DIEGO D'LEON NUNES
DIÓGENES APARECIDO REZENDE

APLICATIVO PARA CONSULTA DE NOTAS

UNIVERSIDADE DO VALE DO SAPUCAÍ
POUSO ALEGRE – MG
2015

DIEGO D'LEON NUNES
DIÓGENES APARECIDO REZENDE

APLICATIVO PARA CONSULTA DE NOTAS

Trabalho de Conclusão de Curso apresentado ao Curso de Sistemas de Informação da Universidade do Vale do Sapucaí como requisito parcial para obtenção do título de bacharel em Sistemas de Informação

Orientador: Prof. MSc. Roberto Rocha Ribeiro

UNIVERSIDADE DO VALE DO SAPUCAÍ
POUSO ALEGRE – MG
2015

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

ACID	Atomicidade, consistência, isolamento e Durabilidade
API	*Application Programming Interface*
GCM	*Google Cloud Messaging*
HTTP	*HyperText Transfer Protocol*
ID	*Identity*
IDE	*Integrated Development Environment*
I/O	*Input/Output*
JVM	*Java Virtual Machine*
JSON	*JavaScript Object Notation*
KB	*kilobyte*
REST	*Representational State Transfer*
SGBD	Sistema de Gerenciamento de Banco de Dados
SSL	*Secure Socket Layer*
SQL	*Structured Query Language*
UNIVAS	Universidade do Vale do Sapucaí
URL	*Universal Resource Locator*
XML	*Extensible Markup Language*

LISTA DE QUADROS

LISTA DE TABELAS

SUMÁRIO

INTRODUÇÃO

Atualmente, com os avanços tecnológicos, as pessoas estão cada vez mais conectadas e procuram soluções para seus problemas, que possam ajudá-las de forma rápida e fácil. Segundo Lecheta (2013), tanto as empresas quantos os desenvolvedores buscam plataformas modernas e ágeis para a criação de aplicações. Esse fato contruibuiu consideravelmente para o crescimento das plataformas móveis de comunicação.

Uma das áreas que mais se expandiu nos últimos anos é a de telefonia móvel. Monteiro (2012, p.1) afirma que "os telefones celulares foram evoluindo, ganhando cada vez mais recursos e se tornando um item quase indispensável na vida das pessoas". Essa evolução no *hardware* possibilitou o crescimento, mobilidade e portabilidade do *software*.

Muito das coisas que antes eram feitas somente em computadores *desktops* já podem ser realizadas nos celulares, como transferências bancarias, localização de taxi, conversas com amigos, entretenimento com jogos e vídeos, entre outros.

Ainda de acordo com Monteiro (2012), a plataforma *Android* se destaca no mercado devido ao grande número de aparelhos espalhados pelo mundo e pela facilidade que provêem aos desenvolvedores. Esta plataforma foi utilizada para o desenvolvimento de vários trabalhos de conclusão de curso como por exemplo Mendes (2011), que criou um aplicativo para que as bandas musicais pudessem ter mais interação com seus fãs. Oglio (2013) do Centro Universitário Univates, criou um sistema que permite o acesso ao portal virtual da sua faculdade.

Pelas facilidades que *smartphones* provêem para conseguir informações rápidas a qualquer hora e local, pensa-se em criar um utilitário que possibilite aos usuários consultarem as suas notas, presenças e provas agendadas no portal do aluno.

O obejetivo principal desta pesquisa é desenvolver um aplicativo, para dispositivos moveis, na plataforma *Android*, que permita aos alunos da Universidade do Vale do Sapucaí consultarem suas notas, presenças e provas agendadas. Porém o objetivo principal pode ser desmembrado em objetivos menores e mais concisos, com a finalidade de conseguir realizá-lo com maior eficácia. Estes, por sua vez são:

- Levantar requisitos do *software* proposto de acordo com as necessidades dos discentes.
- Desenvolver o aplicativo para dispositivos móveis na plataforma *Android*.
- Desenvolver um *web service* para prover os dados necessários ao aplicativo proposto.

Com esses passos espera-se fazer um *software* eficaz que auxiliará no dia-a-dia dos alunos.A escolha por fazer um aplicativo se deu pela necessidade em acessar o portal do aluno para ter informações referente às disciplinas. Com o projeto espera-se contribuir socialmente facilitando o acesso dos usuários às suas notas, provas agendadas e faltas.

O resultado final desta pesquisa também auxiliará os alunos do curso de Sistemas de Informação que necessitarem saber como se desenvolve um aplicativo na plataforma *Android* ou implementar mais funcionalidades nesse projeto.

A plataforma *Android* será utilizada devido a grande popularidade do sistema operacional. Visando facilitar as pesquisas aos conteúdos publicados no portal do alunos, quer-se desenvolver um App[1], pela qual os discentes os terão facilmente, pois serão notificados quando houver alguma mudança, como por exemplo ao ser lançada uma nota.

[1] Abreviação para a palavra *Application*

2 QUADRO TEÓRICO

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

2.1 *Java*

Conforme Deitel e Deitel (2010), *Java* é uma linguagem de programação orientada objetos, baseada na linguagem C++, desenvolvida pela empresa *Sun Microsystem* no ano de 1995, por uma equipe sob liderança de James Gosling.

Segundo Caelum (2015a), para executar as aplicações, o *Java* utiliza uma máquina virtual denominada JVM[1], que liberta os softwares de ficarem presos a um único sistema operacional, uma vez que o programa conversa diretamente com a JMV e fica por conta dela traduzir os *bytecodes* gerados pelo compilador para linguagem de máquina.

De acordo com Oracle (2015a), o *Java* está presente em mais de um bilhão de dispositivos como celulares, computadores, consoles de *games* e pode ser considerado, seguro, rápido e confiável.

Hoje, com a acenção do *Android*, a tendência é aumentar cada vez mais o desenvolvimento em *Java*, uma vez que os aplicativos Android são desenvolvidos nessa linguagem. Nessa pesquisa a linguagem de programação *Java* será utilizado no desenvolvimento tanto do aplicativo quanto do *web service*.

2.2 *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

[1] JVM - Java Virtual Machine

softwares, *hardwares* e telecomunicações, que tem por finalidade desenvolver uma plataforma para dispositivos móveis que seja completa, aberta e gratuita.

Krazit (2009) 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 esteja entre a *tag* `<manifest></manifest>`. Para ser possível acessar a *internet* pelo aplicativo é preciso declarar a permissão de acesso da seguinte forma: `<uses-permission android:name="android.permission.internet"/>`.

Lecheta (2010) diz que as *intents* são recursos tão importantes que podem ser consideradas como o coração do *Android* e que estão presentes em todas as aplicações. 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 *BroadCast 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* explícitas: 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. Na Figura 1 é demostrado um exemplo de ciclo de vida de uma *activity*.

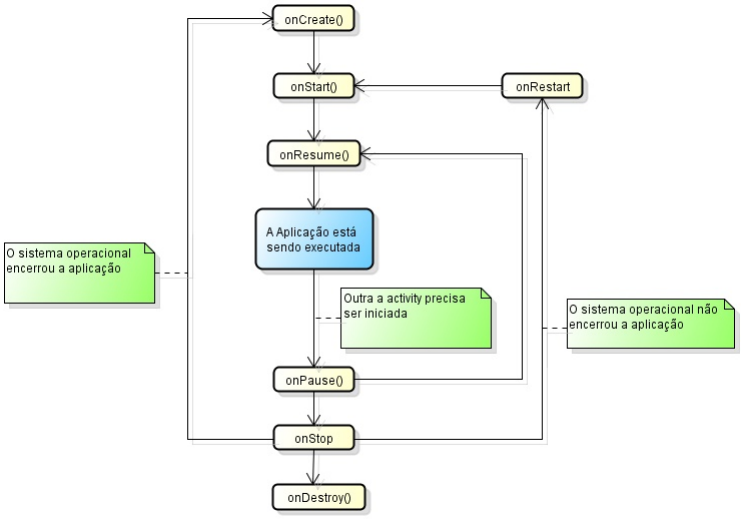

Figura 1 – Ciclo de Vida de uma *Activity*. **Fonte:**Lecheta (2010)

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 discente. 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 informações que 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 ao usuário. Ao encerrar a ligação, há dois caminhos possíveis de se percorrer, o primeiro, seria o caso do sistema operacional encerrar completamente a aplicação, por necessidade de liberar espaço em memória. Dessa forma será necessário chamar o método `onCreate()` novamente seguindo o ciclo normal, porém senão for encerrada completamente, ao findar a ligação será executado o método `onRestart()` e voltar para a *activity* ao qual o usuário se encontrava. Por fim quando o aluno sair da *activity* será 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 `<intent-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[2] 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, que fica normalmente escondida aparecendo apenas quando clicado no canto superior esquerdo.

Segundo Lecheta (2010), a classe *Service* 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 em memória é insuficiente.

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 possibilita às 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 eventos de uma *intent*.

Essa classe sempre é executada em segundo plano, 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 feita no *AndroidManifest.xml* pela *tag* `<receiver>` junto com a *tag* `<intente-filter>` e conterá a rotina a ser realizada quando for chamada.

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

[2] APP - abreviação para *aplication*

dispositivo estiver em forma vertical os itens ficaram um abaixo do outro e quando estiver na horizontal eles ficaram um ao lado do outro.

- *RelativeLayout*: permite posicionar elementos de forma relativa, ou seja um *widget* com relação a outro.
- *TableLayout*: permite criar *layouts* em formato de tabelas. O elemento *TableRow* representa uma linha da tabela e seus filhos são 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, similar ao *combobox*.
- *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.
- *Action Bar*: 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 levará um certo tempo para executar, como por exemplo, fazer um *download*, pode ser feito uma animação informando ao usuário o progresso da operação.
- *SQLite*: é um banco de dados embarcado na plataforma *Android*, que armazena tabelas, *views*, índices, *triggers* em apenas um arquivo em disco. Somente é possível acessa-lo pela aplicação a qual o criou e é deletado caso o aplicativo seja removido.

Além dos recursos acima citados, um outro *widget* que podemos destacar é o *ExpandableListView*, que para Android (2015c), exibe os itens em forma de uma lista da forma como o *ListView*, o que diferencia-o é que ele mostra uma lista de dois níveis de rolagem vertical, em vez de abrir uma outra tela.

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, que 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 *intent* 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 e deenvolvedores.

2.3 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 manualmente.

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

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

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

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 serão em teste, chamadas de *Beta*, *Dev* e *Canary*.

Devido a fácil usabilidade e por ser a IDE oficial para o desenvolvimento *Android*, escolheu-se esse ambiente para a construção do aplicativo.

2.4 *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 estes problemas e proporcionar alta transparência entre as várias plataformas, foram criados as tecnologias *web services*.

De acordo com Erl (2015, s.p):

> 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 o 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 a comunicação entre as diversas plataformas, e atende de uma melhor forma que as tecnologias 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.

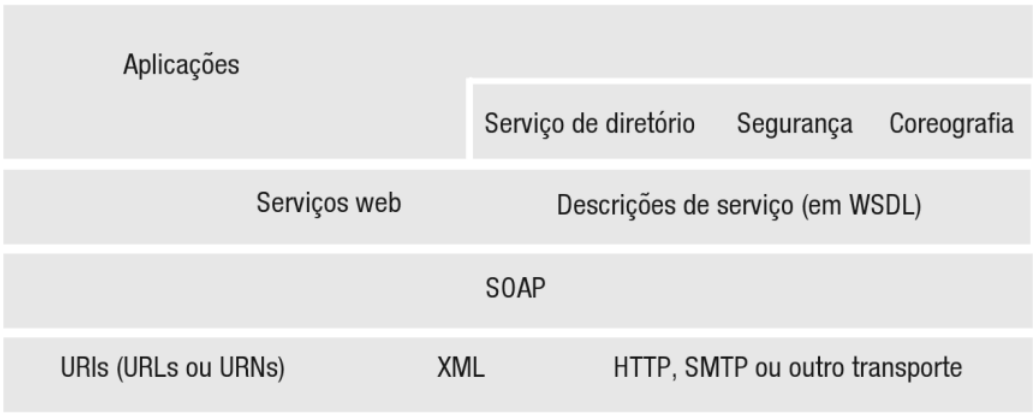

Figura 2 – Infraestrutura e componentes dos serviços *web*. **Fonte:**Coulouris et al. (2013)

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 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, que de acordo com Saudate (2013, p.19), "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 paramêtros referentes à mesma. É válido lembrar que todas requisições são trocadas usando SOAP, e usam o XML com formato oficial. Na Figura 3 está representado o esquema do Envelope SOAP.

Os *web services* além de fornecerem uma padronização de comunicação entre as várias tecnologias existentes, proveem transparência na troca de informações. Isso contribui pelo fato das novas aplicações poderem se comunicar com aplicações mais antigas ou aplicações contruídas sobre outras plataformas.

Além das tecnologias *web services* tradicionais, existe 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

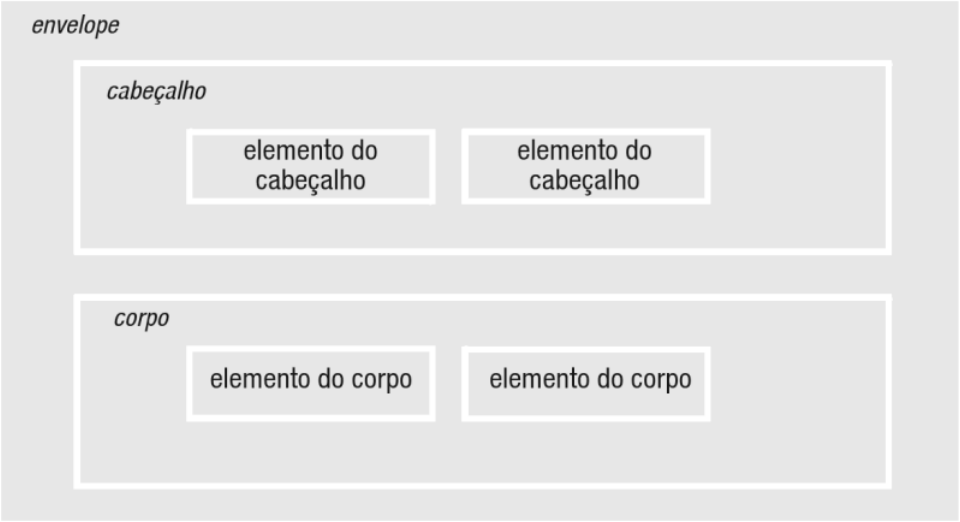

Figura 3 – Esquema de envelope SOAP. **Fonte:**Coulouris et al. (2013)

realmente importa. As caracteristícas do padrão REST serão abordadas na próxima seção.

2.4.1 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 dos 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 as 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 contruído na arquitetura REST basea-se fortemente em recursos. Para exemplificar o que seria um recurso em REST temos o seguinte cenário: considera-se uma URL tal como http://www.univas.edu.br/alunos/ onde pretende-se fazer a gerência dos dados de um aluno 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.

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

- 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 características que segundo Saudate (2013) foram responsáveis pelo sucesso da web. Essas características 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 a sua agilidade.

2.5 *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 (2015b) 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.

[5] XML - *Extensible Markup Language*.
[6] JSON - *JavaScript Object Notation*.
[7] EE - Sigla para enterprise edition

Conforme Devmedia (2015), para o desenvolvimento com *Tomcat* é necessária a utilização das seguintes técnologias:

- 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.

2.6 PostgreSQL

Para Milani (2008), todas as aplicações que armazenam informações para o seu uso posterior devem estar integradas a um banco de dados, seja armazenando em arquivos de textos ou em tabelas. Por isso, o *PostgreSql* tem por finalidade armazenar e administrar os dados em uma solução de informática.

Postgresql (2015a, s.p) define que "o *PostgreSql* é um SGBD (Sistema Gerenciador de Banco de Dados) objeto-relacional de código aberto, com mais de 15 anos de desenvolvimento. É extremamente robusto e confiável, além de ser extremamente flexível e rico em recursos."

Conforme afirma Milani (2008), o PostgreSql é um SGDB[8] de código aberto originado na Universidade de *Berkeley*, na Califórnia (EUA) no ano de 1986, pelo projeto *Postgres* desenvolvido por uma equipe sob liderança do professor Michael Stonebraker. Ele possui os principais recursos dos bancos de dados pagos e está disponível para os sistemas operacionais *Windows*, *Linux* e *Mac*. Atualmente existem bibliotecas e *drivers* para um grande número de linguagens de programação, entre as quais podemos citar: C/C++, PHP, *Java*, ASP, *Python* etc.

De acordo com Postgresql (2015b), existem sistemas com o *PostgreSql* que gerenciam até quatro *terabytes* de dados. Seu banco não possui um tamanho máximo e nem um número máximo de linhas por tabela. Contudo, uma tabela pode chegar a ter um tamanho de trinta e dois *terabytes* e cada campo a um *gigabyte* de informação.

Segundo Milani (2008), são características do *PostgreSql*:

- Suporte a ACID (Atomicidade, Consistência, Isolamento e Durabilidade).

[8] SGDB - Sistema Gerenciador de Banco de Dados

- Replicação de dados entre servidores.
- Cluster.
- Multithreads.
- Segurança SSL[9] e criptografia.

É através do *PostgreSql* que o *webservice* armazenará e posteriormente retornará os dados dos discentes para o aplicativo *Andorid*.

2.7 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.
- O paradigma espiral: engloba as melhores características do ciclo de vida clássica e o paradigma evolutivo. Ele consiste em vá - rios ciclos e cada ciclo representa uma fase do pesquisa.

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.

[9] SSL -*Secure Socket Layer*

2.7.1 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 a evolução das linguagens de programação orientadas a objetos, foram surgindo linguagens de modelagens orientadas a objetos, como um modo alternativo de análise e projeto de *software* usadas 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.

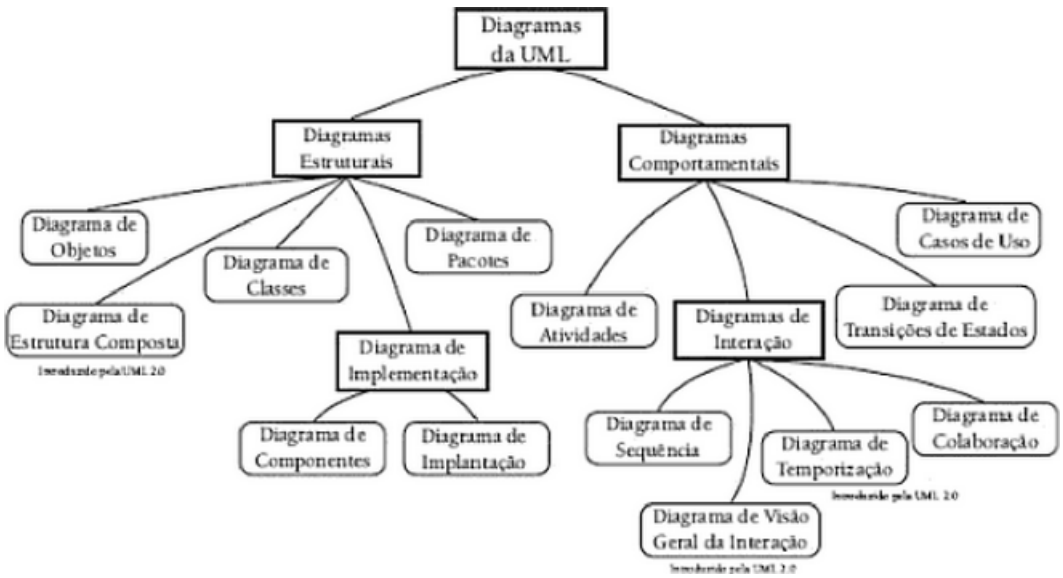

Figura 4 – Diagramas definidos pela UML. **Fonte:**Bezerra (2015)

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, será utilizada a linguagem de modelagem UML para o desenvolvimento desta pesquisa.

2.7.2 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 possui 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 desenvolvê-lo.
- Construção: é a etapa de desenvolvimento e testes.
- 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.

2.8 Google Cloud Messaging

Para que os graduandos sejam notificados quando houver alguma mudança no portal do aluno será utilizada uma API oferecida pela *Google* denominada *Google Cloud Messaging* ou simplesmente GCM[10], um recurso que tem por objetivo notificar as aplicações Android.

[10] *Google Cloud Messaging*

Segundo Leal (2014), ele permite que aplicações servidoras possam enviar pequenas mensagens de até 4 KB[11] para os aplicativos móveis, sem que este necessite estar em execução. Ainda de acordo com Leal (2014) para o bom funcionamento do recurso apresentado, são necessários os seguintes componentes:

- *Sender ID*[12]: 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. Este 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.

2.9 *Jersey*

Atualmente um padrão para desenvolvimento de serviços *web* vem sendo bastante adotado, trata-se do padrão arquitetural REST. De acordo com Saudate (2012) linguagem *Java* possui uma especificação própria para desenvolvimento de serviços REST desde de setembro de 2008, que é a JSR311, ou como é popularmente chamado JAX-RS. Esta especificação provê um conjunto de API's simples, para facilitar o desenvolvimento de serviços *web*. De acordo com Oracle (2015b) "JAX-RS é uma API da linguagem de programação *Java* projetado para tornar mais fácil para desenvolver aplicações que usam a arquitetura REST"[13]. Através desta especificação torna-se mais facíl e ágil a contrução de serviços *web* baseados em REST.

Como JAX-RS é apena uma especificação, ela necessita então de uma implementação. Uma das implementações desta especificação é o *framework Jersey*. Segundo Oracle (2015c)

[11] *KB - Kilobytes*
[12] Identity
[13] Tradução e resumo de informações de responsabilidade dos autores da pesquisa.

"*Jersey*, a implementação de referência de JAX-RS, implementa suporte para as anotações definidas no JSR 311, tornando mais fácil para os desenvolvedores a construir serviços *Web RESTful* usando a linguagem de programação *Java*"[14]. Além das anotações que facilitam seu uso, *Jersey* pode prover serviços com uma infinidade muito grande de tipos de mídias, tais como XML e JSON entre outros.

O *framework Jersey* tem amplo suporte para os varios métodos HTTP. Fazendo uso dele pode-se facilmente implementar recursos REST. Além disso *Jersey* pode rodar tanto em servidores que implementem a especificação *Servlet* ou não. Este *framework* será usado para contruir o que seria a parte responsável por prover os serviços para o aplicativo *Android*.

2.10 *Hibernate*

Com a evolução e popularização da linguagem *Java*, e com o seu uso cada vez maior em ambientes corporativos, percebeu-se que, perdia-se muito tempo com a confecção de queries SQL[15] usadas nas consultas em bancos de dados relacionais e com a construção do código JDBC[16] que era responsável por trabalhar com estas consultas. Além disso era notório que, mesmo a linguagem SQL sendo padronizada, ela apresentava diferenças significativas entre os diversos bancos de dados existentes. Isso fazia com que a implementação de um *software* ficasse amarrada em um banco de dados específico e era extremamente custosa uma mudança poterior. Além disso havia o problema de lidar diretamente com dois paradigmas um pouco diferentes: o orientado a objeto e o relacional. Com o intuito de resolver esses problemas é que surgiram os *frameworks* para ORM[17] tais como *Hibernate*, *EclipseLink*, *Apache OpenJPA* entre outros.

Conforme surgiam novas alternativas e implementações para sanar esses problemas, surgia um novo problema: a falta de padronização entre os *frameworks* de ORM. Para resolver esse problema foi criada o JPA[18] que de acordo com Keith e Schincariol (2009, p.12) "nasceu do reconhecimento das demandas dos profissionais e as existentes soluções proprietárias que eles estavam usando para resolver os seus problemas"[19].

A especificação JPA foi concebida sendo a terceira parte da especificação EJB[20], e deveria atender ao propósitos de persistêcia de dados desta especificação. De acordo com Keith e

[14] Tradução e resumo de informações de responsabilidade dos autores da pesquisa.
[15] *SQL - Structured Query Language*
[16] JDBC - *Java Database Connectivity*
[17] ORM - Object-relational Mapping
[18] JPA - *Java Persistence API*
[19] Tradução de responsabilidade dos autores da pesquisa.
[20] EJB - *Enterprise Java Bean*

Schincariol (2009, p.12) JPA é um *framework* leve baseado em POJO's,[21] para persistência de dados em *Java*, e que embora o mapeamento objeto relacional seja seu principal componente, ele ainda oferece soluções de arquitetura para aplicações corporativas escaláveis[22].

O *framework Hibernate* é uma das implementações da especificação JPA. De acordo com Sourceforge (2015) o *Hibernate* é uma ferramenta de mapeamento relacional, muito popular entre aplicações *Java* e implementa a *Java Persistence API*. Foi criado por uma comunidade de desenvolvedores, do mundo todo, que eram liderados por Gavin King. De acordo com Jboss (2015) "*Hibernate* cuida do mapeamento de classes *Java* para tabelas de banco de dados, e de tipos de dados *Java* para tipos de dados SQL".

O *Hibernate* está bastante difundido na comunidade de desenvolvedores *Java* ao redor do mundo, pelo fato de ser simples de usar, e por evitar esforços desnecessários na parte de infraestrutura das aplicações onde é usado, mantendo assim o foco na lógica de negócio. A pricipais vantagens do uso do *Hibernate* segundo Sourceforge (2015) são:

- Provedor JPA: além de sua API nativa , o *hibernate* também é uma implentação da especificação JPA, podendo assim ser facilmente usado em qualquer ambiente de apoio JPA.
- Persistência idiomática: permite que sejam construídas classes persistentes orientadas a objetos, e que suportem herança e polimorfismo entre outras estratégias, sem a necessidade da contrução de estruturas especiais para tal fim.
- Performance e suporte: permite que sejam usadas várias estratégias de de inicialização. Além disso não necessita de tabelas especiais no banco de dados. Mostra-se vantajoso também por gerar a maior parte do SQL necessário e evitar esforço desnecessário por parte do desenvolvedor, além de ser mais rápido que o JDBC puro.
- Escalável: o *Hibernate* foi projetado para trabalhar em *clusters* de servidores de aplicações e oferecer uma estrutura muito escalável, que se comporta bem tanto com um número muito baixo de usuários até números muitos elevados de usuários.
- Confiável: sua confiabilidade e estabilidade são comprovadas pelo seu grande uso e aceitação atualmente.
- Extensível: Hibernate é altamente configurável e extensível[23].

[21] POJO - *Plain Old Java Object*
[22] Tradução e resumo de informações de responsabilidade dos autores da pesquisa.
[23] Tradução e resumo de informações de responsabilidade dos autores da pesquisa.

O *Hibernate* será usado nesta pesquisa com o intuito de fazer a gerência dos dados coletados e que serão providos para o aplicativo *Android* através do *webservice*, em conjunto com o banco de dados.

3 QUADRO METODOLÓGICO

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

3.1 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 opção 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.

3.2 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.

3.3 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.

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 encontrar 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.

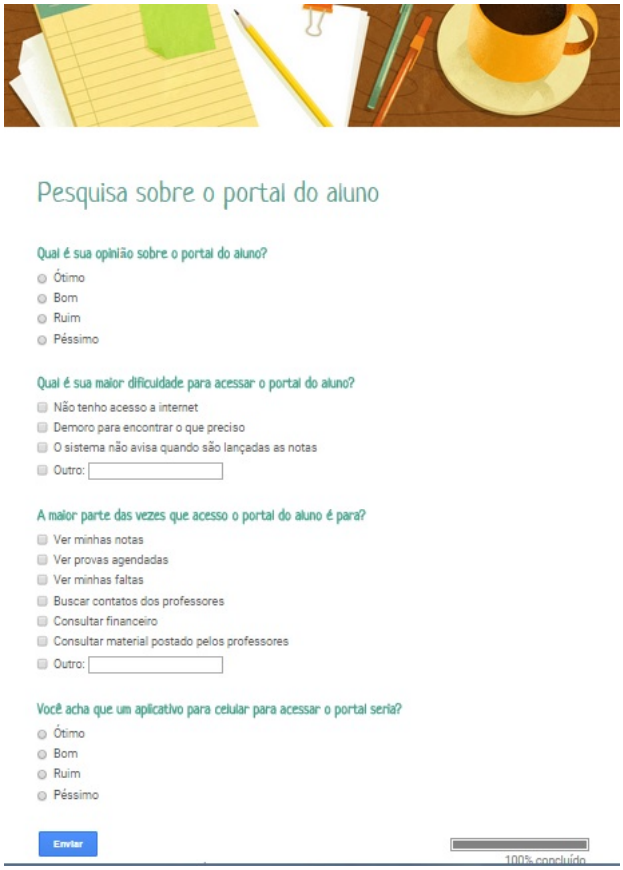

Figura 5 – Quetionário Aplicado. **Fonte:**Elaborado pelos autores.

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*.

3.4 Procedimentos e Resultados

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

Sabendo-se que o planejamento é um passo importante, além do levantamento de requisitos foram utilizados os diagramas de UML para nortear o desenvolvimento. Para se ter uma visão das funcionalidades do sistema por parte do usuário foi criado o diagrama de casos de uso do aplicativo, como demostra na Figura 6

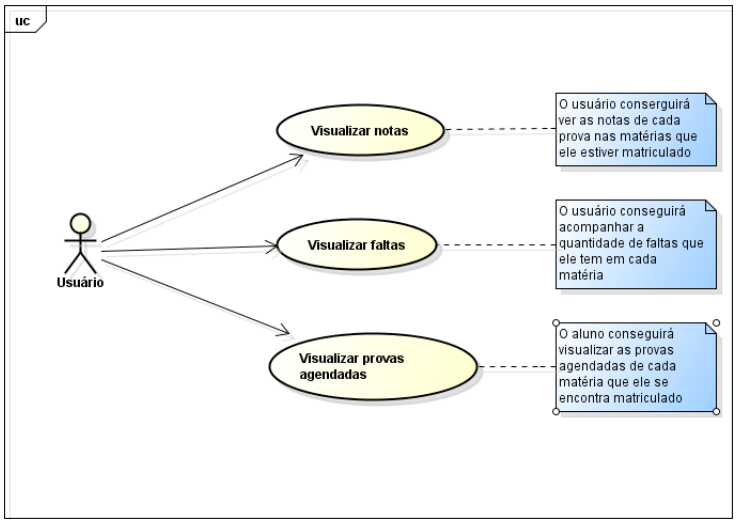

Figura 6 – Diagrama de Casos de Uso. **Fonte:**Elaborado pelos autores.

Logo após, foi projetado o diagrama de atividades para se ter uma visão da sequência do fluxo do sistema, conforme mostra a Figura 7.

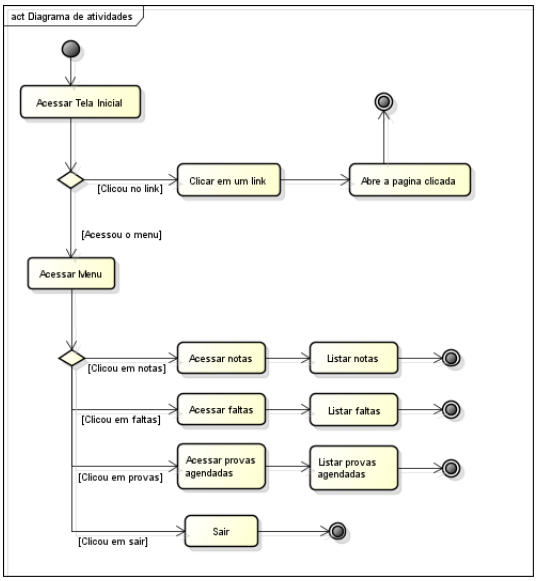

Figura 7 – Diagrama de atividades. **Fonte:**Elaborado pelos autores.

A próxima etapa realizada foi o desenvolvimento do diagrama de classes do aplicativo *Android*, o qual tem por objetivo mostrar todas as classes que o sistema necessita, como pode-se observar na Figura 8.

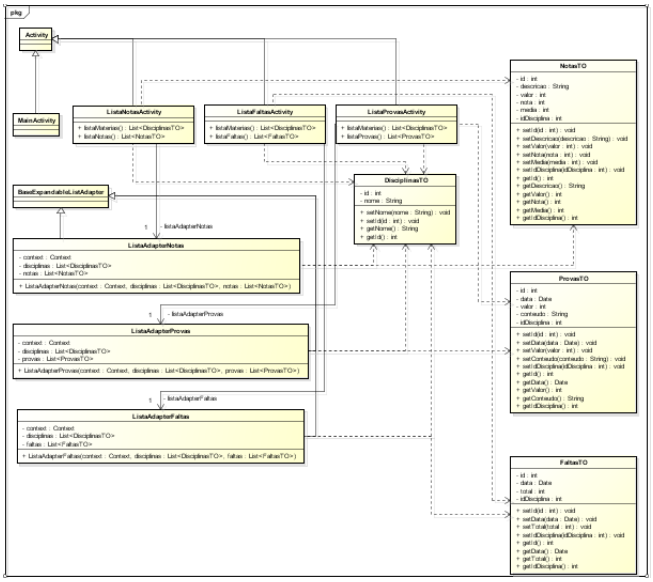

Figura 8 – Diagrama de classes do aplicativo *Android*. **Fonte:**Elaborado pelos autores.

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.

Logo após a configuração dos ambientes de desenvolvimento, 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 do tipo *Navigation Drawer Layout*. Na Figura 9, 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. Um exemplo, caso o usuário escolha o item Notas, o software executará o *case* um, o qual criará uma *intent* e chamará a classe `ListaNotasActivity`.

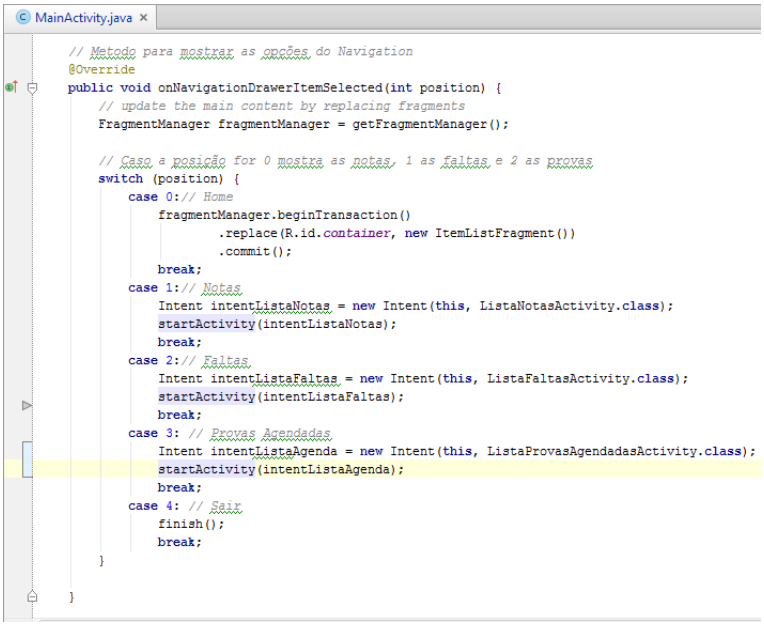

Figura 9 – *MainActivity*. **Fonte:**Elaborado pelos autores.

O próximo passo, foi criar a *activity Home*, o qual trará uma lista de *sites* uteis, com as opções Univás, MEC, Fies, Prouni e *Google* acadêmico. Para que essa lista aparecesse foi utilizada uma *activity* do tipo *Master/Deital Flow* que já traz consigo o *widget* de *ListView*. Na Figura 10, pode-se ver a classe `DummyContent`, local onde é declarado a lista de sites.

```
public static Map<String, DummyItem> ITEM_MAP = new HashMap<>();

static {
    // Add 3 sample items.
    addItem(new DummyItem("1", "Univás", "http://www.univas.edu.br/"));
    addItem(new DummyItem("2", "MEC", "http://portal.mec.gov.br/"));
    addItem(new DummyItem("3", "Fies", "http://sisfiesportal.mec.gov.br/"));
    addItem(new DummyItem("4", "Prouni", "http://prouniportal.mec.gov.br/"));
    addItem(new DummyItem("5", "Google Acadêmico", "https://scholar.google.com.br/"));
}

private static void addItem(DummyItem item) {
    ITEMS.add(item);
    ITEM_MAP.put(item.id, item);
}

/**
 * A dummy item representing a piece of content.
 */
// Altear para poder chamar os links
public static class DummyItem {
    public String id;
    public String website_name;
    public String website_url;

    public DummyItem(String id, String website_name, String website_url) {
        this.id = id;
        this.website_name = website_name;
        this.website_url = website_url;
    }
```

Figura 10 – Classe *DummyContent*. **Fonte:**Elaborado pelos autores.

Quando clicado em algum item, é executada a classe `ItemDetailActivity`, que chamará o `ItemDetailFragment`, responsável por carregar o *site* em questão no *layout* do aplica-

tivo. Abaixo, na Figura 11, vê-se o método responsável por passar para o *layout* o endereço do *site*.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    View rootView = inflater.inflate(R.layout.fragment_item_detail, container, false);

    // Show the dummy content as text in a TextView.
    if (mItem != null) {
        ((WebView) rootView.findViewById(R.id.item_detail)).loadUrl(mItem.website_url);
        //Uri uri = Uri.parse("http://portal.mec.gov.br/");
        //Intent intent = new Intent(Intent.ACTION_VIEW, uri);
        //startActivity(intent);
    }

    return rootView;
}
```

Figura 11 – Método para carregar o *layout* com *web view*. **Fonte:**Elaborado pelos autores.

Para que as informações possam aparecer, foram criadas *activities* do tipo *Blank Activity*. No *layout* dessas *activities* foram inseridas o *widget ExpandableListView*, como é mostrado na Figura 12.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context="com.example.diego.univas.ListaNotasActivity">

    <ExpandableListView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/expandableListView2"
        android:layout_alignParentBottom="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true" />
</RelativeLayout>
```

Figura 12 – *Layout* com *expandable*. **Fonte:**Elaborado pelos autores.

Foi criada uma classe chamada `BuscaDados`, que tem por finalidade receber os dados 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 13. 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.

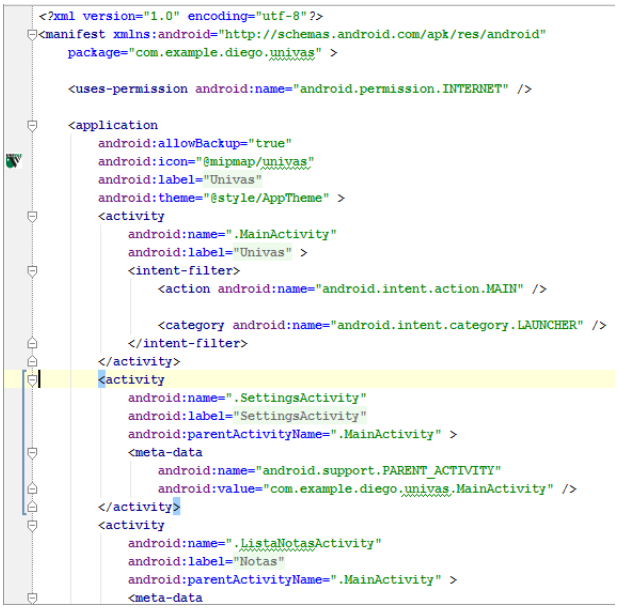

Figura 13 – `AndroidManifest.xml`. **Fonte:**Elaborado pelos autores.

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.

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 14 mostra o Diagrama de Entidade e Relacionamento concebido para esta pesquisa.

[1] HTTP - Hypertext Transfer Protocol

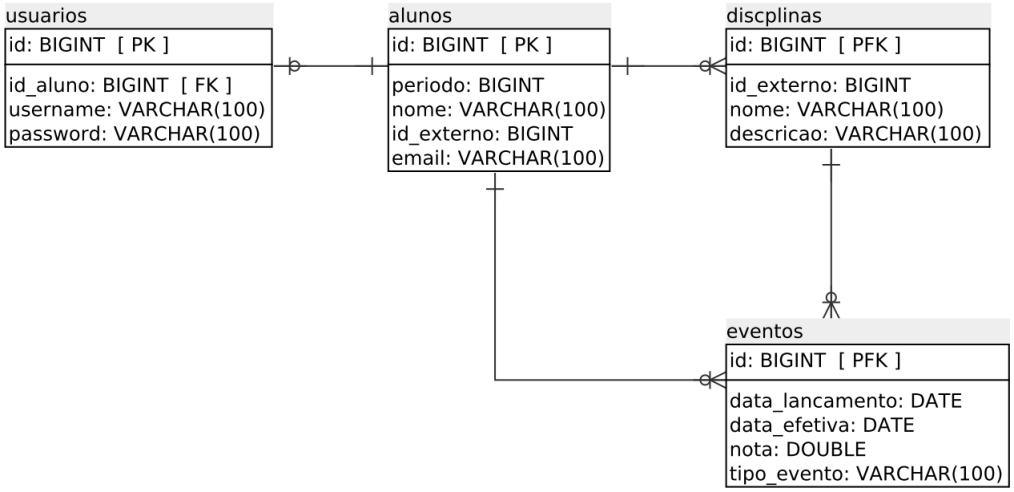

Figura 14 – Diagrama de Entidade e Relacionamento. **Fonte:**Elaborado pelos autores.

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]. Essas classes fazem parte dos mecanismos de persistêcia de dados e são simplesmente t 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 `Aluno.java` que representa a tabela `alunos` no banco de dados e está representada na Figura 15.

[2] JPA - *Java Persistense API*

```java
@Entity
@Table(name = "alunos")
public class Aluno {

    @Id
    @SequenceGenerator(
            name = "id_aluno",
            sequenceName = "seq_id_aluno",
            allocationSize = 1
            )
    @GeneratedValue(
            generator = "id_aluno",
            strategy = GenerationType.IDENTITY)
    @Column(name = "id_aluno", nullable = false)
    private Long idAluno;

    @Column(name = "id_externo", nullable = false)
    private Long idDbExterno;

    @Column(length = 100, nullable = false)
    private String nome;

    @Column(nullable = false)
    private Integer periodo;

    @Column(length = 100, nullable = false)
    private String email;

    @OneToMany(mappedBy = "aluno")
    private List<Evento> eventos;

    @OneToMany(mappedBy = "aluno")
    private List<Disciplina> disciplinas;

    /**
     *
     * GETTERS E SETTERS
     *
     */

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result +
                ((idAluno == null) ? 0 : idAluno.hashCode())
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Aluno other = (Aluno) obj;
        if (idAluno == null) {
            if (other.idAluno != null)
                return false;
        } else if (!idAluno.equals(other.idAluno))
            return false;
        return true;
    }

}
```

Figura 15 – Classe `Aluno`. **Fonte:**Elaborado pelos autores.

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 relaciona-

mentos 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 16.

```
<?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="WsAppUnivas" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
        <properties>
            <property name="javax.persistence.jdbc.url" value="jdbc:postgresql://localhost:5432/wsappunivas" />
            <property name="javax.persistence.jdbc.user" value="postgres" />
            <property name="javax.persistence.jdbc.password" value="password" />

            <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.temp.use_jdbc_metadata_defaults" value="false"></property>
            <property name="hibernate.hbm2ddl.auto" value="create" />
        </properties>
    </persistence-unit>
</persistence>
```

Figura 16 – Arquivo `persistence.xml`. **Fonte:**Elaborado pelos autores.

Em seguida à confecção do `persistence.xml` foi criada a classe `JpaUtil` que está representada na Figura 17. 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.

```java
public class JpaUtil {
    private static EntityManagerFactory factory;

    static {
        factory = Persistence.createEntityManagerFactory("WsAppUnivas");
    }

    public static EntityManager getEntityManager() {
        return factory.createEntityManager();
    }

    public static void close() {
        factory.close();
    }

}
```

Figura 17 – Classe `JpaUtil`. **Fonte:**Elaborado pelos autores.

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 18.

```java
@Path("/alunos")
public class AlunosService {

    /*
     * Busca um Aluno e a suas informações e eventos
     */
    @GET
    @Path(" /{ $cod } ")
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Alunos getAlunoById(@PathParam("cod") Long idAluno) {
        Alunos alunos = new Alunos();
        AlunoCtrl ctrl = new AlunoCtrl();
        alunos.setAlunos(ctrl.getById(idAluno));

        return alunos;
    }
    /*
     * Busca os eventos de um Aluno pelo seu id
     */
    @GET
    @Path("/eventos/{ $cod }")
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Eventos getAll(@PathParam("cod") Long idAluno) {
        Eventos eventos = new Eventos();
        EventoCtrl ctrl = new EventoCtrl();
        eventos.setEventos(ctrl.getById(idAluno));
        return eventos;
    }
}
```

Figura 18 – Classe `AlunosService`. **Fonte:**Elaborado pelos autores.

O *webservice* pode fazer a busca de alunos pelo `id` passado ou retornar uma coleção de eventos vinculados a um alunos, dependendo do recurso acessado. Os tipos de dados que o *webservice* consome e retorna é o JSON[3]. Não foi necessário fazer nenhuma implementação

[3] JSON - *Javascript Object Notation*

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.

Para que fosse possível transmitir dados para o aplicativo, era necessário receber as informações do sistema acadêmico da referida instituição, haja vista que o *web service* é independente do mesmo. Para esse propósito é necessário contruir um módulo que faça a importação dos dados necessários para a base de dados do *web service*. Este por sua vez terá a responsabilidade de fazer a importação dos dados periodicamente, e ainda tratar os tipos de dados recebidos para tipos aplicáveis ao banco de dados local. Além disso é preciso notificar o módulo responsável por invocar o serviço *Google Cloud Messaging* para que os dispositivos dos alunos aos quais houveram atualizações nos dados, fossem notificados e fizessem acesso ao *web service* para solicitar esses dados atualizados.

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

4 DISCUSSÃO DE RESULTADOS

O sistema operacional *Android* mostrou o porquê de ser tão utilizado nos dias atuais. Com uma gama enorme de recursos totalmente gratuitos e com a documentação excelente torna-se claro o que cada função realiza, com decorrer do desenvolvimento do aplicativo.

Como se constatou que os discentes, na maioria das vezes, acessam o portal do aluno para consultar notas, faltas e provas agendadas, o aplicativo tem como importância facilitar para que os graduandos tenham suas informações de maneira simples e rápida. É notório que há mais facilidade em acessar esses dados pelos *smartphones* do que em *desktops*, pois, quando um professor lançar uma determinada nota, o aluno será notificado de que alguma informação nova está no portal, evitando que o usuário fique entrando no portal várias vezes ao dia ansioso em saber sua média final.

O aplicativo é de fácil utilização, pois tem como tela principal, uma *activity* do tipo *Navigation Drawer Layout*, que de acordo com o Android (2015b), a *activity* fica escondida e aparece somente quando chamada pelo usuário exibindo do lado esquerdo do dispositivo todas as opções de navegação do *software*, facilitando para o aluno se localizar, além de ficar com uma aparência mais agradável. A seguir pode-se ver na Figura 19, o layout de menus que aparecerá quando clicado na opção selecionada.

Figura 19 – Menu do Aplicativo. **Fonte:**Elaborado pelos autores.

Na *home* do aplicativo foi utilizado uma lista com *links* de *sites* uteis aos alunos, como é visível na Figura 20. Anteriormente havia se pensado em abrir os sites através de uma *intent* implícita que como ressalta Monteiro (2012) é criada para avisar o sistema operacional *Android*, que é necessário abrir uma outra *activity*, a qual seria responsável por executar uma determinada ação, no caso, abrir uma página *web*. Porém, decidiu-se usar o *widget webView* que ainda segundo Monteiro (2012) permitirá carregar o *site* no próprio aplicativo. Portanto, ao clicar em alguma dessas opções será aberta a *activity* que detalhará o item, evitando abrir o navegador nativo, conforme é mostrado na Figura 21.

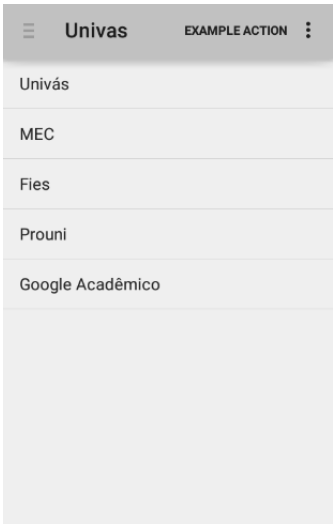

Figura 20 – *Home* do Aplicativo. **Fonte:**Elaborado pelos autores.

Figura 21 – *Site* carregado no `webView`. **Fonte:**Elaborado pelos autores.

As informações dos alunos vem do *web service* e são salvas no banco de dados *SQLite*, que conforme Monteiro (2012) é um banco de dados que já vem na plataforma *Android*, o qual não necessita a instalação ou configuração de ferramentas externas. Logo após as notas, faltas e provas agendadas são apresentadas aos usuários através de uma lista do tipo `ExpadableListView`, este, segundo Android (2015c) traz a vantagem que quando clicado em um de seus itens, é apresentado seus subitens na mesma tela, evitando abrir uma nova *activity* e com isso simplificando e melhorando o desempenho do software. Abaixo, na Figura 22 é possível ver a activity de notas, listando algumas informações com *widget* `ExpadableListView`.

Figura 22 – Tela de Apresentação de notas, faltas e provas agendadas. **Fonte:**Elaborado pelos autores.

O aplicativo resultado desta pesquisa, tinha necessidade de consumir dados para posteriormente apresentá-los ao usuário. Era necessário que, os dados do sistema acadêmico da instituição de ensino que serviu como contexto para esta pesquisa, fossem transmitidos de alguma forma ao aplicativo. Era necessário também que os dados chegassem ao aplicativo respeitando as particularidades de cada usuário, trazendo somente informações relevantes aos mesmos. Com esse intuito de disponibilizar informações já citadas anteriormente, a quem quer que fosse necessário, inclusive aos usuários do aplicativo, foi criado um *Web Service* REST. Este foi um dos resultados alcançados através desta pesquisa.

A construção do *web service*, de ínicio, mostrava-se um tanto quanto custosa, devido a restrições das tecnologias que foram escolhidas. Por se tratar de uma simples *web service* que seria disponibilizado para suplir a demanda de dados do aplicativo, os primeiros serviços foram

construídos e disponibilizados fazendo uso de *servlets* simples e conexão JDBC[1]. Este modo como foi pensado inicialmente, era simples de ser contruído e de uma *performance* aceitável.

Porém de acordo com o crescimento da demanda do serviço, tornou-se inviável a contrução do mesmo com estas técnologias, devido a complexidade com que era necessário contruir os serviços, haja vista que, com estas técnologias era necessário que se fosse configurado praticamente tudo de forma manual inclusive tratamento de erros da aplicação, respostas as requisições e tipos de dados. Esta etapa teve, portanto, um resultado não muito amigável do ponto de vista de sua construção. No entanto se for analizado do ponto de vista do conhecimento adquirido, obteve-se um resulatdo satisfatório, pois, foi na pesquisa e na busca melhores alternativas a estas técnologias, que se chegou ao resultado final.

O *web service* foi desenvolvido com algumas técnolgias que facilitaram a sua construção. Foi usado o *framework Jersey* para prover os serviços necessários. Este *framework* usa *servlets* para disponibilizar os serviços, porém com a facilidade de já ter embutido em si o tratamento para qualquer um dos tipos de midias que foram usadas na implementação de um serviço REST, tanto para entrada ou saida de dados. Foi ainda usado o *framework* de pesistência de dados *Hibernate*. Este por sua vez desenpenhou papel notório, pois, ao mesmo tempo que facilitou o desenvolvimento do *web service* com relação ao foco nas regras de negócio da aplicação e não tanto na implementação, pode-se dizer que se, por algum motivo for necessário migrar de banco de dados este será um fator facilitador. O resultado final foi extremamente satisfatório, pois era notório que era facíl tanto contruir e disponibilizar um novo serviço, quanto consumir o mesmo.

Portanto, conclui-se que se for analizado estes resultados citados de forma conjunta, pode-se dizer que o resultado geral foi satisfatório, pois, os discentes puderam obter as informações de que mais fazem uso de forma prática e rápida.

[1] JDBC - *Java Database Connectivity*

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. : **ExpandableListView.** 2015. Disponível em: <http://developer.android.com/reference/android/widget/ExpandableListView.html>. Acesso em: 24 Agosto 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. : **Apostila Java e Orientação a Objetos.** 2015. Disponível em: <http://www.caelum.com.br/apostila-java-orientacao-objetos/o-que-e-java/#2-3-maquina-virtual>. Acesso em: 18 de Setembro de 2015.

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**. São Paulo: Pearson Prentice Hall, 2010.

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.

JBOSS. : **Hibernate Getting Started Guide**. 2015. Disponível em: <http://docs.jboss.org/hibernate/orm/5.0/quickstart/html/>. Acesso em: 20 de Setembro de 2015.

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

KEITH, M.; SCHINCARIOL, M. : **Pro JPA 2:** *Mastering the Java Persistence API*. New York: Apress, 2009.

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.

MENDES, E. V. : **Um aplicativo para Android visando proporcionar maior interação de uma banda musical e seus seguidores.** Pato Branco: Universidade Tecnológica Federal do Paraná, 2011.

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

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

OGLIO, M. D. : **Aplicativo Android para o ambiente UNIVATES Virtual.** Lajeado: Univates, 2013.

ORACLE. : **O que é a Tecnologia Java e porque preciso dela?** 2015. Disponível em: <https://www.java.com/pt_BR/download/faq/whatis_java.xml>. Acesso em: 17 de Setembro de 2015.

ORACLE. : ***the java ee 6 tutorial*:***Creating a RESTful Root Resource Class*. 2015. Disponível em: <http://docs.oracle.com/javaee/6/tutorial/doc/giepu.html>. Acesso em: 20 de Setembro de 2015.

ORACLE. : ***the java ee 6 tutorial*:***Building RESTful Web Services with JAX-RS*. 2015. Disponível em: <http://docs.oracle.com/javaee/6/tutorial/doc/giepu.html>. Acesso em: 20 de Setembro de 2015.

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

POSTGRESQL. : **O que é PostgreSQL?** 2015. Disponível em: <https://wiki.postgresql.org/wiki/Introdu%C3%A7%C3%A3o_e_Hist%C3%B3rico>. Acesso em: 11 de de 2015.

POSTGRESQL. : **Sobre o PostgreSQL.** 2015. Disponível em: <http://www.postgresql.org.br/old/sobre>. Acesso em: 11 de de 2015.

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

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.

SOURCEFORGE. : **Hibernate**. 2015. Disponível em: <http://sourceforge.net/projects/hibernate/>. Acesso em: 20 de Setembro de 2015.

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