha de acordo com prioridades ou ações solicitadas pelo usuário.

É permitido ao programador Android executar ações dependendo do estado atual de uma atividade. Uma atividade pode assumir os seguintes estados: onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy() e onRestart().

Ao executar uma aplicação ela entra em um estado de onCreate(). Neste estado é exibida a tela e seus componentes. É um método obrigatório chamado uma

única vez que em seguida invoca o onStart().

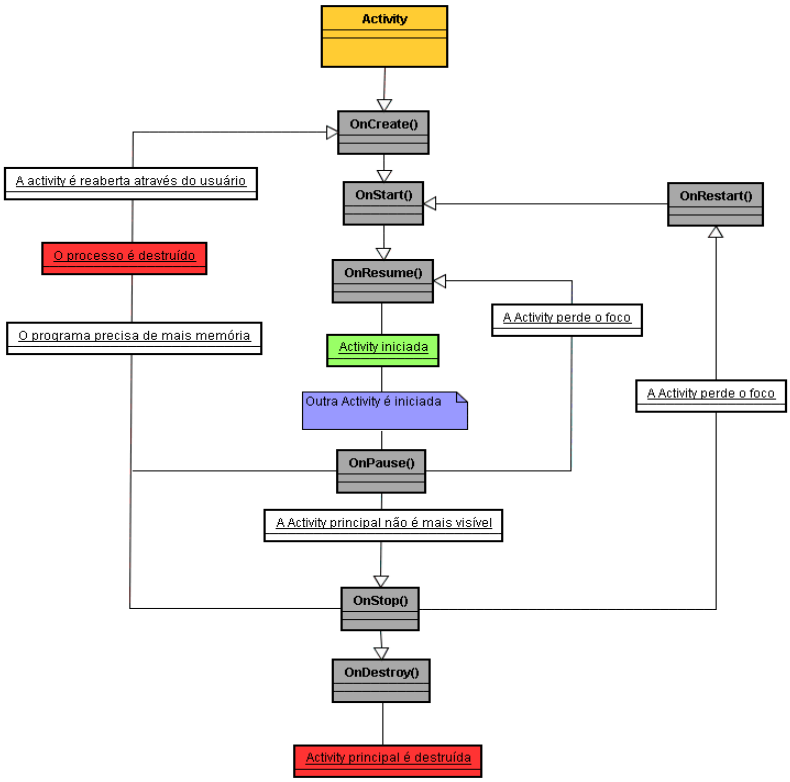
O método onStart() é chamado imediatamente após o onCreate() ou então quando uma atividade volta a ter foco. Uma atividade também pode ficar um tempo maior fora de foco. Nesse caso, ao retornar, chamaria o método onRestart(). Após estes dois métodos a atividade entra no estado onResume() que siguinifica que está em foco.

O dispositivo pode também entrar no modo de espera e nestes casos é chamando o método onPause(). Este método também é usado quando outra atividade entra em foco.

Quando o usuário sai da aplicação é chamado o onStop() para que a aplicação não fique mais visível na tela do dispositivo.

E por fim o onDestroy() que pode ser invocado pelo próprio sistema operacional e faz a tarefa de desalocação dos recursos. A Figura 3.4 exemplifica estes estados.

Figura 3.4 - Modelo de clico de vida de uma atividade



Fonte: Site do DevMedia[7].

[7] Disponível em: <http://www.devmedia.com.br/entendendo-o-ciclo-de-vida-de-uma-aplicacao-android/22922> Acesso em mai 2014.

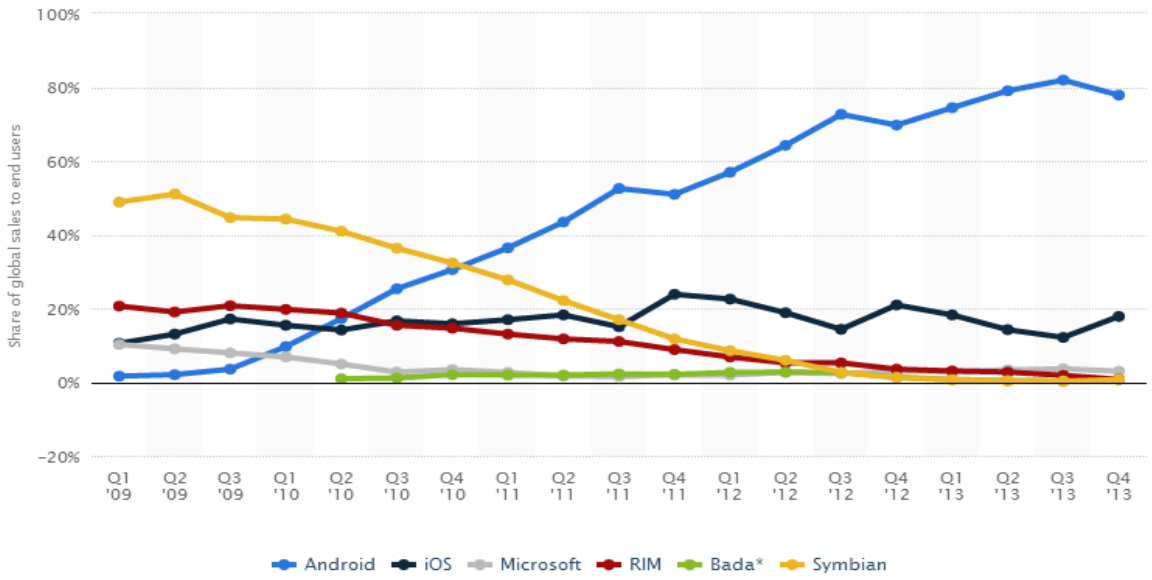
3.5 Outras plataformas

O mercado de dispositivos móveis está em ascensão. Segundo o Datafolha (2014) cerca de 43 milhões de brasileiros com 12 anos ou mais navegam pela internet utilizando dispositivos móveis. Isto mostra que estas tecnologias ganharam mercado e possuem grande representatividade como meio de acesso à Internet.

Existem outros projetos e sistemas operacionais para dispositivos móveis. Dos mais expressivos podemos destacar o iOS da Apple, a Microsoft com o Windows Phone, a Research in Motion (RIM) com o BlackBerry, Bada, Synbian, FirefoxOS, entre outros.

De acordo com o site Statista, grande portal de estatísticas e dados, o Android estava em 77,83% dos dispositivos no final de 2013. Em segundo lugar com 18,8% temos o iOS seguindo do WindowsPhone da Microsoft com 2,94%. De acordo com o Gráfico 3.1 percebe-se o grande crescimento que o sistema Android teve nos últimos trimestres e sua grande vantagem numérica de usuários comparado ao iOS.

Gráfico 3.1 - Quota global de mercado detida pelos principais sistemas operacionais de smartphones nas vendas para usuários do primeiro trimestre de 2009 até o 4 º trimestre de 2013



Fonte: Site de estatísticas Statista[8].

[8] Disponível em: <http://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/> Acesso em mai 2014

Além de ser o sistema mais usado, o Android também é bem visto devida suas vantagens como ser um sistema operacional aberto, que acarreta em aparelhos mais baratos para as operadoras e o fato de que muitas aplicações na loja do Google são gratuitas.

4 LINGUAGENS

Este capítulo apresenta as linguagens de programação utilizadas no aplicativo ModuloParque.

4.1 Java

Java foi desenvolvida por James Gosling da Sun Microsystems nos anos 90. Um dos principais objetivos dele ao propor esta tecnologia era criar uma linguagem de fácil entendimento com a qual novos programadores seriam capazes de propor soluções com a linguagem desde o início da codificação, visto a facilidade de compreensão da mesma. Também como objetivo, aplicar os conceitos de orientação a objetos de forma completa e eficiente.

Um programa escrito na linguagem Java, seja para um desktop ou para um celular, nada mais é que um conjunto de instruções que são enviadas ao processador para que o mesmo realize as tarefas necessárias para que o programa consiga rodar. Segundo Boratti(2007, pag. 68):

> Na verdade, o computador não entende Java, assim como não entende as demais linguagens de programação comumente usadas. Um computador só entende a sua linguagem, denominada Linguagem de Máquina. Essa linguagem é totalmente dependente da máquina e utiliza apenas os dígitos 0 e 1 – Sistema Binário.

Então depois de escrito um programa, as instruções devem ser traduzidas para o computador através de um compilador. A linguagem Java é dita híbrida, pois seu código fonte é sujeito a um compilador e posteriormente interpretado por uma máquina virtual. O compilador Java é chamado de Java compilador (Javac) e tem como objetivo transformar o código fonte em bytecodes. Bytecodes são códigos que podem ser interpretados pela máquina virtual Java e assim executados. A Figura 4.1 exemplifica os processos que um código java é submetido para ser executado.

Figura 4.1 - Passos da execução de um programa Java

Fonte: Site da Unicamp[9].

4.2 XML

O XML, do inglês eXtensible Markup Language, é uma linguagem de marcação utilizada principalmente em documentos hierarquicamente organizados.

Ela trabalha com o conceito de tags (rótulos) que servem como marcação de um determinado comando ou estrutura.

É uma linguagem bem versátil, que permite a criação destas tags seguindo as regras que você definir em um determinado contexto.

A figura 4.2 exemplifica a versatilidade da linguagem XML.

Figura 4.2 - Exemplo de código em XML

```
<recipe id="117" category="sobremesa">
<title> Bolo de banana </title>
<author>
<email> Miguel Furtado furtado@predialnet.com.br </email>
</author>
<date>Sunday,04 Jun 2000</date>
<description> O bolo de banana é feito com banana prata e
possui um sabor maravilhoso.
Pode ser servido quente ou frio. </description>
<ingredients> ... </ingredients>
<preparation>
 Combine tudo no liquidificador e bata até misturar bem.
 Leve ao forno por 20 minutos. Pronto. É só servir! </preparation>
<related url="#BoloChocolate">Bolo de Chocolate</related></recipe>
```

Fonte: Site da UFRJ[10].

Pode-se notar é possível criar tags de diversos nomes e, além disso, atribuir

[9] Disponível em: <http://www.unicamp.br/fea/ortega/info/cursojava/aula01.htm> Acesso em jun 2014
[10] Disponível em: <http://www.gta.ufrj.br/grad/00_1/miguel/link9.htm> Acesso em jun 2014

valores dentro destas tags. O Android faz uso do XML principalmente nas telas, visto que é necessário instanciar botões, caixas de texto e demais componentes com valores específicos de tamanho, cor, ação, entre outras características.

5 FERRAMENTAS UTILIZADAS

Este capítulo irá apresentar todas as ferramentas utilizadas para o desenvolvimento de aplicativos Android. As ferramentas de desenvolvimento podem ser instaladas em qualquer sistema operacional: Windows, Mac OS X ou Linux.

5.1 JDK ou JRE

Para o desenvolvimento das aplicações foi necessário a instalação da Máquina Virtual Java (Java Virtual Machine - JVM). A máquina virtual Java pode ser baixada através dos pacotes Java Runtime Environment (JRE) ou pelo pacote Java Development Kit (JDK). O JDK é o pacote completo de desenvolvimento, incluindo o JRE, que possui além da máquina virtual os compiladores e ferramentas (JavaDOC e o Java Debuger) para compilar os programas em linguagem java.

É possível fazer o download gratuito destas ferramentas no próprio site da Oracle, empresa multinacional que apoia o uso da linguagem Java.

JavaDoc é um gerador de documentação para códigos em Java. A partir de marcações no código fonte, comentários, ele gera toda a documentação do código em HTML. JavaDebugger, jdb, é um aplicativo utilizado para fazer depuração de códigos Java. Compilador Java, javac, é um aplicativo utilizado para transformar o código fonte em Java em bytecodes.

5.2 Integrated Development Enviroment (IDE)

Um IDE é uma ferramenta que auxilia no processo de desenvolvimento de um software. Para desenvolver um aplicativo Java basta um editor de texto simples, mas existem ferramentas que auxiliam no processo de desenvolvimento oferecendo funções mais complexas como: função de autocompletar, gerenciamento de projetos, auxílio no processo de depuração, realce de cores, dentre outros.

Existem várias IDEs, mas as que mais se destacam para o desenvolvimento de aplicativos para o sistema Android é o Eclipse, que por padrão vem no pacote do Android SDK.

5.2 O Android SDK

Para o desenvolvimento de aplicativos para o sistema Android especificadamente, o próprio Google disponibiliza um kit chamado SDK. É um Kit de Desenvolvimento de Software que possui bibliotecas e ferramentas necessárias para construir, testar e depurar aplicativos para o Android. Este kit provê grandes facilidades, pois dentro do pacote SDK vem um ambiente integrado para desenvolvimento de software denominado Eclipse.

A versão do Eclipse disponibilizada pelo SDK possui um plugin do pacote ADT instalado, que inclui os componentes essenciais para agilizar o desenvolvimento de aplicativos Android.

O objetivo do Google em disponibilizar e gerenciar de forma gratuita uma IDE de desenvolvimento para o Android é difundir o desenvolvimento de aplicativos para a plataforma e assim aumentar ainda mais o número de aplicativos para o sistema e promover o crescimento da comunidade.

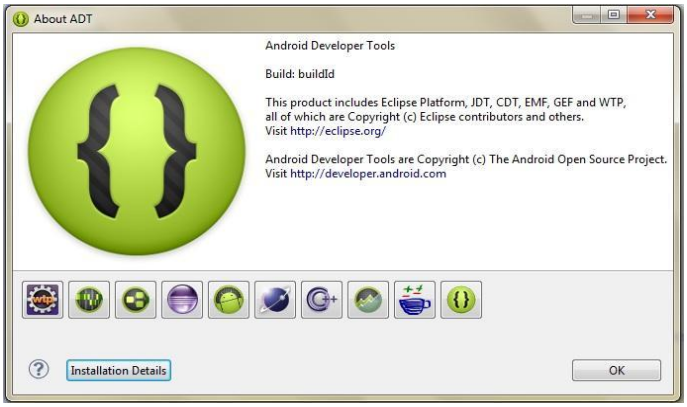
Estão presentes no pacote ADT os seguintes itens:

- Eclipse + ADT plugin
- Android SDK Tools
- Ultima versão da plataforma Android
- Ultima versão do emulador

5.2.1 Eclipse + ADT plugin

Trata-se de um pacote com a IDE Eclipse, desenvolvida de modo open source e que oferece suporte a diversas linguagens de programação, como: Java, C/C++, PHP, Python, entre outras. O ADT plugin vem acoplado ao Eclipse e trata-se das bibliotecas necessárias para o desenvolvimento. A Figura 5.1 exibe o painel sobre do ADT.

Figura 5.1 - Painel sobre o ADT



Fonte: *print screen* do painel Sobre da aplicação Eclipse.

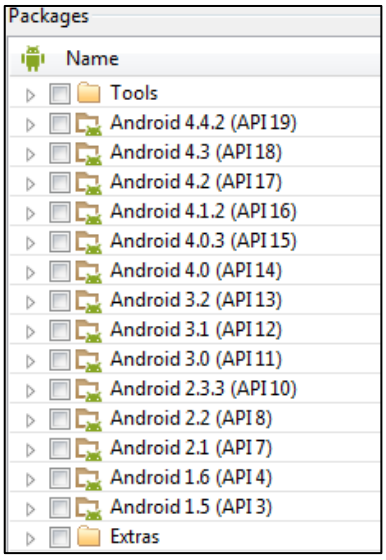
5.2.2 Android SDK Tools

Ferramentas usadas no desenvolvimento de depuração de aplicativos Android.

5.2.3 Android Platform-tools

Lista das versões disponíveis e possibilidade de fazer o download das ferramentas para determinada versão do Android. A figura 3.2 mostra estas possíveis versões para o desenvolvimento.

Figura 5.2 - Versões do Android listadas



Fonte: *print screen* das versões no Android SDK Mananger.

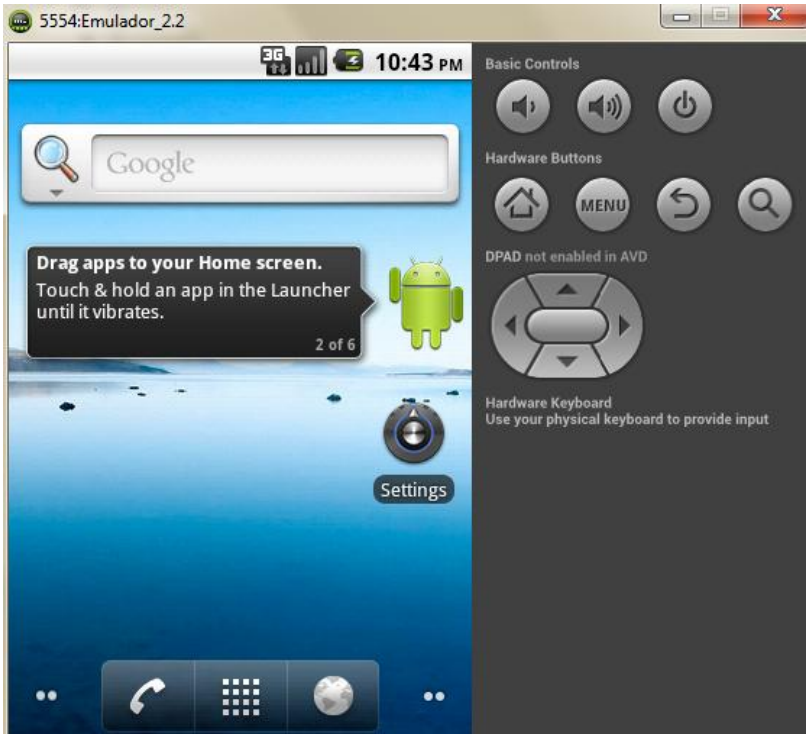
5.2.4 Ultima versão da plataforma Android

Trata-se das ferramentas para manipulação na última versão do sistema existente no mercado. Isto inclui as bibliotecas adicionais para manipulação de novos recursos que surgem em outras versões do sistema.

5.2.5 Ultima versão do emulador

É um emulador de dispositivo móvel que roda de forma virtual. Ele serve para que o desenvolvedor possa testar a sua aplicação sem a necessidade de um dispositivo físico. A Figura 5.3 exemplifica o emulador em funcionamento.

Figura 5.3 - Emulador Android rodando versão 2.2 do sistema



Fonte: *print screen* do emulador 2.2.

6 Desenvolvimento

O presente capítulo abordará o passos do desenvolvimento da aplicação ModuloParque.

6.1 Plataforma Escolhida

Para o desenvolvimento desta aplicação foi escolhido o sistema operacional Android devido a diversos fatores. Primeiro que é uma plataforma livre, que não necessita de investimentos em dinheiro com hardware próprio ou software pago para realizar o desenvolvimento de aplicações. Segundo, pois é um projeto que está em constante desenvolvimento e é gerenciado por cerca de 83 empresas além do próprio Google. Terceiro e por fim, por ser uma plataforma com vasto material para estudo e este conteúdo é de fácil acesso seja pela internet ou livros.

6.2 Levantamento dos Requisitos

Para a realização do levantamento dos requisitos que o software deveria atender utilizou-se a técnica de encontros. Foram realizadas reuniões envolvendo os pesquisadores do parque e o professor Alessandro Vivas Andrade.

Durante o processo de desenvolvimento, versões beta do aplicativo foram mostradas aos pesquisadores, visando a validação dos requisitos propostos.

6.2.1 Requisitos Essenciais

- Permitir a listagem das espécies de plantas do parque.
- Adicionar nova espécie com nome, nome científico, um texto explicativo, imagem e localização.
- Deletar espécie.
- Mapa mostrando a localização das espécies cadastradas.

6.2.2 Requisitos Desejáveis

- Possibilidade de importar foto tanto da câmera, quanto da galeria.
- Adicionar um texto com informações do parque.

6.3 Arquitetura do Software

O aplicativo ModuloParque foi desenvolvido para smartphones e tablets com sistema operacional Android versão 2.2 ou superior.

Necessário que o dispositivo acesse uma rede de dados para exibição dos mapas do Google e que o usuário habilite as opções de geolocalização no dispositivo.

O aparelho deve possuir câmera memória suficiente para armazenamento das foto e imagens associadas ao aplicativo.

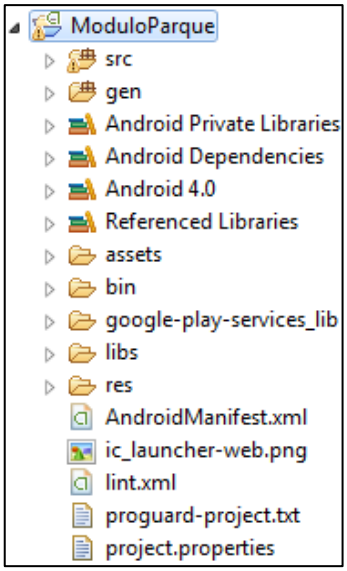
6.4 Implementação

Esta seção abordará todos os passos envolvidos na construção do e implementação do software ModuloParque.

6.4.1 Hierarquia de Pastas em uma Aplicação Android

Um projeto de aplicação para Android possui uma hierarquia de pastas bem definida. Esta hierarquia começa com a parta raiz que possui o nome do projeto. Dentro da pasta raiz há outros diretórios com os códigos, imagens, telas e arquivos de configuração. A seguir foi usada a estrutura do projeto ModuloParque como exemplo desta hierarquia. A Figura 6.1 representa todo o projeto ModuloParque em sua forma compactada.

Figura 6.1 - Estrutura do projeto MóduloParque

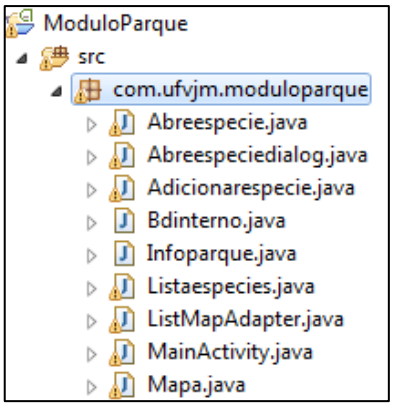


Fonte: *print screen* da estrutura do projeto no Eclipse.

6.4.1.1 Pasta src

Esta pasta contém toda a codificação Java do projeto. Aqui estão todas as classes utilizadas que representam telas, e classes de dependência que ofereceram alguma funcionalidade para a aplicação. A Figura 6.2 mostra a pasta src.

Figura 6.2 - Pasta src



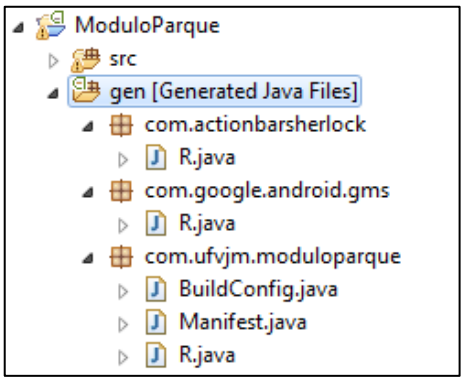
Fonte: *print screen* da pasta src.

6.4.1.2 Pasta gen

Esta pasta possui arquivos que são gerados automaticamente pelo

Eclipse. Estes arquivos são responsáveis por indexar todos componentes, telas, imagens e arquivos incorporados ao projeto. A Figura 6.3 ilustra a pasta gen.

Figura 6.3 - Pasta gen

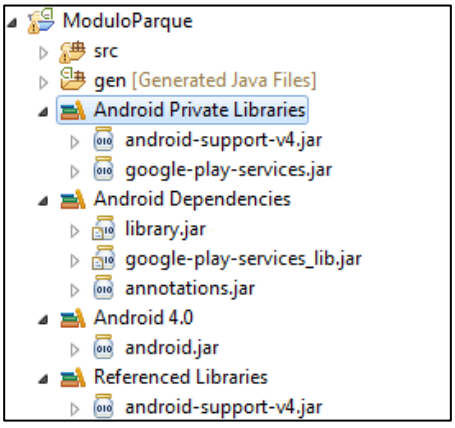


Fonte: *print screen* da pasta gen.

6.4.1.3 Bibliotecas

Região criada automaticamente com as bibliotecas utilizadas pelo aplicativo. No aplicativo MóduloParque utilizou-se a biblioteca do Android 4.0 pois é uma biblioteca que atenderia aos requisitos funcionais do aplicativo, a biblioteca de suporte versão 4, para que dispositivos com versão do Android inferior a 4.0 possam rodar a aplicação da mesma forma que dispositivos mais atuais e a biblioteca de serviços do GooglePlay que fornece o serviço de mapas do Google. A Figura 6.4 exibe as bibliotecas utilizadas na aplicação.

Figura 6.4 - Bibliotecas

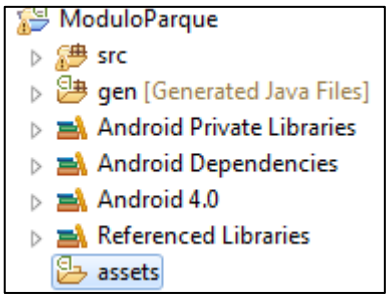


Fonte: *print screen* das bibliotecas.

6.4.1.4 Pasta assets

Esta pasta contém arquivos que podem ser incorporados ao aplicativo, como imagens, músicas e arquivos de qualquer extensão. O projeto ModuloParque não faz referência a nenhum arquivo externo, portanto não utilizou a pasta assets. A Figura 6.5 apresenta a pasta assets.

Figura 6.5 - Pasta assets



Fonte: *print screen* da pasta assets.

6.4.1.5 Pasta bin

Esta pasta contem os JARs que são arquivos compactados com classes Java. A Figura 6.6 exibe a pasta bin.

Figura 6.6 - Pasta bin

Fonte: *print screen* da pasta bin.

6.4.1.6 Pasta res

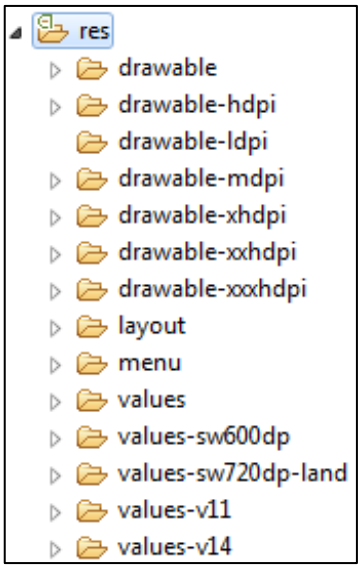
Esta pasta contém todas imagens utilizadas no aplicativo. Ela é subdividida em pastas chamadas drawable, fazendo referência ao inglês que

significaria algo que possa ser desenhado. As pastas drawable possuem denominações que servem para adaptar ao tipo de tela do dispositivo que a aplicação estiver rodando. As imagens do aplicativo são separadas nas pastas ldpi, mdpí, hdpi, xhdpi, xxhdpi e xxxhdpi.

A quantidade de pixels dentro de uma área física de tela é referida normalmente como dpi (pontos por polegada). Por exemplo, uma tela de densidade baixa possui uma quantidade menor de pixels dentro de uma área física, em comparação com uma outra tela com muitos pixels dentro de uma mesma área e portanto será uma tela de alta densidade de pixels.

Os grupos de densidades são separados no Android em quatro densidades generalizadas: baixa, média, alta e extra-alto. Percebe-se pela Figura 6.7 estes grupos de densidades de pixels e pastas drawable com nomenclaturas diferentes.

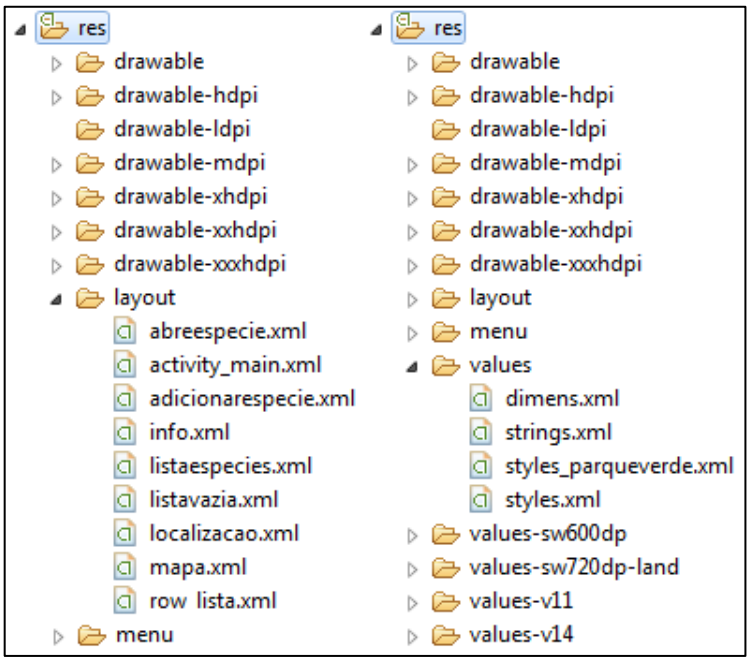
Figura 6.7 - Pasta res



Fonte: *print screen* da pasta res.

Dentro da pasta res existe uma subpasta chamada layouts e outra chamada values. A pasta layouts contém todas as telas da aplicação no formato de arquivo .XML. A pasta values contém valores que podem ser referenciados. Na Figura 6.8 são mostradas as telas em formato XML e os arquivos da pasta values.

Figura 6.8 - Pasta res/layout e pasta res/values

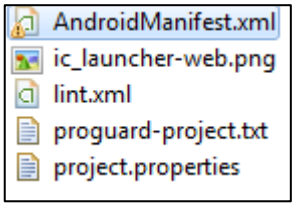


Fonte: *print screen* da pasta res.

6.4.1.7 Arquivos de configuração

Trata-se se arquivos que servem para informar a versão do aplicativo, nome, pacote, permissões e atividades. Dentro do AndroidManifest.xml também é especificada a chave de acesso à API de mapas do Google. Na Figura 6.9 é possível ver o importante arquivo project.properies que define para qual versão do Android a aplicação será compilada.

Figura 6.9 - Arquivos de configuração



Fonte: *print screen* dos arquivos de configuração.

O sistema operacional Android exige que a aplicação solicite permissões para usar os sensores, módulos e recursos do dispositivo, no qual que estiver executando. O aplicativo ModuloParque faz uso das seguintes permissões:

- INTERNET: Permissão para usar enviar e receber dados da internet.
- ACCESS_NETWORK_STATE: Permissão para acessar o estado da rede do dispositivo para saber se ele está conectado ou não juntamente com o tipo de conexão que pode ser Wifi, 2G, 3G ou 4G.
- READ_EXTERNAL_STORAGE: Permissão para escrever na memória externa.
- WRITE_EXTERNAL_STORAGE: Permissão para ler da memória externa.
- WRITE_SETTINGS: Permissão para escrever nas opções do aplicativo.
- MAPS_RECEIVE: Permissão para receber os mapas do Google.
- READ_GSERVICES: Permissão para ler o Google Play Services.
- ACCESS_COARSE_LOCATION: Permissão para acessar a latitude e longitude com base na rede wifi que estiver conectado.
- ACCESS_FINE_LOCATION: Permissão para acessar a latitude e longitude com base no GPS.

6.4.2 Principais Classes

Cada componente lógico de uma aplicação, tais como: telas que exercem funcionalidades, banco de dados, adaptadores, entre outros, devem possui classes escritas na linguagem Java que vão exercer toda a função lógica para o mesmo.

O projeto ModuloParque utilizou as seguintes classes para representação das suas funcionalidades: MainActivity.java, Bdinterno.java, Listaespecies.java, ListMapAdapter.java, Abreespecie.java, Adicionarespecie.java, Mapa.java e Infoparque.java.

6.4.2.1 MainActivity.java

A classe MainActivity é a classe principal do programa e primeira chamada no AndroidManifest.xml através do comando da Figura 6.10:

Figura 6.10 - Chamada da classe MainActivity como LAUCHER.

```
<activity
    android:name="com.ufvjm.moduloparque.MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Fonte: *print screen* de parte do arquivo AndroidManifest.xml.

Esta classe é responsável por incorporar à tela do dispositivo o menu principal da aplicação e fornecer ações para cada um dos botões presentes neste menu.

Primeiramente para carregar o arquivo de tela no formato XML no Android foi usada a notação da Figura 6.11.

Figura 6.11 - Sobrescrita do onCreate e chamada de tela XML

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```

Fonte: *print screen* de parte da classe MainActivity.java.

Após isso, são criados os ImageButtons que serão manipulados pela classe e posteriormente inicializados conforme visto na Figura 6.12.

Figura 6.12 - Criação e inicialização dos botões do menu principal

```
ImageButton btlistaespecies, btlocalizacao, btinfo;

btlistaespecies = (ImageButton)findViewById(R.id.btlistaespecies);
btlocalizacao = (ImageButton)findViewById(R.id.btlocalizacao);
btinfo = (ImageButton)findViewById(R.id.btinfo);
```

Fonte: *print screen* de parte da classe MainActivity.java.

E para atribuir ação a um botão utilizou-se o método setOnClickListener. Este método pertence a classe View. ImageButton é uma classe padrão do Android que

herda os métodos da classe View. Na figura 6.13 é chamado o setOnClickListener sobre o ImageButton para saber qual ação tomar ao botão ser clicado.

Figura 6.13 - Chamada do método setOnClickListener sobre o botão de espécies

```
btlistaespecies.setOnClickListener(new View.OnClickListener() {

    public void onClick(View v) {

        Intent chamaespecie = new Intent(getApplicationContext(),
                Listaespecies.class);
        startActivity(chamaespecie);
    }
});
```

Fonte: *print screen* de parte da classe MainActivity.java.

Neste caso ao ser clicado o botão cria uma intenção de chamar outra classe. Ao iniciar a nova classe, a classe base permanece em onPause até que seja chamada novamente e volte ao estado de onResume.

6.4.2.2 Bdinterno.java

A classe Bdinterno.java serve de apoio para criar o banco de dados da aplicação. Esta classe possui o método onCreate sobrescrito, no qual quando chamado, executa um comando SQL de criação do banco de dados. O comando da Figura 6.14 é executado ao criar um objeto da classe Bdinterno.

Figura 6.14 - Comando SQL executado para criar a tabela de espécies

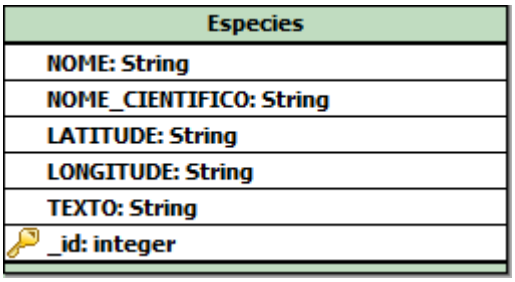
```
CREATE_TABELA = "create table " + TBL + "(
        NOME            + " text," +
        NOME_CIENTIFICO + " text," +
        LATITUDE        + " text," +
        LONGITUDE       + " text," +
        TEXTO           + " text," +
        SET_ID          + " integer primary key autoincrement);";
```

Fonte: *print screen* de parte da classe Bdinterno.java.

Através deste comando é gerada a tabela Espécies que possui os atributos de nome da espécie, o nome científico, a latitude que foi encontrada, longitude e um

texto explicativo para a espécie. Uma representação lógica da tabela Espécies pode ser observada na Figura 6.15.

Figura 6.15 - Estrutura da tabela Espécies e seus campos



Fonte: elaborada pelo autor.

6.4.2.3 ListMapAdapter.java

Esta classe é responsável pela criação do adaptador para gerar a lista de espécies. O método construtor desta classe recebe como parâmetros vetores de Strings e utiliza uma classe padrão do Android chamada ViewHolder que retorna as views por getView() automaticamente.

Essa estrutura de dados faz referencia para os objetos que vc quer atribuir valores. Esta classe evita diversos usos do método findViewById() toda vez que for gerar uma nova View. Dentro do método getView() temos a estrutura da Figura 6.16.

Figura 6.16 - Indexação do holder para os campos na tela e atribuição de valores

```
ViewHolder holder;

holder.nome = (TextView) convertView.findViewById(R.id.tvnome);
holder.nomecientifico = (TextView) convertView.findViewById(R.id.tvnomecientifico);

holder.nome.setText(this.arrayNomes[position]);
holder.nomecientifico.setText(this.arrayNomes_Cientificos[position]);
```

Fonte: *print screen* de parte da classe ListMapAdapter.java.

6.4.2.4 Listaespecies.java

Esta classe exerce a função de carregar do banco de dados as informações das espécies cadastradas e exibir em uma lista simples os nomes das espécies

associado com os nomes científicos.

Para ler os dados do banco inicializamos uma variável que irá ajudar a acessar o banco na forma de leitura e em seguida utilizar um cursor para percorrer os itens da tabela Especies. O código da Figura 6.17 ilustra o procedimento.

Figura 6.17 - Criação do helper para auxiliar na manipulação da base

```
helper = new Bdinterno(this);
bdespecies = helper.getReadableDatabase();
cursor = bdespecies.rawQuery("Select * from Especies", null);
```

Fonte: *print screen* de parte da classe Listaespecies.java.

Depois deste procedimento a classe Listaespecies.java irá verificar se algum registro foi encontrado na tabela. Se não foi encontrado, é exibida uma tela informando ao usuário que não existem espécies cadastradas e oferece a possibilidade de adicionar uma nova como mostrado na Figura 6.18.

Figura 6.18 - Tela de lista sem nenhuma espécie cadastrada

Fonte: *print screen* da tela de lista vazia.

Se durante a consulta foi encontrado algum registro, é mostrada na tela do dispositivos uma lista contendo os nomes e nomes científicos das espécies. Para tanto, utiliza-se de dois vetores de Strings que são inicializados com o número de

registros encontrados pelo cursor. A Figura 6.19 mostra esta inicialização.

Figura 6.19 - Inicialização dos vetores auxiliares para posterior leitura

```
lista_nome = new String[cursor.getCount()];
lista_nome_cientifico = new String[cursor.getCount()];
```

Fonte: *print screen* de parte da classe Listaespecies.java.

Depois estes vetores são preenchidos com todas as informações da base de dados conforme visto na Figura 6.20.

Figura 6.20 - Preenchimento dos vetores

```
int aux = 0;
cursor.moveToFirst();
while (!cursor.isAfterLast()) {
    try {

        lista_nome[aux] = cursor.getString(0);
        lista_nome_cientifico[aux] = cursor.getString(1);

        cursor.moveToNext();
    } catch (Exception e) {
        break;
    }
    ++aux;
}
```

Fonte: *print screen* de parte da classe Listaespecies.java.

Em seguida é criado um adaptador para a lista utilizando os serviços prestados da classe ListMapAdapter.java. Um adaptador é responsável por carregar um conteúdo de uma fonte, como exemplo o resultado de uma consulta de banco de dados e converte cada item do resultado em uma view que é colocada na lista. Na Figura 6.21 o adaptador é criado e em seguida atribui o adaptador na lista.

Figura 6.21 - Criação do adaptador e inserção na lista

```
adaptador = new ListMapAdapter(getApplicationContext(), lista_nome,
                lista_nome_cientifico);

lvespecies.setAdapter(adaptador);
```

Fonte: *print screen* de parte da classe Listaespecies.java.

Após mostrar a lista é implementada a ação de setOnItemClickListener sobre cada uma das views criadas na qual são passados por uma Intent os dados de nome, nome científico, latitude, longitude e o texto referentes à linha no banco de dados referente ao item clicado. A Figura 6.22 exemplifica o procedimento.

Figura 6.22 - Chamada onItemClickListener da lista

```
lvespecies.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView arg0, View view,
            int position, long index) {
        try {

            cursor.moveToPosition(position);

            String aux0 = cursor.getString(0);//Nome
            String aux1 = cursor.getString(1);//Nome cientifico
            String aux2 = cursor.getString(2);//Latidude
            String aux3 = cursor.getString(3);//Longitude
            String aux4 = cursor.getString(4);//Texto sobre

            chamaitem.putExtra("aux0", aux0);
            chamaitem.putExtra("aux1", aux1);
            chamaitem.putExtra("aux2", aux2);
            chamaitem.putExtra("aux3", aux3);
            chamaitem.putExtra("aux4", aux4);

            startActivity(chamaitem);

        } catch (Exception erro) {

        }

    }
});
```

Fonte: *print screen* de parte da classe Listaespecies.java.

Uma Intent é classe padrão do Android para realizar operações. Pode ser usada para lançar uma Activity, trocar dados entre telas, ou para se comunicar com um serviço, como por exemplo executar uma outra aplicação.

Outras duas funcionalidades que a classe Lista especies.java provê são botões na ActionBar de adicionar ou retirar itens da lista. A opção de retirar um dado opera diretamente no banco de dados, enquanto a opção de adicionar chama a classe Adicionarespecie.java.

6.4.2.5 Abreespecie.java

A classe Abreespecie.java é chamada ao clicar em algum item da lista de espécies. O procedimento executado por esta classe é receber os dados que foram passados por Intent e exibi-los em uma tela mostrando o nome da espécie, nome científico, a latitude, longitude e o texto associado. A Figura 6.23 exibe o procedimento de colocar os textos recebidos nos seus devidos lugares.

Figura 6.23 - Dados recebidos por Itent

```
intent = getIntent();
setTitle(intent.getStringExtra("aux0"));
tvnomeespecie.setText(intent.getStringExtra("aux0"));
tvnomecientificoespecie.setText(intent.getStringExtra("aux1"));
tvlatitude.setText(intent.getStringExtra("aux2"));
tvlongitude.setText(intent.getStringExtra("aux3"));
tvtextoespecie.setText(intent.getStringExtra("aux4"));
```

Fonte: *print screen* de parte da classe Abreespecie.java.

6.4.2.6 Adicionarespecie.java

É a classe responsável por adicionar espécies no banco de dados. Esta classe implementa a interface da classe LocationListener padrão do Android, para utilizar os serviços de localização com base no GPS e com base na rede wireless que o dispositivo estiver conectado.

Esta classe irá ler as entradas relacionadas ao nome, nome científico, latitude, longitude e texto da espécie para armazenar em variáveis temporárias. Este procedimento pode ser observado na Figura 6.24.

Figura 6.24 - Leitura dos dados digitados pelo usuário

```
String nome = etnome.getText().toString();
String nomecientifico = etnomecientifico.getText().toString();
String sobre = ettexto.getText().toString();
String latitude = etlatitude.getText().toString();
String longitude = etlongitude.getText().toString();
```

Fonte: *print screen* de parte da classe Adicionarespecie.java.

Também é possível associar uma imagem a um registro da espécie. São fornecidas duas possibilidades ao usuário. A primeira é utilizar a câmera do dispositivo para capturar uma foto no momento. A segunda é carregar da galeria uma foto ou imagem previamente adquirida. Para isso foi implementado por Intent a ação de chamar o capturador de imagens e a opção de selecionar arquivo de imagem externa à aplicação. A Figura 6.25 exemplifica estes dois tipos de captura.

Figura 6.25 - Implementação dos dois tipos de carregamento de imagem

```
btcamera.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {

        Intent cameraIntent = new Intent(
                android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
        startActivityForResult(cameraIntent, 1);
    }
});

btgaleria.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {

        Intent galeriaIntent = new Intent(
                Intent.ACTION_PICK,
                android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
        startActivityForResult(galeriaIntent, 1);

    }
});
```

Fonte: *print screen* de parte da classe Adicionarespecie.java.

A classe Adicionarespecie.java também executa uma importante ação que consta em adquirir a latitude e longitude que o dispositivo se encontra.

O primeiro passo é saber se o dispositivo permite que o aplicativo acesse as funções de localização do mesmo. Para isso é criado uma variável da classe LocationManager. Esta classe é padrão no Android e oferece informações a respeito das configurações de localização do dispositivo. Através do método isProviderEnabled é possível verificar sobre qual conexão a aplicação obterá a geolocalização, uma vez que os retornos destas funções são armazenados em variáveis booleanas. A criação destas variáveis para teste pode ser observada na Figura 6.26.

Figura 6.26 - Inicialização das variáveis booleanas

```
gps_enabled = locationManager
        .isProviderEnabled(LocationManager.GPS_PROVIDER);

network_enabled = locationManager
        .isProviderEnabled(LocationManager.NETWORK_PROVIDER);
```

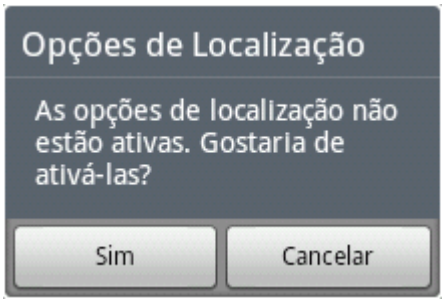
Fonte: *print screen* de parte da classe Adicionarespecie.java.

Se o dispositivo permitir a obtenção dos dados através do GPS é realizada uma requisição para a classe LocationManager passando como parâmetro o GPS_PROVIDER a fim de se obter a latitude e longitude precisa.

Se o dispositivo permitir a obtenção dos dados através da rede wireless que estives conectado é realizada uma requisição para a classe LocationManager passando como parâmetro o NETWORK_PROVIDER a fim de se obter a latitude e longitude próxima do ponto de acesso.

Caso nenhuma forma de obter a geolocalização estiver habilitada, é gerado um alerta como na Figura 6.27, que sugere ao usuário que habilite uma delas na tela de configuração do dispositivo.

Figura 6.27 - Mensagem para ativar a localização



Fonte: *print screen* do alerta de localização desativada.

6.4.2.7 Mapa.java

A API de mapas do Google precisa de uma classe que irá fornecer alguns parâmetros de inicialização do mapa, como o tipo, posição inicial, proximidade do zoom e demais configurações.

Na aplicação ModuloParque a classe Mapa.java faz estas configurações

iniciais conforme visto na Figura 6.28.

Figura 6.28 - Inicialização do mapa

```
GoogleMap map;
map.moveCamera(CameraUpdateFactory.newLatLngZoom(
        new LatLng(-17.795140, -43.799927), 20));
map.animateCamera(CameraUpdateFactory.zoomTo(10), 2000, null);
map.setMapType(GoogleMap.MAP_TYPE_HYBRID);
```

Fonte: *print screen* de parte da classe Mapa.java.

Foi criada uma variável da classe GoogleMap, a qual é estabelecida uma localização para o carregamento inicial, o zoom estabelecido em 10 e o tipo de mapa híbrido.

Um mapa híbrido possui tanto informações de rodovias, ruas, nomes de unidades de conservação, como também mostra o mapa com suas características de vegetação. Por isso este tipo foi escolhido para a aplicação ModuloParque.

Após abrir o mapa, é desenhado o contorno do Parque Nacional das Sempre-Vivas utilizando a função desenha_parque(). A função utiliza um vetor que possui as informações de latitude e longitude de 200 pontos no mapa obtidos do arquivo de limites do parque no site ICMBIO.

O primeiro passo é ler do arquivo String.xml os dados armazenados em um vetor de Strings chamado Parquelocalizacoes. A Figura 6.29 exibe esta chamada dos recursos.

Figura 6.29 - Acesso às coordenadas do paque

```
String[] latlong_parque_todo = getResources()
                    .getStringArray(R.array.Parquelocalizacoes);
```

Fonte: *print screen* de parte da classe Mapa.java.

Depois deste procedimento o vetor latlong_parque_todo possuirá 400 posições visto que foi armazenados os dados de forma contínua primeiro pela latitude e em sequência a longitude de um ponto específico. A estrutura do arquivo com os dados de latitude e longitude do parque pode ser observada na Figura 6.30.

Figura 6.30 - Vetor de latitudes e longitudes

strings.xml

```
20      <string-array name="Parquelocalizacoes">
21          <item>-17.9558790</item>
22          <item>-43.6713680</item>
23          <item>-17.9531440</item>
24          <item>-43.6694220</item>
25          <item>-17.9469360</item>
26          <item>-43.6612820</item>
27          <item>-17.9434490</item>
28          <item>-43.6610680</item>
29          <item>-17.9407020</item>
30          <item>-43.6578750</item>
31          <item>-17.9369910</item>
32          <item>-43.6580850</item>
33          <item>-17.9358770</item>
```

Fonte: *print screen* de parte do arquivo strings.xml.

A função desenha_parque() cria uma linha de espessura 2 e cor vermelha entre o primeiro ponto lido e o segundo ponto lido. Após isso é criada uma segunda linha entre o segundo ponto lido e o terceiro e assim sucessivamente. O procedimento pode ser observado na Figura 6.31.

Figura 6.31 - Função desenha_parque()

```
public void desenha_parque() {

    int tamanho = latlong_parque_todo.length;


    for(int i = 0; i <= tamanho-4; i++){

    Polyline line = map.addPolyline(new PolylineOptions()

            .add(new LatLng(Double.valueOf(latlong_parque_todo[i]),Double.valueOf(latlong_parque_todo[i+1])),
                 new LatLng(Double.valueOf(latlong_parque_todo[i+2]), Double.valueOf(latlong_parque_todo[i+3])))
            .width(2)
            .color(Color.RED));

    }
}
```

Fonte: Fonte: *print screen* de parte da classe Mapa.java.

A última funcionalidade fornecida por esta classe é colocar marcadores no mapa em tempo de execução com a função criar_marcadores_especies(). Esta função faz a leitura das espécies cadastradas no banco de dados e executa outra função chamada criar_marcador() passando como parâmetros a latitude, longitude um título que é o nome da espécie e um inteiro que representa a imagem do marcador. A Figura 6.32 exemplifica esta ação.

Figura 6.32 - Criação dos marcadores

```
helper = new Bdinterno(this);
bdinterno = helper.getReadableDatabase();
cursor = bdinterno.rawQuery("Select * from  Especies", null);

cursor.moveToFirst();
while (!cursor.isAfterLast()) {
    try {

        criar_marcador(cursor.getString(2), cursor.getString(3),
                cursor.getString(0), R.drawable.marcador_verde);

        cursor.moveToNext();
    } catch (Exception e) {
        Toast.makeText(getApplicationContext(),
                "Erro na criação de algum marcador", Toast.LENGTH_SHORT)
                .show();
        break;

    }
}
```

Fonte: *print screen* de parte da classe Mapa.java.

Deste modo são criados todos os marcadores de espécies e fica fácil para o usuário perceber em os lacais de incidências de tais espécies.

Ao clicar em um marcador é executado uma função semelhante à da classe ListaEspecies.java pois é passado por Intent os dados do marcador clicado e abre em uma caixa de diálogo as informações da espécie. A Figura 6.33 representa um exemplo das informações que serão exibidas na caixa de diálogo.

Figura 6.33 - Menu Principal do Software ModuloParque

Fonte: *print screen* da dialog aberta ao clicar sobre um marcador.

6.4.2.8 Infoparque.java

Esta classe é chamada ao clicar no botão Informações do menu principal. É mostrada uma tela com um breve texto com informações sobre o parque e fotos.

6.4.3 Bibliotecas utilizadas

Visando uma padronização e também utilizar serviços do Google, algumas bibliotecas foram incorporadas à aplicação que foram a SherlockActionBar e o GooglePlayServices.

6.4.3.1 Sherlock Action Bar

Uma barra de ação, ou mais comumente chamadas de ActionBar, consiste em uma barra de ações que tornou-se padrão no Android. Esta barra possui o ícone da aplicação, um título, que pode ser o nome da aplicação, e ícones que vão exercer alguma ação como podemos ver na figura 2.4.3.1. A Figura 6.34 representa uma Action Bar.

Figura 6.34 - Uma actionBar incluindo [1] o ícone da aplicação, [2] dois ícones de ação e um ícone overflow.

Fonte: Site Google Developers[11].

As primeiras ActionBar apareceram no Android 3.0 o que ocasiona uma disparidade entre a forma que uma aplicação se comporta em dispositivos de versões de Android diferentes.

Buscando a padronização foi utilizado no software ModuloParque a biblioteca SherlockActionBar. É um projeto OpenSource que permite o fácil desenvolvimento

[11] Disponível em: <http://developer.android.com/guide/topics/ui/actionbar.html> Acesso em jun 2014.

de uma aplicação com uma barra de ação para todas as versões do Android 2.x para cima.

6.4.3.2 Google Play Services

É um conjunto de bibliotecas e APIs disponibilizadas pelo Google para oferecer serviços da mesma forma para diferentes versões do Android. O uso desta biblioteca foi necessário para incorporar ao aplicativo o serviço de mapas do Google.

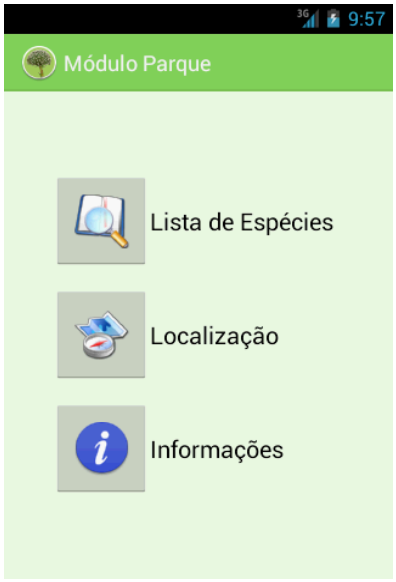
6.4.4 Interfaces de Usuário

O software ModuloParque possui basicamente cinco telas que exercem a iteração com o usuário.

6.4.4.1 Tela do Menu Principal

É uma tela com o menu principal, na qual o usuário têm acesso a partir de somente um único clique três das funções principais do aplicativo que consistem em mostrar a lista de espécies cadastradas, mostrar o mapa iterativo e mostrar as informações do parque. A tela do Menu Principal pode ser observada na Figura 6.35.

Figura 6.35 - Menu Principal do Software ModuloParque

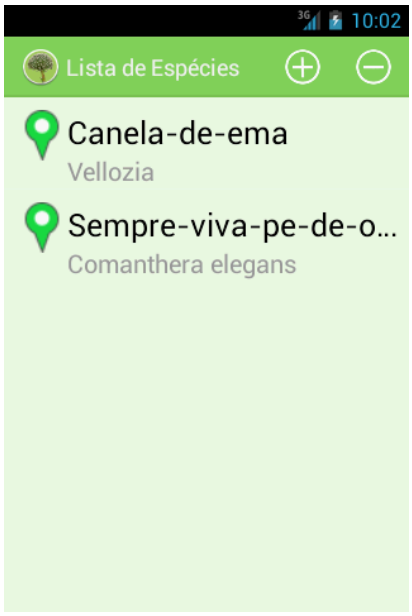


Fonte: *print screen* da tela Menu Principal.

6.4.4.2 Tela de Lista de Espécies

Ao clicar no botão Lista de Espécies, o usuário é direcionado para uma tela que contém uma lista preenchida com as espécies cadastradas. Cada item da lista é composto pelo nome da espécie e o nome científico associado. A Figura 6.36 exemplifica esta tela.

Figura 6.36 – Tela de Lista de Espécies



Fonte: *print screen* da tela da Lista de Espécies.

6.4.4.3 Tela de Espécie

Ao clicar sobre um dos itens da lista o usuário é direcionado para a tela da espécie clicada, a qual mostra as informações juntamente com a imagem cadastrada para aquela espécie ou a foto que foi tirada que foi armazenada no momento do cadastro. A Figura 6.37 mostra a Tela de Espécie.

Figura 6.37 - Tela de espécie

Fonte: *print screen* da tela Espécie.

6.4.4.4 Tela do Mapa

É a tela que irá conter os marcadores e que permite uma melhor experiência de usuário por ser algo iterativo e dinâmico. A Figura 6.38 mostra a Tela do Mapa.

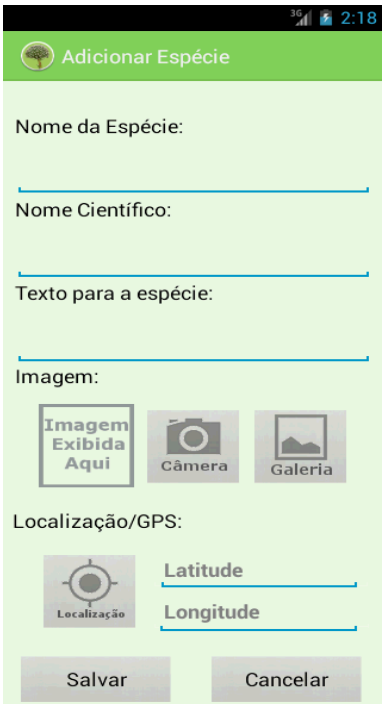
Figura 6.38 - Tela do Mapa

Fonte: *print screen* da tela Espécie.

6.4.4.5 Tela de Adicionar Espécie

É uma tela que fornece um pequeno formulário para que o usuário insira uma nova espécie na aplicação. O usuário deve informar o nome da espécie, nome científico, um texto que define a espécie, selecionar uma imagem da galeria ou da câmera e inserir de maneira automática ou manual os dados de latitude e longitude que a espécie foi encontrada. A Figura 6.39 exibe a Tela de Adicionar Espécie.

Figura 6.39 - Tela de Adicionar Espécie



Fonte: *print screen* da tela de Adicionar Espécie.

7 CONCLUSÕES E TRABALHOS FUTUROS

O desenvolvimento do presente trabalho trouxe uma significativa contribuição para o Parque Nacional das Sempre-Vivas uma vez que irá servir de apoio à realização de pesquisas relacionadas ao mesmo.

O crescente uso de smartphones e a ascensão do sistema operacional Android motivou ainda mais o desenvolvimento desta ferramenta visto que no site do Google Android Developers existe uma grande coletânea de documentação das bibliotecas e exemplos de implementação para os diversos tipos de situações que podem ocorrer.

Ao final do trabalho como resultado temos o software ModuloParque que funciona em plataformas Android com sistema superior a versão 2.2. Adquirindo a geolocalização e marcando em um mapa para ser posteriormente visualizado, percebe-se que existem inúmeras inferências que podem ser realizadas a partir destas informações.

Como trabalhos futuros podemos ter uma nova versão do software com marcações de outros tipos, como possíveis focos de incêndio, incidências, catalogação de futuros pontos turísticos, entre outras melhorias.

REFERÊNCIAS

TANENBAUM, Andrew S. **Sistemas operacionais: projeto e implementação** / Andrews S. Tanenbaum e Albert S. Woodhull; trad. Edson Furmankiewicz. 2. Ed. Porto Alegre; Bookman, 2000.

NANJI, Ayaz. **Web Traffic From Mobile Devices Up 78% Year Over Year.** Disponível em: <http://www.marketingprofs.com/charts/2013/11010/web-traffic-from-mobile-devices-up-78-year-over-year> Acesso em: 12 jun. 2014.

MICROSOFT. **O que é um driver?** Disponível em: < http://windows.microsoft.com/pt-br/windows/what-is-driver#1TC=windows-7> Acesso em: 05 jul. 2014.

DEITEL, H.M. **C++: como programar**/H.M. Deitel e P.J. Deitel trad. Carlos Arthur Lang Lisbôa e Maria Lúcia Lang Lisbôa. - 3.ed. - Porto Alegre : Bookman, 2001.

MULLER, Leonardo. **Android e iOS juntos representaram 93,8% das vendas de smartphones em 2013**. Disponível em: <http://www.tecmundo.com.br/celular/50290-android-e-ios-juntos-representaram-93-8-das-vendas-de-smartphones-em-2013.htm> Acesso em 15 jun. 2014.

LECHETA, Ricardo R. **Google Android: aprenda a criar aplicações para dispositivos móveis com o Android SDK**. Novatec, 2010. ISBN 978-85-7522-244-7.

LUZZI, Luciano**. Android – persistência de dados usando sq-lite**, 2014. Disponível em: <http://www.mobiltec.com.br/blog/index.php/android-persistencia-de-dados-usando-sqlite/>. Acesso em 25 jan. 2014.

LAUREANO, Marcos Aurelio Pchek. **Máquinas Virtuais e Emuladores – Conceitos, Técnicas e Aplicações.** 1 ed. Novatec, 2006.

DATAFOLHA. **43 milhões de brasileiros acessam internet por dispositivos móveis**. Disponível em: <http://datafolha.folha.uol.com.br/mercado/2014/01/1400618-43-milhoes-de-brasileiros-acessam-internet-por-dispositivos-moveis.shtml> Acesso em 6 jul. 2014.

BORATTI, Isaias Camilo**. Programação Orientada a Objetos em Java.** Florianópolis. Visual Books, 2007. 316p.