DIEGO D'LEON NUNES
DIÓGENES APARECIDO REZENDE

APLICATIVO PARA CONSULTA DE NOTAS

UNIVERSIDADE DO VALE DO SAPUCAÍ
POUSO ALEGRE – MG
2015

SUMÁRIO

1 QUADRO TEÓRICO

Neste capítulo serão descritos os principais conceitos e características das tecnologias utilizadas para o desenvolvimento dos *softwares* propostos nos objetivos deste trabalho.

1.1 *Java*

Segundo Deitel e Deitel (2003) o *java* veio ao público em 1995 pela *Sun Microsystem*. Os criadores dessa nova tecnologia liderados por James Gosling basearam-se em duas linguagens muito utilizadas no mundo, C e C++. Isso deu ao java uma base para implementar em novos sistemas como, sistemas operacionais, sistemas de comunicações, sistema de banco de dados e aplicativos para computadores pessoais.

Entre as principais características pode-se dizer que o Java é:

- Orientado a objeto.
- Seguro.
- Independente de plataforma.

Deitel e Deitel (2003) o *java* será utilizado para codificar e criar regras para proteger o banco de dados, gerenciando a infraestrutura que o próprio java fornece como, Transação, Acesso remoto, Web services, Gerenciamento de threads, Gerenciamento de conexões http[1].

Para acessar o servidor precisamos de um serviço *web* (*Web services*) que utilizará o *java* para criar as regras e serviços de compilação de dados. Segundo Deitel e Deitel (2003) existem cinco fases para que os dados cheguem em seu dispositivo que são, armazenamento em disco, o compilador cria os *bytecodes*, transfere-os para memória, verifica a sua integridade, para que não haja nenhuma violação das restrições de segurança e por ultimo o interpretador(JMV[2]) lê os *bytecodes* e os traduz para a linguagem que o computador entenda e possivelmente armazena os valores dos dados enquanto executa o programa.

Romanato (2015) afirma que o *Java* usa a JMV, que é uma máquina virtual capaz de converter os *bytecodes* para a linguagem do sistema operacional utilizado pelo cliente, sem a

[1] HTTP - HyperText Transfer Protocol.
[2] JVM - Java Virtual Machine

necessidade de compilá-lo para cada plataforma. Dessa maneira um *software* que foi desenvolvido para um sistema operacional, funcionará normalmente em qualquer outro.

1.2 *Android*

Segundo Monteiro (2012), *Android* é um sistema operacional baseado em *Linux*, de código aberto e que utiliza a linguagem de programação *Java* para o desenvolvimento de seus aplicativos. Criado especialmente para dispositivos móveis, começou a ser desenvolvido no ano de 2003 pela então empresa Android Inc, que em 2005 foi agregada ao Google. A partir de 2007 o projeto *Android* uniu-se a *Open Handset Alliance*, uma associação de empresas de *softwares*, *hardwares* e telecomunicações, que tem por finalidade desenvolver uma plataforma para dispositivos móveis que seja completa, aberta e gratuita.

Krazit (2009) publicou uma entrevista com Rubin, um dos idealizadores do *Android*, o qual afirma que o sistema pode rodar em equipamentos de diversos fabricantes, evitando assim ficar limitado a poucos dispositivos. Conforme informações do site Android (2015a), hoje em dia existe mais de um bilhão de aparelhos espalhados pelo mundo com esse sistema operacional.

De acordo com Monteiro (2012), as aplicações são executadas em uma máquina virtual Java denominada *Dalvik*. Cada aplicativo, usa uma instância dessa máquina virtual tornando-o assim mais seguro. Por outro lado, os *softwares* só podem acessar os recursos do dispositivo, como uma lista de contatos, caso seja formalmente aceito pelo usuário nos termos de uso ao instalá-lo.

As configurações de uma aplicação na plataforma *Android* ficam salvas em um arquivo XML denominado AndroidManifest.xml que se localiza na raiz do projeto. Para Lecheta (2010) as informações devem estar entre tags correspondentes ao recurso. De forma, que o arquivo se inicie pela tag <manifest> e se encerre por </manifest>. Com necessidade de acessar a internet é preciso declarar a permissão da seguinte forma <uses-permission android:name="android.permission.internet"/>.

Lecheta (2010) diz que as *intents* podem ser consideradas como o coração do *Android*, uma vez que essa classe está presente em todos os lugares de uma aplicação. De acordo com K19 (2012, p.29), "são objetos responsáveis por passar informações, como se fossem mensagens, para os principais componentes da API do *Android*, como as *Activities*, *Services* e *Broad-Cast Receivers*". Monteiro (2012) diz que as *Intents* são criadas quando se tem a intenção de realizar algo como por exemplo compartilhar uma imagem, utilizando os recursos já existentes

no dispositivo. Existem dois tipos de Intents:

- *Intents* implícitas - Quando não é informada qual *Activity* deve ser chamada, ficando assim por conta do sistema operacional verificar qual a melhor opção.
- *Intents* explicitas - Quando é informada qual *Activity* deve ser chamada. Usada normalmente para chamar *activities* da mesma aplicação.

Segundo K19 (2012), uma aplicação *Android* pode ser construída com quatro tipos de componentes: *Activity*, *Services*, *Content Providers* e *Broadcast Receivers*.

As *activities* são as telas com interface gráfica, que permitem interações com os usuários. De acordo com Lecheta (2013), cada *activity* tem um ciclo de vida, uma vez que ela pode estar sendo executada, estar em segundo plano ou totalmente destruída.

Toda vez que é iniciada uma *activity*, ela vai para o topo de uma pilha denominada *activity stack*. O bom entendimento do ciclo de vida é importante, pois quando uma aplicação é interrompida é possível salvar as informações ou ao menos voltar ao estágio a qual o usuário se encontrava.

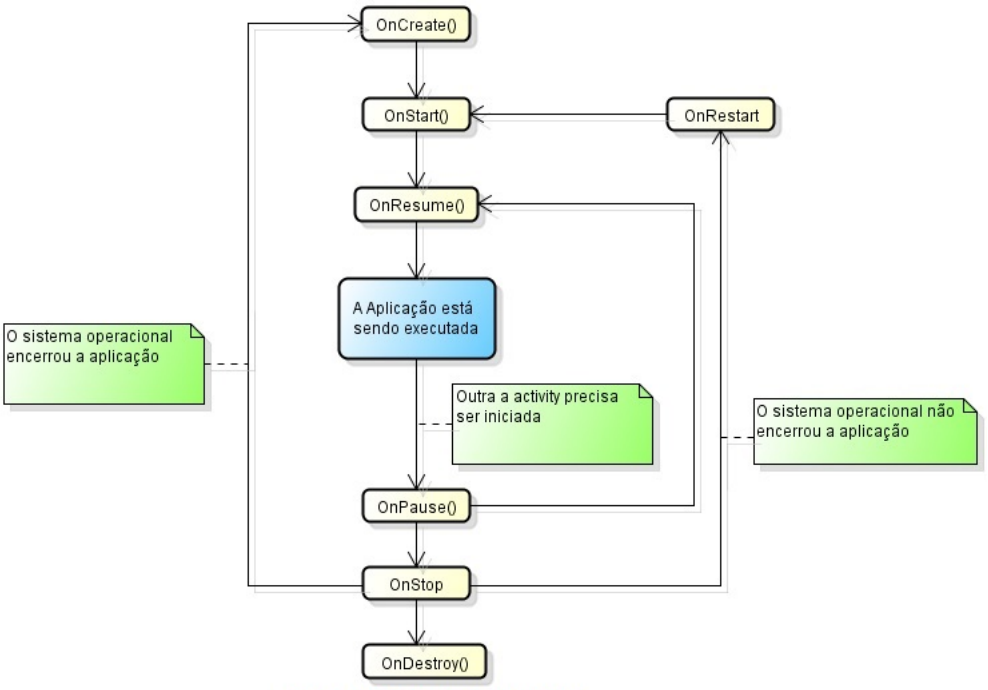


Figura 1 – Ciclo de Vida de uma *Activity*. **Fonte:**Lecheta (2010)

Na Figura 1 é demostrado um exemplo de ciclo de vida de uma *activity*. Para que se possa entender melhor imagina- se o seguinte cenário: Um usuário entra no aplicativo de notas da Univás. Para que a *activity* seja criada é chamado o método *OnCreate()*, logo após é executado o método *OnStart()* e ao finalizar do ciclo anterior é chamado o *OnResume()*, só

a partir de então, a *activity* é visualizada pelo usuário. Contudo, durante a navegação o aluno recebe uma ligação. Nessa hora o sistema operacional chama o método *OnPause()* para pausar a aplicação e abrir uma outra *activity* para que o usuário possa atender a chamada telefônica. É possível nesse método salvar informação da qual o usuário está utilizando. Ao concluir o método de pausa é executado o método *OnStop()*, a partir de agora a *activity* da Univás não será mais visível para o usuário. Ao encerrar a ligação, há dois caminhos possíveis de se percorrer, o primeiro, é o caso do sistema operacional não encerrar completamente a aplicação da memória, dessa forma será chamado o método *OnRestart()* do aplicativo de notas e voltar de onde o usuário estava, porém se foi necessário encerrar completamente a aplicação, devido à falta de memória, será necessário chamar o método *OnCreate()* novamente. Por fim quando o usuário sair da aplicação é chamado o método *OnDestroy()* encerrando se assim o ciclo de vida.

No arquivo AndroidManifest.xml as *activities* devem estar entre as tags <*activity*> <*/activity*> e a *activity* principal, ou seja, pela qual será iniciada a aplicação deve conter a tag <*intente-filter*> além de <*action android:name="android.intent.action.MAIN"*/> indicando que essa atividade deverá ser chamada ao iniciar a aplicação e <*category android:name="android.intent.category.LAUNCHER"*/> que implica que esse APP ficará disponível junto aos outros aplicativos no dispositivo.

A *Activity* a ser utilizada para iniciar a aplicação é uma *Navigation Drawer*. Segundo o site Android (2015b) ela exibe do lado esquerdo as principais funções do *software*, semelhante a um menu. Fica normalmente escondida aparecendo apenas quando se clica no canto superior esquerdo.

Segundo Lecheta (2010), a classe *Services* existe com intuito em executar processos que levarão um tempo indeterminado para serem executados e que normalmente consomem um alto nível de memória e processamento. Esses processos são executadas em segundo plano enquanto o cliente realiza outra tarefa. Assim um usuário pode navegar na internet enquanto é feito um *download*. O serviço é geralmente iniciado pelo *Broadcast Receiver* e quem o gerencia é o sistema operacional que só o finalizará ao concluir a tarefa, salvo quando o espaço de memória é muito baixo.

Para Lecheta (2010), a função da classe *Content Provider* é prover conteúdos de forma pública para todas as aplicações, dessa forma usando-se essa classe é possível as aplicações consultar, salvar, deletar e alterar informações no *smartphone*. Assim afirma Lecheta (2010, p.413) "O *Android* tem uma série de provedores de conteúdo nativos, como, por exemplo, consultar contatos da agenda, visualizar os arquivos, imagens e vídeos disponíveis no celular". Portanto, um contato pode ser salvo por um aplicativo e alterado por outro.

Para Lecheta (2010), a classe *Broadcast Receiver* é muito importante para a plataforma *Android*, uma vez que ela é responsável por agir em determinados eventos de uma *intent*.

Essa classe sempre é executada em segundo plano, de forma que o usuário não veja o que está sendo executado. Portanto, quando uma pessoa está utilizando uma aplicação e recebe uma mensagem de SMS, o *Broadcast Receiver* capta essa informação e a processa sem a necessidade do cliente ter que parar de realizar suas tarefas.

A configuração de um *Broadcast Receiver* é escrita pela tag <*receiver*> junto com a tag <*intente-filter*> que conterá o que deve ser feito dentro do AndroidManifest.xml.

Em uma aplicação, um elemento fundamental é a interface gráfica, que deverá ser organizada, simples e elegante. Conforme Monteiro (2012) esses são os principais *Layouts* do sistema operacional Android:

- *LinearLayout* - Permite posicionar os elementos em forma linear, dessa forma quando o dispositivo estiver em forma vertical os itens ficaram um abaixo do outro e quando estiver na posição horizontal eles ficaram um ao lado do outro.

- *RelativeLayout* - Permite posicionar elementos de forma relativa, ou seja um item com relação a outro.

- *TableLayout* - Permite criar *layouts* em formato de tabelas. O elemento *TableRow* representa uma linha da tabela e seus filhos as células. Dessa maneira, caso um *TableRow* possua dois itens significa que essa linha tem duas colunas.

- *DatePicker* - *Widget* desenvolvido para a seleção de datas que podem ser usadas diretamente no *layout* ou através de caixas de diálogo.

- *Spinner* - *Widget* que permite a seleção de itens.

- *ListViews* - Permite exibir itens em uma listagem. Dessa forma, em uma lista de compras, clicando em uma venda é possível listar os itens dessa venda selecionada.

- *Menus* - Um item muito importante, pois apresenta aos usuários as opções existentes no aplicativo.

- *AlertDialog* - Apresenta informações aos usuários através de uma caixa de diálogo. Comumente utilizado para perguntar ao cliente o que deseja fazer quando seleciona algum elemento.

- *ProgressDialog* e *ProgressBar* - Utilizado quando uma aplicação necessita de um recurso que será demorado, como por exemplo, fazer um *download*, pode ser feito uma animação informando ao usuário o progresso da operação.

Para uma maior interação, as aplicações normalmente utilizam API's de terceiros, como o Google *Maps*, quando necessita encontrar alguma localização. Para Monteiro (2012) essa comunicação pode utilizar o REST[3], que envia requisições através da URL. Ao receber informações pedidas a um outro serviço, ela pode estar no padrão XML ou JSON. O REST será detalhado mais adiante.

Outra ferramenta importante e muito utilizada do Android é a Notificação. Segundo Phillips e Hardy (2013), quando uma aplicação está sendo executada em segundo plano e necessita comunicar-se com o usuário, o aplicativo cria uma notificação. Normalmente as notificações aparecem na barra superior, o qual pode ser acessado arrastando para baixo a partir da parte superior da tela. Assim que o usuário clica na notificação ela cria uma *activity* abrindo a aplicação em questão.

Com a ideia de desenvolver um aplicativo para dispositivos móveis, a plataforma Android foi escolhida devido ao seu destaque no mercado e pela facilidade que apresenta aos usuários.

1.3 Android Studio

Umas das ferramentas mais utilizadas para o desenvolvimento em Android é o *Eclipse IDE*, contudo a Google criou um *software* especialmente para esse ambiente, chamado *Android Studio*. Segundo Gusmão (2014), *Android Studio* é uma IDE baseado no ItelliJ *Idea* e foi apresentado na conferência para desenvolvedores I/O de 2013.

De acordo com Hohensee (2013), o *Android Studio* tem um sistema de construção baseado em *Gradle*, que permite aplicar diferentes configurações no código quando há necessidade de criar mais de uma versão, como por exemplo, um *software* que terá uma versão gratuita e outra paga, melhorando a reutilização do código. Com o *Gradle* também é possível fazer os *downloads* de todas as dependências de uma forma automática sem a necessidade de importar bibliotecas.

[3] REST - *Representational State Transfer* ou Transferência de Estado Representativo.

Hohensee (2013) afirma que o *Android Studio* é um editor de código poderoso, pois tem como característica a edição inteligente, pois ao digitar já completa as palavras reservadas do sistema operacional e fornece uma organização do código mais legível.

Segundo Android (2015c), a IDE tem suporte para a edição de interface, o que possibilita ao desenvolvedor arrastar os componentes que deseja. Ao testar o aplicativo permite o monitoramento do consumo de memória e de processador por parte do utilitário.

Gusmão (2014) diz que a plataforma tem uma ótima integração com o *GitHub* e está disponível para *Windows*, *Mac* e *Linux*. Além disso os programadores terão disponíveis uma versão estável e mais três versões que estarão em teste chamadas de *Beta*, *Dev* e *Canary*.

Devido ao *Android Studio* ser uma ferramenta de fácil usabilidade e a IDE oficial para o desenvolvimento Android, esta foi escolhida como ambiente de construção do aplicativo.

1.4 *Web Services*

Nos tempos atuais, com o grande fluxo de informação que percorre pelas redes da *internet* é necessário um nível muito alto de integração entre as diversas plataformas, tecnologias e sistemas. Como uma provável solução para esse ponto, já existem as tecnologias de sistemas distribuídos. Porém essas tecnologias sofrem demasiadamente com o alto acoplamento de seus componentes e também com a grande dependência de uma plataforma para que possam funcionar. Com intuito de solucionar a estes problemas e proporcionar alta transparência entre as várias plataformas, foram criados as tecnologias *web services*.

De acordo com Erl (2015):

> No ano de 2000, a W3C (*World Wide Web Consortium*) aceitou a submissão do *Simple Object Access Protocol* (SOAP). Este formato de mensagem baseado em XML estabeleceu uma estrutura de transmissão para comunicação entre aplicações (ou entre serviços) via HTTP. Sendo uma tecnologia não amarrada a fornecedor, o SOAP disponibilizou uma alternativa atrativa em relação aos protocolos proprietários tradicionais, tais como CORBA e DCOM.

Considera-se então a existência dos *web services* a partir daí. De acordo com Durães (2005), *Web Service* é um componente que tem por finalidade integrar serviços distintos. O que faz com que ele se torne melhor que seus concorrentes é a padronização do XML (*Extensible Markup Language*) para as trocas de informações. A aplicação consegue conversar com o servidor através do WSDL que é documento que contém as regras de funcionamento do *web service*.

Segundo Coulouris et al. (2013), “Um serviço *Web* (*Web service*) fornece uma interface de serviço que permite aos clientes interagirem com servidores de uma maneira mais geral do que acontece com os navegadores *Web*”. Ainda de acordo com Coulouris et al. (2013) os clientes (que podem ser desde um navegador até mesmo outro sistema) acessam serviços *Web* fazendo uso de requisições e respostas formatadas em XML e sendo transmitidos pelo uso do protocolo HTTP. O uso dessas tecnologias tende a facilitar acomunicação entre as diversas plataformas, e atende de uma forma melhor que as técnologias já existentes. Porém, para que haja uma interação transparente e eficaz, entre as diversas plataformas, é necessário uma infraestrutura um pouco mais complexa para integrar todas essas tecnologias. Essa infraestrutura é composta pelas tecnologias já citadas e por outros componentes essenciais para disponibilização de serviços *web*, como mostra a Figura 2.

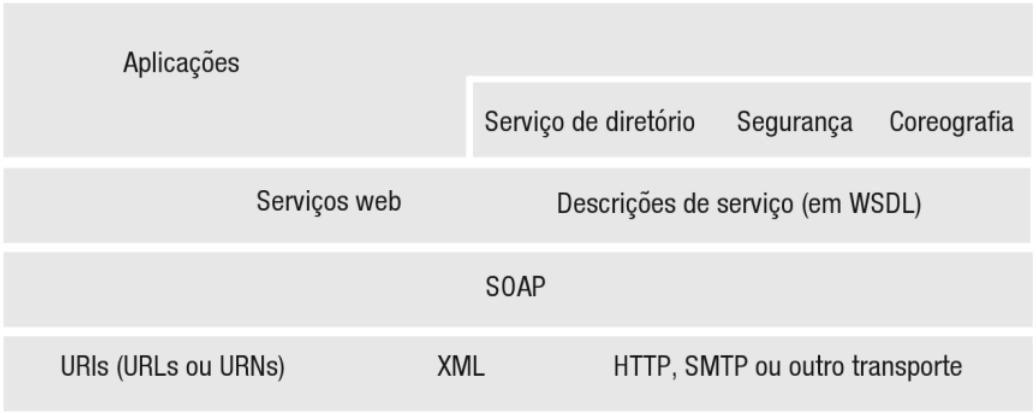


Figura 2 – Infraestrutura e componentes dos serviços *web*. **Fonte:**Coulouris et al. (2013)

Os *web services* geralmente fazem uso do protocolo SOAP, para estruturar e encapsular as mensagens trocadas. De acordo com Coulouris et al. (2013, p.381), "O protocolo SOAP(*Simple Object Access Protocol*) é projetado para permitir tanto interação cliente-servidor como assíncrona pela *Internet*". Segundo Sampaio (2006, p.27), "O SOAP foi criado inicialmente, para possibilitar a invocação remota de métodos através da internet". As mensagens SOAP possuem um elemento Envelope. De acordo com Saudate (2013, p.19), "O elemento Envelope é puramente um *container* para os elementos *Header* e *Body*". Ainda de acordo com Saudate (2013), o elemento *header* transporta metadados relativos à requisição tais como autenticação, endereço de retorno da mensagem, etc. Já o elemento *body* carrega o corpo da requisição, que nada mais é do que o nome da operação e paramêtro referentes à mesma. É valido que todas requisições do trocadas usando SOAP usam o XML com formato oficial.

Os *web services* além de fornecerem uma padronização de comunicação entre as várias tecnologias existentes, provêem transparência na troca de informações. Isso contribui pelo fato

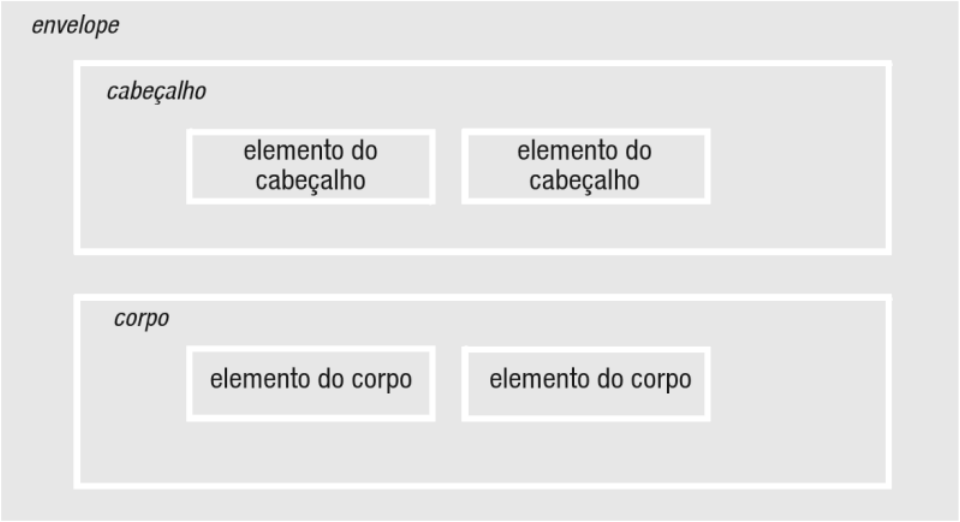


Figura 3 – Esquema de envelope SOAP. **Fonte:**Coulouris et al. (2013)

de que as novas aplicações possam se comunicar com aplicações mais antigas ou aplicações contruídas sobre outras plataformas.

Além das tecnologias *web services* tradicionais, existe também os *web services* REST que também disponibilizam serviços, porém não necessitam de encapsulamento de suas mensagens assim como os *web Services* SOAP. Este fato influencia diretamente na performance da aplicação como um todo, haja vista que não sendo necessário o encapsulamento da informação requisitada ao *web service*, somente é necessário o processamento e tráfego da informação que realmente importa. As caracteristícas do padrão REST serão abordadas a seguir.

1.4.1 REST

Segundo Saudate (2012), REST é a sigla de *Representational State Transfer*, desenvolvido por Roy Fielding na defesa de sua tese de doutorado. Segundo o próprio Fielding (2000) REST é um estilo que deriva do vários estilos arquitetónicos baseados em rede e que combinado com algumas restrições, fornecem uma interface simples e uniforme para fornecimento de serviços[4].

Rubbo (2015) afirma que os dados e funcionalidade de um sistema são considerados recursos e podem ser acessados através das URI's *(Universal Resource Identifier)*, facilitando dessa forma a comunicação do servidor com o cliente. Um serviço baseado na arquitetura REST basea-se fortemente em recursos. Para exemplificar o que seria um recurso em REST têm-se o seguinte cenário: considera-se uma URL tal como http://www.univas.edu.br/alunos/ onde

[4] Tradução e resumo de informações de responsabilidade dos autores da pesquisa.

pretende-se fazer a gerência dos dados de um ou de um conjunto de alunos. A recuperação de dados, bem como sua edição e/ou deleção fazendo uso pleno dos métodos HTTP, através dessa URL pode ser considerado como um recurso.

Saudate (2012), explica ainda que os métodos do HTTP podem fazer modificações nos recursos, da seguinte forma:

- GET – Para recuperar algum dado.
- POST – Para criar algum dado.
- PUT – Para alterar algum dado.
- DELETE – Para excluir algum dado.

Como o próprio Fielding (2000) também foi um dos criadores de um dos protocolos mais usados na web, o HTTP, pode-se dizer que o REST foi concebido para rodar sobre esse protocolo com a adição de mais algumas caracteristicas que segundo Saudate (2013) foram responsáveis pelo sucesso da web. Essas caracteristicas são:

- URLs bem definidas para recursos;
- Utilização dos métodos HTTP de acordo com seus propósitos;
- Utilização de *media types* efetiva;
- Utilização de *headers* HTTP de maneira efetiva;
- Utilização de códigos de status HTTP;
- Utilização de Hipermídia como motor de estado da aplicação.

Segundo Godinho (2009), não há um padrão de formato para as trocas de informações, mas as que mais são utilizadas é o XML[5] e o JSON[6]. O REST é o mais indicado para aplicações em dispositivos moveis, devido sua agilidade.

[5] XML - *Extensible Markup Language*.
[6] JSON - *JavaScript Object Notation*.

1.5 *Apache Tomcat*

De acordo com Tomcat (2015), *Apache Tomcat* é uma implementação de código aberto das especificações *Java Servlet* e *JavaServer Pages*. O *Apache Tomcat* é um *Servlet Container*, que disponibiliza serviços através de requisições e respostas. Caelum (2015) afirma que ele utilizado para aplicações que necessitam apenas da parte *Web* do Java EE[7].

Segundo Tomcat (2015), o projeto desse *software* começou com a *Sun Microsystems*, que em 1999 doou a base do código para *Apache Software Foundation*, e então seria lançada a versão 3.0.

Conforme Devmedia (2015), para o desenvolvimento usando o código livre com *Tomcat* é necessária a utilização das seguintes linguagens:

- JAVA - É utilizado em toda parte lógica da aplicação.
- HTML - É utilizado na parte de interação com o usuário.
- XML - É utilizado para as configurações do software.

Desta forma, o cliente envia uma requisição através do seu navegador, o servidor por sua vez a recebe, executa o *servlet* e devolve a resposta ao usuário.

1.6 PostgreSQL

Segundo Milani (2008), o postgreSQL é um SGBD (Sistema de Gerenciamento de Banco de Dados) que tem suporte para ACID (Atomicidade, consistência, isolamento e Durabilidade), que são serviços que garantem a qualidade que um banco de dados. A seguir algumas das principais características e recursos existentes no postgreSQL, que são, Replicação, Cluster (Alta disponibilidade), Multithreads, segurança ssl e criptografia.

Segundo Milani (2008):

- Replicação: É o compartilhamento de processos e distribuição das informações em diferentes bancos de dados. Ou seja as informações que serão armazenadas no servidor serão replicadas para um servidor secundário, mantendo os dados íntegros.
- Cluster: É a interligação de dois ou mais computadores e a sincronização entre eles, assim aumentando a capacidade de demanda do banco de dados.

[7] EE - Sigla para enterprise edition

- Multithreads: É a manipulação de dados de forma que mais de uma pessoa tenha acesso a mesma informação sem ocasionar atrasos ou filas de acessos.

- Segurança SSL[8]e criptografia: Possibilita criar conexões seguras, tanta para trafegar informações de login quando aquelas consideradas sigilosas.

Stones e Matthew (2005) dizem que um dos pontos fortes do postgreSQL deriva-se de sua arquitetura onde pode ser usado em um ambiente cliente/servidor, beneficiando o usuário e o desenvolvedor.

Segundo Stones e Matthew (2005), postgreSQL é comparado com qualquer outro SGBD, contendo todas as características que encontraria em outro banco de dados, e algumas características que não encontra em outro *software* como, transações, subconsultas, chave estrangeira e regras de herança.

O PostgreSQL é um *software* de fácil utilização e multiplataforma o que leva a ser implantado em muitas empresas.

1.7 Engenharia de *Software*

De acordo com Carvalho e Chiossi (2001), a engenharia de *software* surgiu na década de 80, com intuito de melhorar o desenvolvimento de *software*, produzindo sistemas de alta qualidade com a redução do custo e do tempo.

Segundo Pressman (2011, p.39), engenharia de *software* é "o estabelecimento e o emprego de sólidos princípios da engenharia de modo a obter *software* de maneira econômica, que seja confiável e funcione de forma eficiente em máquinas reais".

Como afirmam Carvalho e Chiossi (2001), a engenharia possui modelos de processos que possibilitam ao gerente controlar o desenvolvimento e aos programadores uma base para produzir. Alguns desses paradigmas são:

- Ciclo de vida clássico - Utiliza o método sequencial, em que o final de uma fase é o início da outra.

- O paradigma evolutivo - Baseia-se no desenvolvimento e implementação de um produto inicial. Esse produto passa por críticas dos usuários e vai recebendo melhorias e versões até chegar ao produto desejado.

[8] SSL-*Secure Socket Layer*

- O paradigma espiral - Engloba as melhores características do ciclo de vida clássica e o paradigma evolutivo. Ele consiste em vários ciclos e cada ciclo representa uma fase do projeto.

De toda a engenharia de *software*, o que mais será utilizado nesse projeto é a linguagem UML, que através dos seus diagramas norteará os caminhos a serem seguidos.

1.7.1 UML

De acordo com Booch, Rumbaugh e Jacobson (2012) "A UML (*Unified Modeling Language*) é uma linguagem-padrão para a elaboração da estrutura de projetos de *software*". Na decada de 80 seguindo o surgimento e evolução das linguagens de programação orientadas a objetos, foram surgindo as linguagens de modelagens orientadas a objetos, como um modo alternativo de análise e projeto de *software* que era usada na época. De acordo com Guedes (2011, p.19):

> A UML surgiu da união de três métodos de modelagem: o método de Booch, o método OMT (*Object Modeling Technique*) de Jacobson, e o método OOSE (*Object-Oriented Software Engineering*) de Rumbaugh. Estes eram, até meados da década de 1990, os métodos de modelagem orientada a objetos mais populares entre os profissionais da área de desenvolvimento de *software*. A união desses métodos contou com o amplo apoio da *Rational Software*, que a incentivou e financiou.

Segundo Booch, Rumbaugh e Jacobson (2012, p.13) "A UML é independente de processo, apesar de ser perfeitamente utilizada em processo orientado a casos de usos, centrado na arquitetura, interativo e incremental". A linguagem de modelagem UML além de fornecer um vocabulário próprio, também provê uma série de diagramas que tem inúmeras finalidades diferentes. Tais finalidades e suas subdivisões estão descritas na Figura 4.

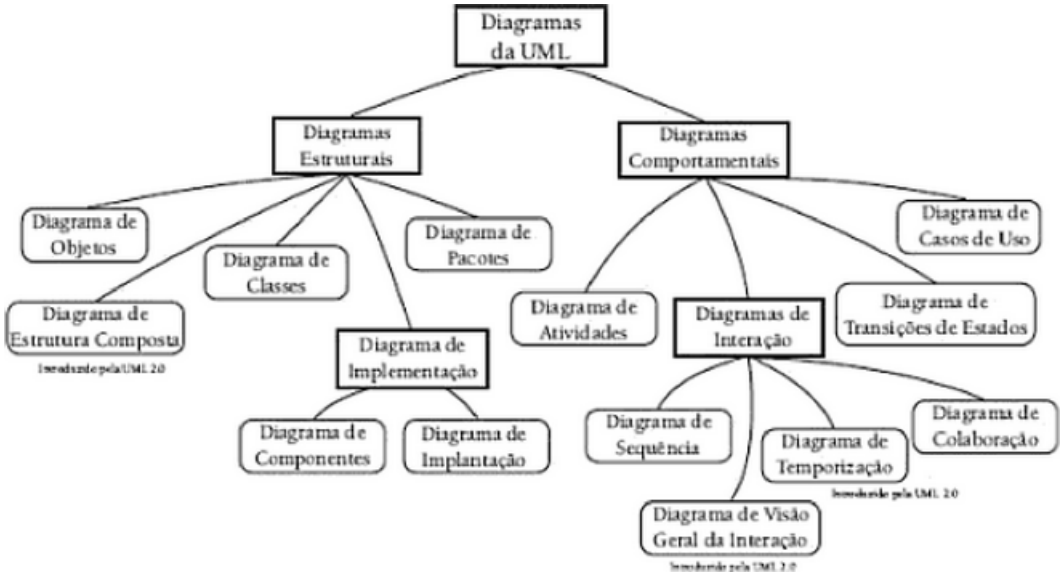


Figura 4 – Diagramas definidos pela UML. **Fonte:**Bezerra (2015)

A linguagem de modelagem UML não é um processo rígido e permite uma adequação de acordo com a situação do projeto em que é aplicada. Por permitir essa flexibilidade e prover suporte adequado para determinados casos de um projeto, é que a linguagem de modelagem UML será usada na modelagem dos *softwares* propostos nos objetivo s dessa pesquisa.

1.7.2 Processos de *Software*

Segundo Pressman (2011, p.52), um processo de *software* é "uma metodologia para as atividades, ações e tarefas necessárias para desenvolver um *software* de alta qualidade".

Para Sommerville (2003), não existe um processo ideal, pois isso dependerá de cada projeto, possibilitando cada qual implementar algum modelo já existente. Contudo Pressman (2011) afirma que uma metodologia genérica tem cinco passos:

- Comunicação - Antes de iniciar os trabalhos técnicos deve-se entender os objetivos do sistema e levantar requisitos para o bom funcionamento do *software*.
- Planejamento - Cria um plano de projeto, que conterá as tarefas a serem seguidas, riscos prováveis e recursos necessários.
- Modelagem - Esboça o sistema para que se tenha uma ideia de como ele deverá ficar e como encontrar a melhor solução para desenvolve-lo.
- Construção - É a etapa de desenvolvimento e teste.

- Emprego - O *software* pronto em sua totalidade ou parcialmente é implantado no cliente e este retorna o seu *feedback*.

Dessa forma, normalmente qualquer um dos modelos (ciclo de vida clássico, evolutivo ou espiral) utilizaram os princípios das metodologias acima citadas.

1.8 Google Cloud Messaging

Para que os alunos sejam notificados quando houver alguma mudança no portal do aluno será utilizada uma API oferecida pela *Google* denominada *Google Cloud Messaging* ou simplesmente GCM[9], um recurso que tem por objetivo notificar as aplicações Android. Segundo Leal (2014) ele permite que aplicações servidoras possam enviar pequenas mensagens de até 4 KB[10] para os aplicativos moveis, sem que esta necessite estar em execução, pois o sistema operacional envia mensagens em *broadcast*, dessa forma a aplicação pode tratar as mensagens como bem preferir.

De acordo com Leal (2014) para o bom funcionamento do recurso apresentado, são necessários os seguintes componentes:

- *Sender ID*[11] – É o identificador do projeto. Será utilizado pelo servidores da *Google* para identificar a aplicação que envia a mensagem.
- *Application ID* – É o identificador da aplicação Android. O identificador é o nome do pacote do projeto que consta no AndroidManifest.xml.
- *Registration ID* – É o identificador gerado pelo servidor GCM quando aplicação Android se conecta a ele. Deve ser enviado também a aplicação servidora.
- *Sender Auth Token* – É uma chave que é incluída no cabeçalho quando a mensagem é enviada da aplicação servidora para o GCM. Essa chave é para que a API da Google possa enviar as mensagens para o aplicativo correto.

De acordo com os componentes acima citados, quando uma aplicação servidora enviar uma mensagem para o aplicativo Android, na verdade está enviando para o servidor GCM que será encarregado de enviar a mensagem para a aplicação mobile.

[9] *Google Cloud Messaging*
[10] *Kilobytes*
[11] Identity

2 QUADRO METODOLÓGICO

Nesse capítulo serão apresentados os métodos adotados para se realizar esta pesquisa, tais como tipo de pesquisa, contexto, procedimentos, entre outros.

2.1 Tipo de pesquisa

Uma pesquisa é o ato de buscar e procurar pela resposta de algo. Marconi e Lakatos (2002, p. 15) definem pesquisa como "uma indagação minuciosa ou exame crítico e exaustivo na procura de fatos e princípios".

Existem diversos tipos de pesquisa, no entanto percebeu-se que para o propósito desta, a mais indicada foi a pesquisa aplicada, pois está se desenvolvendo um projeto real que poderá ser utilizado por qualquer instituição de ensino, mas que não mudará a forma com que as pessoas recebam suas informações, apenas acrescentará mais uma forma de consultá-las.

Segundo Marconi e Lakatos (2002, p. 15), uma pesquisa do tipo aplicada "caracteriza-se por seu interesse prático, isto é, que os resultados sejam aplicados ou utilizados, imediatamente, na solução de problemas que ocorrem na realidade".

Dessa maneira, percebeu-se que esta pesquisa enquadra-se no tipo de pesquisa aplicada, pois com a execução da mesma resolve um problema específico, e para isso está desenvolvendo-se um aplicativo para dispositivos móveis que facilitará aos graduandos acessarem o sistema *web* de uma universidade.

2.2 Contexto de pesquisa

Essa pesquisa será benéfica a qualquer instituição educacional que possua um portal acadêmico *online*, pois facilitará o acesso dos discentes às suas informações escolares.

Este trabalho visa desenvolver um aplicativo para dispositivos móveis, com plataforma *Android*, o qual notifica os usuários quando houver alguma atualização nos seus dados, como por exemplo ao ser lançada a nota de uma prova.

O aluno consegue acessar o aplicativo com o mesmo *login* do sistema *web*. O utilitário acessa o *webservice* que é responsável por buscar as informações no banco de dados e apresentá-las no dispositivo.

2.3 Instrumentos

Pode-se dizer que um questionário é uma forma de coletar informações através de algumas perguntas feitas a um público específico. Segundo Gunther (2003), o questionário pode ser definido como um conjunto de perguntas que mede a opinião e interesse do respondente.

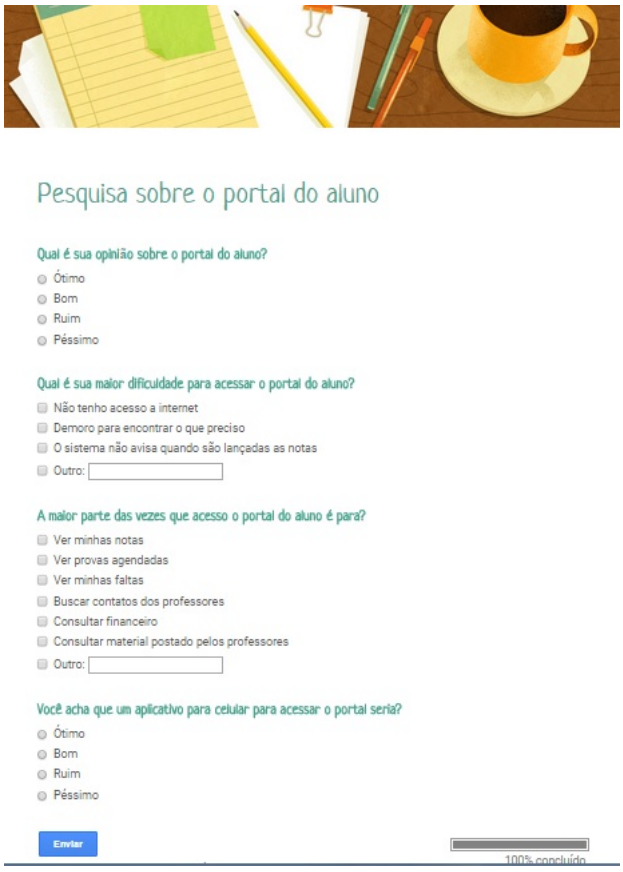


Figura 5 – Quetionário Aplicado. **Fonte:**Elaborado pelos autores.

Foi realizado um questionário simples, que está apresentado na Figura 5, contendo quatro perguntas e enviado para *e-mails* de alguns alunos da universidade. O foco desse questionário era saber o motivo pelo qual os usuários mais acessavam o portal do aluno e se tinham alguma dificuldade em encontrar o que procuravam. Obteve-se um total de treze respostas, no qual pode-se perceber que a maioria dos entrevistados afirmam terem dificuldades para encon-

trar o que precisam e que o sistema não avisa quando ocorre alguma alteração. Sobre o motivo do acesso cem por cento respondeu que entram no sistema web para consultar suas notas.

Outro instrumento utilizado para realizar esta pesquisa foram as reuniões, ou seja, reunir-se com uma ou mais pessoas em um local, físico ou remotamente para tratar algum assunto específico. Para Ferreira (1999), reunião é o ato de encontro entre algumas pessoas em um determinado local, com finalidade de tratar qualquer assunto.

Durante a pesquisa, foram realizadas reuniões entre os participantes com o objetivo de discutir o andamento das tarefas pela qual cada integrante ficou responsável. Além disso entravam em discussão, nessas reuniões, o cumprimento das metas propostas por cada participante e o estabelecimento de novas metas. Foram utilizadas nesta pesquisa, referências de livros, revistas, manuais e *web sites*.

2.4 Procedimentos

O primeiro procedimento realizado para chegar à pesquisa proposta, foi planejar o *software* através da linguagem UML, que necessitou da instalação da ferramenta *Astah* na sua versão 6.8.0.37.

Como o planejamento é um passo importante para o aplicativo Android, fez-se necessários os diagramas de classe, de caso de uso principal e de atividade.

Para a contrução do *software* proposto por esta pesquisa, além do levantamento de requisitos que é peculiar da construção de qualquer *software*, foram utilizados alguns diagramas que fazem parte do UML, tais como:

- Diagrama de casos de uso;
- Diagrama de sequência;
- Diagrama de Entidade e Relacionamento (ou Modelo de Entidade e Relacionamento)

Para iniciar o desenvolvimento do aplicativo, primeiramente fez-se necessária a instalação e configuração da plataforma *Android Studio* versão 1.1.0 e *Android SDK* versão 24.0.2. Ao concluir essa tarefa, deu-se início ao aplicativo *Android*.

O primeiro passo foi a criação de uma *activity main*, denominada `MainActivity`, a qual será executada quando a aplicação for iniciada. O tipo de *activity* escolhida é chama-se *Navigation Drawer Layout*. Na Figura 6, pode-se ver o método `onNavigationDrawerItemSelected()`,

que tem por finalidade mostrar as opções de navegação e o que deverá acontecer quando uma delas for clicada.

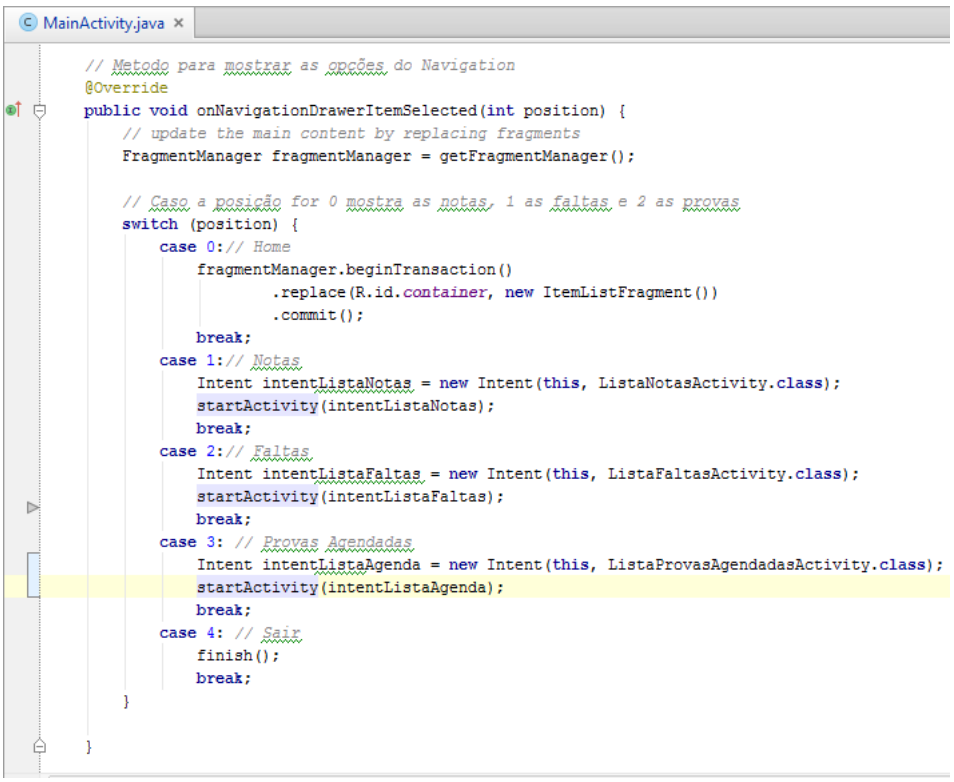


Figura 6 – Implementação do método `onNavigationDrawerItemSelected()`. **Fonte:**Elaborado pelos autores.

Ao executá-la, funcionará como um menu, com as opções *Home*, Notas, Faltas, Provas Agendadas e Sair. Essas opções ficam escondidas e só apareceram quando clicado no canto superior esquerdo como mostra a Figura 7.

Sempre que a aplicação é iniciada ou quando o usuário retorna para a *Home*, é apresentada uma lista com acesso a sites úteis. Para que essa lista aparecesse foi utilizada uma *activity* do tipo *Master/Deital Flow* que já traz consigo o *widget* de *ListView*. Essa *activity* mostrará as seguintes opções: Univás, MEC, Fies, Prouni e Google Acadêmico. Ao clicar em um desses itens, será chamada uma atividade que mostrará os detalhes do item escolhido. Nesse caso, na *activity* que detalhará o item, foi utilizada um *widget WebView* que mostrará a página *web* no espaço reservado. Dessa maneira, quando for clicado na opção Univás será carregada o site da universidade no aplicativo. Na Figura 8, é possível ver a tela *Home* do aplicativo.

Figura 7 – Representação Visual de `onNavigationDrawerItemSelected()`. **Fonte:**Elaborado pelos autores.

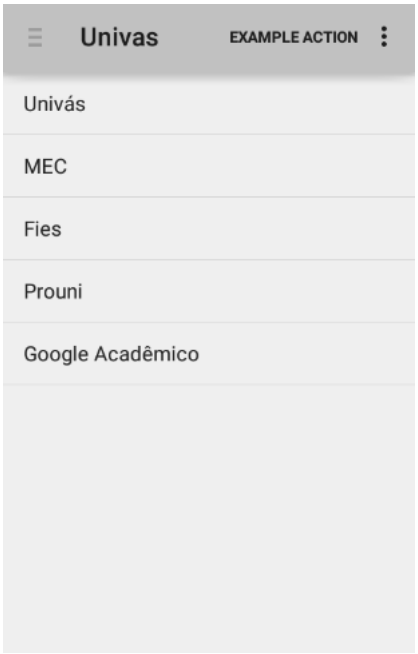


Figura 8 – *Activity Home* do aplicativo. **Fonte:**Elaborado pelos autores.

Logo após foram criadas três novas *activities* do tipo *Blank Activity*, as quais listarão as matérias cursadas pelo aluno. Nelas foram inseridas o *widget ExpandableListView*, para que quando clicado em um item listado, ele se expanda e mostre seus detalhes. Com isso, quando se escolhe o menu notas, será apresentada uma *activity* que listará todas as disciplinas cursadas e, ao clicar em uma delas, apresentaram as notas referente a ela. Na Figura 9, é possível ver uma *activity* que lista algumas matérias e ao clicar em uma delas foi possível ver as notas.

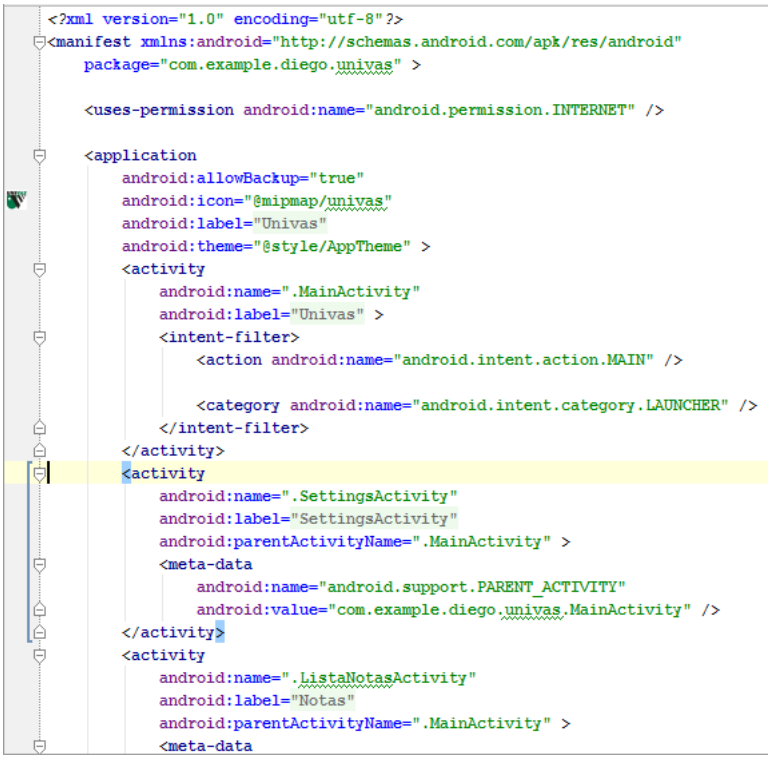
Foi criada uma classe chamada `BuscaDados`, que tem por finalidade receber os dados

Figura 9 – *Activity* Notas. **Fonte:**Elaborado pelos autores.

do *webservice*, salvá-los no banco de dados local do aplicativo e entregá-los para as classes que implementam o *Adapter* para listar as informações na tela do dispositivo.

No arquivo `AndroidManifest.xml` foi necessário alterar a opção `Android:icon`, que define qual será o ícone do aplicativo. Por padrão, ele apresenta o mascote do *Android*, no entanto foi definida uma imagem do logo da universidade. Foi necessário também incluir uma tag de `uses-permission`, que obriga o usuário a permitir o uso da *internet* pelo aplicativo, conforme pode ser visto na Figura 10. Pode-se perceber que cada *activity* encontra-se dentro das tags `<activity><activity>` e que a *activity main*, deve conter a tag `<intent-filter>` e dentro dela `<action android:name="android.intent.action.MAIN"/>`, indicando que ela será a primeira a executar e `<category android:name="android.intent.category.LAUNCHER"/` informando que ficará disponível junto aos outros aplicativos do dispositivo.

O *Android Studio* tem uma facilidade de se trabalhar com controladores de versão, nesse caso foi escolhido o *GitHub*. Nele foi criada uma pasta e compartilhada entre os participantes e, por fim, configurou-se o *Git* com a IDE para que cada um possa ter a versão mais atualizada do projeto.



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.diego.univas" >

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/univas"
        android:label="Univas"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="Univas" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".SettingsActivity"
            android:label="SettingsActivity"
            android:parentActivityName=".MainActivity" >
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.example.diego.univas.MainActivity" />
        </activity>
        <activity
            android:name=".ListaNotasActivity"
            android:label="Notas"
            android:parentActivityName=".MainActivity" >
            <meta-data
```

Figura 10 – `AndroidManifest.xml`. **Fonte:**Elaborado pelos autores.

No que diz respeito à contrução do *webservice*, foi necessária a instalação e configuração de um ambiente de desenvolvimento compatível com as necessidades apresentadas pelo *software* e que foram levantadas através dos requisitos. Foi instalado o *Servlet Container Apache Tomcat* em sua versão de número 7. O *Servlet Container* foi instalado para que o *Web Service* pudesse fornecer os serviços necessários para o consumo de dados do Aplicativo *Android*, haja vista que *Apache tomcat* faz uso amplo do protocolo HTTP[1] e da plataforma *Java* de desenvolvimento.

Para armazenar os dados gerados e/ou recebidos, foi necessário fazer a intalação do Sistema Gerenciador de Banco de Dados(SGBD) *PostGreSql* na sua versão de número 9.2. Através de um levantamento de requisitos parciais e das reuniões entre os participantes foi possível construir um Diagrama de Entidade e Relacionamento, no qual ficou definida a estrutura do banco de dados da aplicação. A Figura 11 mostra o Diagrama de Entidade e Relacionamento concebido para esta pesquisa.

Fazendo uso desse diagrama foi possível criar todas as classes *Java* que representam as entidades do mapeamento objeto-relacional. Essas classes foram criadas fazendo uso de anotações próprias do *Hibernate*, que é um *framework* que implementa a especificação JPA[2]. Es-

[1] HTTP - Hypertext Transfer Protocol
[2] JPA - *Java Persistense API*

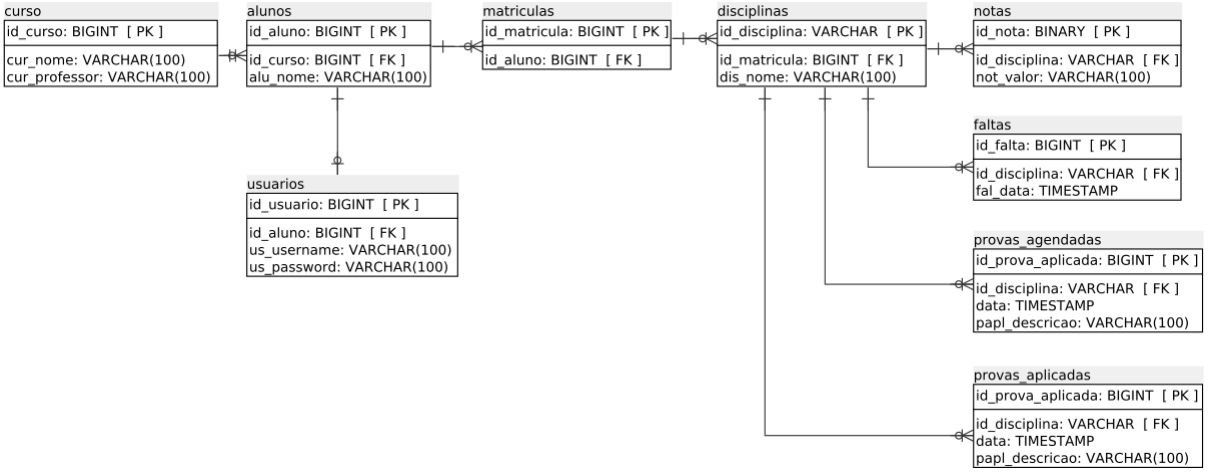


Figura 11 – Diagrama de Entidade e Relacionamento. **Fonte:**Elaborado pelos autores.

sas classes fazem parte dos mecanismos de persistêcia de dados e são simplesmente POJO's,[3] ou seja, objetos simples que contêm somente atributos privados e os métodos *getters* e *setters* que servem apenas para encapsular estes atributos. Uma das classes criadas, foi a classe `Curso.java` que representa a tabela `cursos` no banco de dados e está representada na Figura 2.1.

[3] POJO - *Plain Old Java Object*

Código 2.1 – Classe `Curso`. **Fonte:** Elaborado pelos autores.

```
@Entity(name = "cursos")
public class Curso {

    private Long idCurso;
    private String nome;
    private String professor;

    @Id
    @GeneratedValue
    @Column(name = "id_curso")
    public Long getIdCurso() {
      return idCurso;
    }

    public void setIdCurso(Long idCurso) {
      this.idCurso = idCurso;
    }

    @Column(length = 100, nullable = false)
    public String getNome() {
      return nome;
    }

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

    @Column(length = 100, nullable = false)
    public String getProfessor() {
      return professor;
    }

    public void setProfessor(String professor) {
      this.professor = professor;
    }

    /**
     *hashCode e Equals
     */
  }
```

Foram criadas outras classes *Java* com a mesma finalidade da anterior, porém com pequenas diferenças no que diz respeito à atributos, metodos e anotações. Estas classes representam, de maneira individual, as tabelas no banco de dados. Certos atributos dessas classes têm por finalidade representar as colunas de cada tabela. Já os atributos que armazenam instâncias de outras classes ou até mesmo conjuntos (coleções) de instâncias representam os relacionamentos entre as tabelas. E por fim, para cada classe que representa uma entidade, foi necessário implementar os métodos `hashCode` e `equals`, para que estas pudessem facilmente ser comparadas e diferenciadas em relação aos seus valores, haja visto que cada instância destas classes representa um registro no banco de dados.

Em seguida à criação das entidades, foi necessário configurar o arquivo `persistence.xml`

que fica dentro do *classpath* do projeto *Java* ou seja, dentro da mesma pasta onde estão contidos pacotes do projeto. Este arquivo é extremamente importante, pois é nele que estão todas as configurações relativas à conexão com o banco de dados, configurações referentes ao Dialeto SQL que vai ser usado para as consultas e configurações referentes ao *persistence unit* que é o conjunto de classes mapeadas para o banco de dados. O arquivo `persistence.xml` está exposto no código 2.2.

Código 2.2 – `persistence.xml`. **Fonte:** Elaborado pelos autores.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
  xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
  http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
    <persistence-unit name="WsUnivas">
      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <properties>
        <property name="javax.persistence.jdbc.url" value="jdbc:postgresql://localhost:5432/wsu
        <property name="javax.persistence.jdbc.user" value="postgres" />
        <property name="javax.persistence.jdbc.password" value="2289cpm22" />
        <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver" />
        <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect" />
        <property name="hibernate.show_sql" value="true" />
        <property name="hibernate.format_sql" value="true" />
        <property name="hibernate.hbm2ddl.auto" value="update" />
      </properties>
    </persistence-unit>
</persistence>
```

Em seguida à confecção do `persistence.xml` foi criada a classe `JpaUtil` que está representada na Figura 2.3. Esta classe é responsável por criar uma `EntityManagerFactory` que é uma fábrica de instâncias de `EntityManager` que nada mais é que um *persistence unit* ou unidade de persistência. Essa classe tem a responsabilidade de prover um modo de comunicação entre a aplicação e o banco de dados. No entanto a classe `JpaUtil` cria uma única instância de `EntityManagerFactory`, que é responsável por disponibilizar e gerenciar as instâncias de `EntityManager` de acordo com a necessidade da aplicação.

Código 2.3 – Classe `JpaUtil`. **Fonte:** Elaborado pelos autores.

```
1     public class JpaUtil {
2         private static EntityManagerFactory factory;
3
4         static {
5                 factory = Persistence.createEntityManagerFactory("
                      WsUnivas");
6         }
7
8         public static EntityManager getEntityManager() {
9                 return factory.createEntityManager();
10        }
11
12        public static void close() {
13                factory.close();
14        }
15
16 }
```

Em seguida à construção das classes que fazem a parte da persistência de dados, foi desenvolvido a parte de disponibilização de serviços *RESTful*, fazendo uso do *framework Jersey*. Com isso pode-se construir a classe que representa o primeiro serviço do *webservice*, que é a classe `Alunos`. Essa classe representa um contexto REST, e portanto, dispõe de alguns recursos. Esses recursos fazem a recuperação e a transmissão dos dados do *webservice* para o aplicativo *Android*. Essa classe e seus respectivos métodos estão representada na Figura **??**

```
1
2
3  @Path("/alunos")
4  public class Alunos {
5
6    @GET
7    @Path("/{$cod}")
8    @Consumes(MediaType.APPLICATION_JSON)
9    @Produces(MediaType.APPLICATION_JSON)
10   public String getAlunoById(Long idAluno) {
11     /**
12      * Implementacao
13      */
14     return aluno;
15
16   }
17
18   @GET
19   @Path("/")
20   @Consumes(MediaType.APPLICATION_JSON)
21   @Produces(MediaType.APPLICATION_JSON)
22   public String getAllAluno(Long idAluno) {
23     /**
24      * Implementacao
25      */
26     return alunos;
27
28   }
29 }
30 \caption[Classe \texttt{Alunos}]{Classe \texttt{Alunos}. \textbf{Fonte
       :}Elaborado pelos autores.}
```

```
31 \label{cod:serviceAlunos}
```

O *webservice* pode fazer a busca de alunos pelo `id` passado ou retornar uma coleção de alunos, dependendo do recurso acessado. Os tipos de dados que o *webservice* consome e retorna é o JSON[4]. Não foi necessário fazer nenhuma implementação adicional relativa a este formato, pois o próprio *framework Jersey* faz o tratamento e a conversão dos tipos de entrada e saída de dados. No caso do saída de dados, faz a conversão de objetos *Java* para JSON. E no caso de entrada tranforma um JSON em objeto *Java* já conhecido pelo *webservice*. Com isso concluiu-se o desenvolvimento do *webservice* que fornece os dados para o aplicativo.

Os procedimentos acima citados foram os passos até agora realizados com o propósito de se alcançar os resultados esperados para essa pesquisa.

[4] JSON - *Javascript Object Notation*

REFERÊNCIAS

ANDROID. : **A história do Android.** 2015. Disponível em: <https://www.android.com/history/>. Acesso em: 25 de Fevereiro de 2015.

ANDROID. : **Creating a Navigation Drawer**. 2015. Disponível em: <https://developer.android.com/training/implementing-navigation/nav-drawer.html>. Acesso em: 28 de julho de 2015.

ANDROID. : **Android Studio Overview.** 2015. Disponível em: <http://developer.android.com/tools/studio/index.html>. Acesso em: 12 de Março de 2015.

BEZERRA, E. : **Princípios De Análise E Projeto De Sistemas Com Uml**. 3ª. ed. São Paulo: Elsevier, 2015.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. : **UML:** guia do usuário. 2ª. ed. Rio De Janeiro: CAMPUS, 2012.

CAELUM. : **Java para Desenvolvimento Web.** 2015. Disponível em: <https://www.caelum.com.br/apostila-java-web/o-que-e-java-ee/#3-4-servlet-container>. Acesso em: 15 de Fevereiro de 2015.

CARVALHO, A.; CHIOSSI, T. : **Introdução à Engenharia de Software.** Campinas: Editora da Unicamp, 2001.

COULOURIS, G. et al. : **Sistemas Distribuídos** conceitos e projeto. 5ª. ed. Porto Alegre: Bookman Editora, 2013.

DEITEL, H.; DEITEL, P. : **Java como Programar** projeto orientado a objetos com a uml e padrões de projeto. 4ª. ed. Porto Alegre: Pearson Prentice Hall, 2003.

DEVMEDIA. : **Conheça o Apache Tomcat.** 2015. Disponível em: <http://www.devmedia.com.br/conheca-o-apache-tomcat/4546>. Acesso em: 08 de Março de 2015.

DURãES, R. : **Web Services para iniciantes**. 2005. Disponível em: <http://imasters.com.br/artigo/3561/web-services/web-services-para-iniciantes/>. Acesso em: 10 de Março de 2015.

ERL, T. : **Introdução às tecnologias Web Services:** soa, soap, wsdl e uddi. 2015. Disponível em: <http://www.devmedia.com.br/introducao-as-tecnologias-web-services-soa-soap-wsdl-e-uddi-parte1/2873>. Acesso em: 26 de Abril de 2015.

FERREIRA, A. B. H. : **Novo Aurélio Século XXI:** o dicionário da língua portuguesa. 3ª. ed. Rio de Janeiro: Nova Fronteira, 1999.

FIELDING, R. T. : **Architectural Styles and the Design of Network-based Software Architectures**. Tese (Doutorado) — University of California, 2000.

GODINHO, R. : **Criando serviços REST com WCF.** 2009. Disponível em: <https://msdn.microsoft.com/pt-br/library/dd941696.aspx>. Acesso em: 01 de Março de 2015.

GUEDES, G. T. A. : **UML 2 :** uma abordagem prática. 2ª. ed. São Paulo: Novatec, 2011.

GUNTHER, H. : **Como Elaborar um Questionário.** 2003. Disponível em: <http://www.dcoms.unisc.br/portal/upload/com_arquivo/como_elaborar_um_questionario.pdf>. Acesso em: 15 de Abril de 2015.

GUSMãO, G. : **Google lança versão 1.0 do IDE de código aberto Android Studio.** 2014. Disponível em: <http://info.abril.com.br/noticias/it-solutions/2014/12/google-lanca-versao-1-0-do-ide-de-codigo-aberto-android-studio.shtml>. Acesso em: 03 de Março de 2015.

HOHENSEE, B. : **Getting Started with Android Studio.** Gothenburg: [s.n.], 2013.

K19. : **Desenvolvimento mobile com Android**. 2012.

KRAZIT, T. : **Google's Rubin:** android 'a revolution'. 2009. Disponível em: <http://www.cnet.com/news/googles-rubin-android-a-revolution/>. Acesso em: 20 de Fevereiro de 2015.

LEAL, N. : **Dominando o Android:** do básico ao avançado. 1ª. ed. São Paulo: Novatec, 2014.

LECHETA, R. R. : **Google Android:** aprenda a criar aplicações para dispositivos móveis com android sdk. 2ª. ed. São Paulo: Novatec, 2010.

LECHETA, R. R. : **Google Android:** aprenda a criar aplicações para dispositivos móveis com o android sdk. 3ª. ed. São Paulo: Novatec, 2013.

MARCONI, M. A.; LAKATOS, E. M. : **Técnicas de pesquisas:** planejamento e execução de pesquisas, amostragens e técnicas de pesquisas, elaboração, análise e interpretação de dados. 5ª. ed. São Paulo: Atlas, 2002.

MILANI, A. : **PostgreSQL.** São Paulo: Novatec, 2008.

MONTEIRO, J. B. : **Google Android:** crie aplicações para celulares e tablets. São Paulo: Casa do Código, 2012.

PHILLIPS, B.; HARDY, B. : **Android Programming:** the big nerd ranch guide. Atlânta: Big Nerd Ranch, 2013.

PRESSMAN, R. : **Engenharia de software:** uma abordagem profissional. 7ª. ed. Porto Alegre: AMGH, 2011.

ROMANATO, A. : **Entenda como funciona a Java Virtual Machine (JVM).** 2015. Disponível em: <http://www.devmedia.com.br/entenda-como-funciona-a-java-virtual-machine-jvm/27624>. Acesso em: 08 de Março de 2015.

RUBBO, F. : **Construindo RESTful Web Services com JAX-RS 2.0.** 2015. Disponível em: <http://www.devmedia.com.br/construindo-restful-web-services-com-jax-rs-2-0/29468>. Acesso em: 03 de Março de 2015.

SAMPAIO, C. : **SOA e Web Services em Java**. 1ª. ed. Rio de Janeiro: Brasport, 2006.

SAUDATE, A. : **REST:** construa api's inteligentes de maneira simples. São Paulo: Casa do Código, 2012.

SAUDATE, A. : **SOA aplicado:** integrando com web serviços e além. 1ª. ed. São Paulo: Casa do Código, 2013.

SOMMERVILLE, I. : **Engenharia de Software.** 6ª. ed. São Paulo: Addison Wesley, 2003.

STONES, R.; MATTHEW, N. : **Beginning Databases with PostgreSQL:** from novice to professional. New York: Apress, 2005.

TOMCAT, A. : **The Tomcat Story.** 2015. Disponível em: <http://tomcat.apache.org/heritage.html>. Acesso em: 08 de Março de 2015.