

**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 Universi-  
dade do Vale do Sapucaí como requisito parcial  
para obtenção do título de bacharel em Sistemas  
de Informação

Orientador: Prof. MSc. Roberto Ribeiro Rocha

**UNIVERSIDADE DO VALE DO SAPUCAÍ  
POUSO ALEGRE – MG**

**2015**

## LISTA DE FIGURAS

Figura 1 – Participação de mercado do <i>Android</i> em 2016 . . . . .	7
Figura 2 – Ciclo de Vida de uma <i>Activity</i> . . . . .	11
Figura 3 – Infraestrutura e componentes dos serviços <i>web</i> . . . . .	16
Figura 4 – Quetionário Aplicado . . . . .	25
Figura 5 – Sender ID do GCM . . . . .	27
Figura 6 – Geração da credencial do GCM . . . . .	27
Figura 7 – sem legenda . . . . .	29
Figura 8 – sem legenda . . . . .	30
Figura 9 – sem legenda . . . . .	31
Figura 10 – sem legenda . . . . .	32
Figura 11 – sem legenda . . . . .	33
Figura 12 – sem legenda . . . . .	34
Figura 13 – sem legenda . . . . .	34
Figura 14 – sem legenda . . . . .	35
Figura 15 – sem legenda . . . . .	36
Figura 16 – sem legenda . . . . .	36
Figura 17 – sem legenda . . . . .	37
Figura 18 – sem legenda . . . . .	38
Figura 19 – sem legenda . . . . .	38
Figura 20 – <code>pom.xml</code> . . . . .	40
Figura 21 – Estrutura do Projeto . . . . .	41
Figura 22 – Arquivo <code>persistence.xml</code> . . . . .	42
Figura 23 – Classe <code>JpaUtil</code> . . . . .	43
Figura 24 – Classe <code>AlunosService</code> . . . . .	43

## LISTA DE SIGLAS E ABREVIATURAS

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

## **LISTA DE QUADROS**

## **LISTA DE TABELAS**

# SUMÁRIO

INTRODUÇÃO .....	7
2 <b>QUADRO TEÓRICO .....</b>	<b>9</b>
2.1 <i>Java</i> .....	9
2.2 <i>Android</i> .....	9
2.3 <b>Android Studio</b> .....	14
2.4 <i>Web Services</i> .....	14
2.4.1 <b>REST</b> .....	16
2.5 <i>Apache Tomcat</i> .....	17
2.6 <b>PostgreSQL</b> .....	18
2.7 <b>UML</b> .....	19
2.8 <b>Google Cloud Messaging</b> .....	20
2.9 <i>Jersey</i> .....	21
2.10 <i>Hibernate</i> .....	21
3 <b>QUADRO METODOLÓGICO .....</b>	<b>24</b>
3.1 <b>Tipo de pesquisa</b> .....	24
3.2 <b>Contexto de pesquisa</b> .....	24
3.3 <b>Instrumentos</b> .....	25
3.4 <b>Procedimentos e Resultados</b> .....	26
3.4.1 <b>Modelagem</b> .....	26
3.4.2 <i>Google Cloud Messaging</i> .....	26
3.4.3 <b>Aplicativo</b> .....	27
3.4.4 <i>Web service</i> .....	39
REFERÊNCIAS.....	47

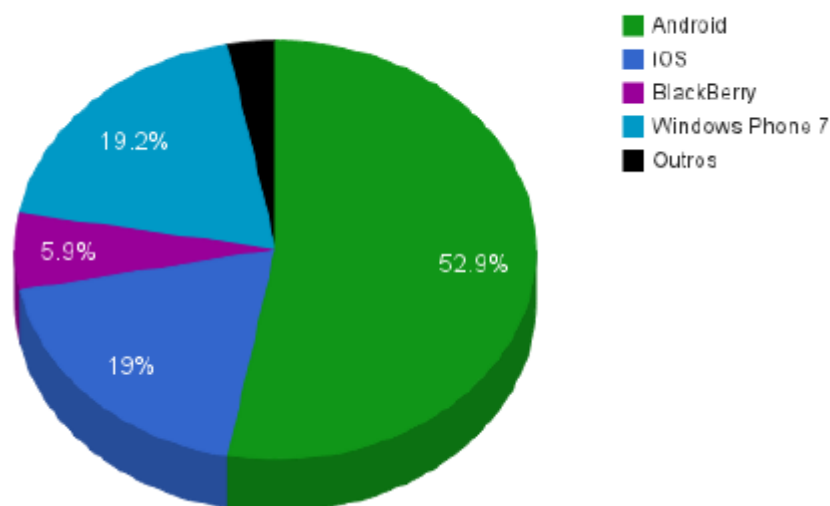
## INTRODUÇÃO

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 quanto os desenvolvedores buscam plataformas modernas e ágeis para a criação de aplicações. Esse fato contribuiu 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*.

Muitas coisas que antes eram feitas apenas em computadores *desktops* já podem ser realizadas nos celulares, como transferências bancárias, localização de taxi, conversas entre 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 além das facilidades que provêem aos desenvolvedores. Na Figura 1, é possível ver um gráfico informando que no ano de 2016, 52,9% dos *smartphones* serão do sistema operacional *Android*.

Participação no mercado



**Figura 1** – Participação de mercado do *Android* em 2016. **Fonte:**Monteiro (2012)

Hoje em dia, há uma gama enorme de aplicativos para *Android* que tem como objetivo resolver problemas específicos. Mendes (2011), vendo as dificuldades encontradas pelas novas



bandas musicais em saber as opiniões de seus fãs referente a *shows* realizados, criou um aplicativo que tornou possível a interação entre eles. Oglio (2013), desenvolveu um utilitário para *Android* que possibilitou aos alunos do Centro Universitário Univates acessarem o portal virtual de sua faculdade.

Pensando-se nas facilidades providas pelos dispositivos móveis em conseguir informações rápida a qualquer hora e local e visando facilitar o acesso dos alunos as suas notas, faltas e provas agendadas, essa pesquisa tem por finalidade o desenvolvimento de um aplicativo para a plataforma *Android* que possibilite aos alunos da Universidade do Vale do Sapucaí receberem notificações e consultarem suas notas, faltas e provas agendadas.

Para alcançar o propósito principal do trabalho, o objetivo geral foi dividido em alguns objetivos específicos, aos quais pode-se citar:

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

Desta maneira, esse trabalho contribui socialmente com os graduandos, pois espera-se agilizar o processo em que eles consultam os resultados dos exercícios avaliativos, faltas e as provas agendadas. O *software* também irá notificá-los no momento em que haver algum lançamento referente as disciplinas cursadas, evitando assim que o estudante tenha que acessar o portal do aluno várias vezes ao dia, ansioso em saber seu rendimento.

O projeto coopera na qualificação dos envolvidos do trabalho, tendo em vista o aumento na procura por profissionais habilitados com tecnologias atuais como Java, *Android*, REST, Hibernate entre outros.

Para a universidade, essa pesquisa à coloca como pioneira nesse quesito e demonstra a sua preocupação com o bem estar de seus alunos, pois existem pessoas que não tem computadores, mas possuem *smartphones*, com isso eles também conseguirão ter suas informações em tempo real.

Na área acadêmica, esse trabalho contribui aos alunos do curso de sistemas de informação servindo lhes como referência para pesquisas futuras ou usá-lo para implementar funcionalidades a este projeto.

## 2 QUADRO TEÓRICO

Neste capítulo serão descritos os principais conceitos e características das tecnologias utilizadas para o desenvolvimento desta pesquisa.

### 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<sup>1</sup>, que liberta os softwares de ficarem presos a um único sistema operacional, uma vez que o programa conversa diretamente com a JVM 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 acensão 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*, 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 te-

---

<sup>1</sup> JVM - Java Virtual Machine

lecomunicações, 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 pasta raiz do projeto. Para Lecheta (2010), as informações devem estar entre *tags* correspondentes ao recurso. 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"/>`.

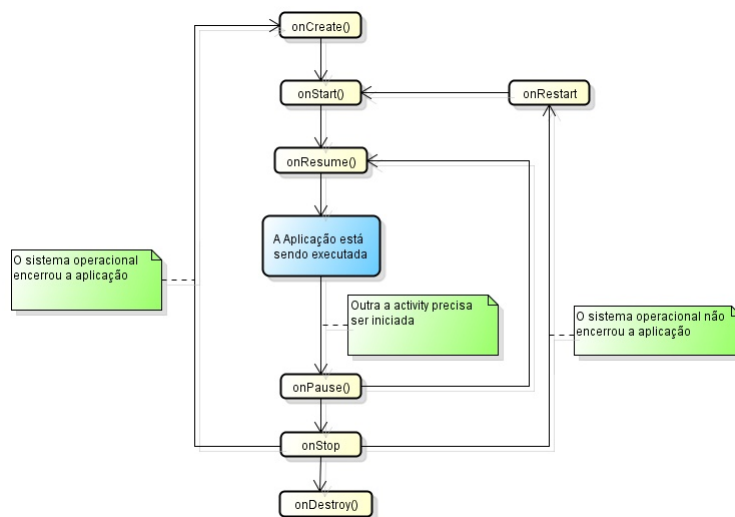
Lecheta (2010) diz que as *intents* são recursos tão importantes que podem ser consideradas como o coração do *Android* e que estão presentes em todas as aplicações. De acordo com K19 (2012, p.29), "são objetos responsáveis por passar informações, como se fossem mensagens, para os principais componentes 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 *app's* 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 de seu 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 2 é demonstrado o ciclo de vida de uma *activity*.



**Figura 2** – 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, então nessa hora o sistema operacional chama o método `onPause()` para interromper 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. Para destruí-la é chamado o método `onDestroy()`. Dessa forma, para executar o aplicativo da Univás será necessário chamar o método `onCreate()` novamente seguindo o ciclo normal, porém se não for encerrada completamente, ao findar a ligação será executado o método `onRestart()` e voltar para a *activity* ao qual o usuário se encontrava.

No arquivo `AndroidManifest.xml` as *activities* devem estar entre as tags `<activity>` e `</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 ficará disponível junto aos outros aplicativos no dispositivo.

A *Activity* a ser utilizada para iniciar a aplicação é uma *Navigation Drawer*, que 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 executados 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 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 na agenda de contatos do dispositivo por um aplicativo e alterado por outro.

Para Monteiro (2012), o *Broadcast Receiver*, é um componente do *Android* responsável por responder a eventos do sistema. Ele não possui interface gráfica e normalmente interage com os usuários através de notificações.

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 ficarão um abaixo do outro e quando estiver na horizontal eles ficarão 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*.
- *ListView*: 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 na parte superior 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 ele 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. Somente é possível acessá-lo pela aplicação a qual o criou e é excluído caso o aplicativo seja removido.

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

Outra ferramenta importante e muito utilizada do *Android* é a notificação. Segundo Philips 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 desenvolvedores.

### 2.3 Android Studio

Uma 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 *IntelliJ 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.

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 à 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<sup>2</sup>. 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 a estrutura 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 3.

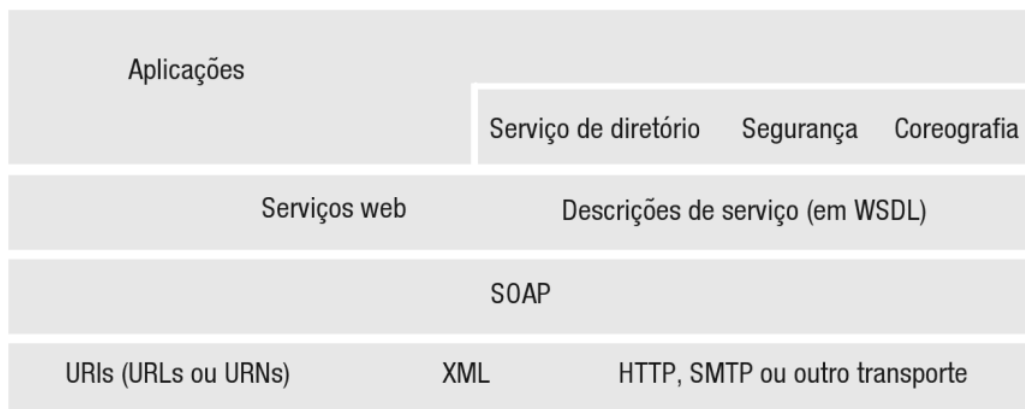
Os *web services* geralmente fazem uso do protocolo SOAP, para estruturar e encapsular as mensagens trocadas. De acordo com Coulouris et al. (2013, p.381), "o protocolo SOAP é projetado para permitir tanto interação cliente-servidor de forma assíncrona pela *Internet*". Segundo Sampaio (2006, p.27), "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*". O elemento

---

<sup>2</sup> HTTP - *HyperText Transfer Protocol*





**Figura 3** – Infraestrutura e componentes dos serviços *web*. **Fonte:**Coulouris et al. (2013)

*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 parâmetros referentes à mesma. É válido lembrar que todas requisições são trocadas usando SOAP, e usam o XML como formato oficial.

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

Além das tecnologias *web services* tradicionais, existem 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, 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 características do padrão REST serão abordadas na próxima seção.

## 2.4.1 REST

Segundo Saudate (2012), REST<sup>3</sup> foi 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 arquitetônicos baseados em rede e que combinado com algumas restrições, fornecem uma interface simples e uniforme para fornecimento de serviços<sup>4</sup>.

<sup>3</sup> REST -*Representational State Transfer* ou Transferência de Estado Representativo.

<sup>4</sup> Tradução e resumo de informações de responsabilidade dos autores da pesquisa.

Rubbo (2015) afirma que os dados e funcionalidades 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. 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 características que segundo Saudate (2013), foram responsáveis pelo sucesso da web:

- 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;

Segundo Godinho (2009), não há um padrão de formato para as trocas de informações, mas as que mais são utilizadas é o XML<sup>5</sup> e o JSON<sup>6</sup>. O REST é o mais indicado para aplicações em dispositivos móveis, devido a agilidade que proporciona na comunicação entre cliente e servidor.

## 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*,

---

<sup>5</sup> XML - *Extensible Markup Language*.

<sup>6</sup> JSON - *JavaScript Object Notation*.

que disponibiliza serviços através de requisições e respostas. Caelum (2015b) afirma que ele é utilizado para aplicações que necessitam apenas da parte *Web* do Java EE<sup>7</sup>.

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 com *Tomcat* é necessária a utilização das seguintes tecnologias:

- 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<sup>8</sup> 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 podem ser citadas: *C/C++*, *PHP*, *Java*, *ASP*, *Python* etc.

---

<sup>7</sup> EE - Sigla para enterprise edition

<sup>8</sup> SGDB - Sistema Gerenciador de Banco de Dados

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).
- Replicação de dados entre servidores.
- *Cluster*.
- *Multithreads*.
- Segurança SSL<sup>9</sup> e criptografia.

É através do *Postgresql* que o *web service* armazenará e posteriormente retornará os dados dos discentes para o aplicativo *Andorid*.

## 2.7 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 década 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, iterativo e incremental". A linguagem de modelagem UML, além de fornecer um vocabulário próprio, também provê uma série de diagramas que tem inúmeras finalidades diferentes.

---

<sup>9</sup> SSL -*Secure Socket Layer*

A linguagem de modelagem UML não é 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.

## 2.8 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, 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<sup>10</sup> 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*<sup>11</sup>: 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 à 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 serve para que a API da Google possa enviar as mensagens para o dispositivo 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*.

---

<sup>10</sup> KB - Kilobytes

<sup>11</sup> Identity

## 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), a 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* projetada para tornar mais fácil desenvolver aplicações que usam a arquitetura REST"<sup>12</sup>. Através desta especificação torna-se mais fácil e ágil a construção de serviços *web* baseados em REST.

Como JAX-RS é apenas uma especificação, ela necessita então de uma implementação. Uma das implementações desta especificação é o *framework Jersey*. Segundo Oracle (2015c) "*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*"<sup>13</sup>. Além das anotações que facilitam seu uso, *Jersey* pode prover serviços com uma infinidade 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 o *Jersey* pode rodar tanto em servidores que implementem a especificação *Servlet* ou não. Este *framework* será usado para contruir 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*<sup>14</sup> usadas nas consultas em bancos de dados relacionais e com a construção do código JDBC<sup>15</sup>, 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

<sup>12</sup> Tradução e resumo de informações de responsabilidade dos autores da pesquisa.

<sup>13</sup> Tradução e resumo de informações de responsabilidade dos autores da pesquisa.

<sup>14</sup> SQL - *Structured Query Language*

<sup>15</sup> JDBC - *Java Database Connectivity*

amarrada em um banco de dados específico e era extremamente custosa uma mudança posterior. 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 estes problemas, é que surgiram os *frameworks* ORM<sup>16</sup> 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<sup>17</sup> 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"<sup>18</sup>.

A especificação JPA foi concebida sendo a terceira parte da especificação EJB<sup>19</sup>, e deveria atender ao propósitos de persistência de dados desta especificação. De acordo com Keith e Schincariol (2009, p.12), JPA é um *framework* leve baseado em POJO's<sup>20</sup>, 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<sup>21</sup>.

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. As principais vantagens do uso do *Hibernate* segundo Sourceforge (2015) são:

- Provedor JPA: além de sua API nativa, o *Hibernate* também é uma implementação da especificação JPA, podendo assim ser facilmente usado em qualquer ambiente de apoio JPA.
- Persistência idiomática: permite que sejam construídas classes persistentes, e que suportem herança e polimorfismo entre outras estratégias, sem a necessidade da construção de estruturas especiais para tal fim.

---

<sup>16</sup> ORM - Object-relational Mapping

<sup>17</sup> JPA - *Java Persistence API*

<sup>18</sup> Tradução de responsabilidade dos autores da pesquisa.

<sup>19</sup> EJB - *Enterprise Java Bean*

<sup>20</sup> POJO - *Plain Old Java Object*

<sup>21</sup> Tradução e resumo de informações de responsabilidade dos autores da pesquisa.

- Performance e suporte: permite que sejam usadas várias estratégias 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 número pequeno de usuários até números mais elevados.
- Confiável: sua confiabilidade e estabilidade são comprovadas pelo seu grande uso e aceitação atualmente.
- Extensível: Hibernate é altamente configurável e extensível<sup>22</sup>.

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 *web service*, em conjunto com o banco de dados.

---

<sup>22</sup> Tradução e resumo de informações de responsabilidade dos autores da pesquisa.



### 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

Marconi e Lakatos (2002, p.15) definem pesquisa como “uma indagação minuciosa ou exame crítico e exaustivo na procura de fatos e princípios”. Gonçalves (2008), por sua vez, conclui que uma pesquisa constitui-se em um conjunto de procedimentos visando alcançar o conhecimento de algo.

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, este projeto enquadra-se no tipo de pesquisa aplicada, pois desenvolveu-se um produto real com intuito de resolver um problema específico, no caso um aplicativo para plataforma *Android* que permita aos alunos da universidade do Vale do Sapucaí, consultarem suas notas, faltas e provas agendadas.

#### 3.2 Contexto de pesquisa

Para que os alunos possam saber suas notas, faltas e provas agendadas, é necessário aos discentes acessarem o portal do aluno para consulta-las.

O *software* desenvolvido nesse trabalho, é um aplicativo para dispositivos móveis com sistema operacional *Android*, o qual tem por finalidade facilitar aos graduandos o acesso as suas informações escolares mais procuradas.


Os alunos acessarão o aplicativo com mesmo usuário e senha do portal do aluno, e quando houver o lançamento de alguma nota ou prova agendada, o estudante será notificado em seu dispositivo. Ao clicar na notificação o sistema lhe apresentará a informação referente.

### 3.3 Instrumentos

Os instrumentos de pesquisa existem para que se possam levantar informações para realizar um determinado projeto.

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.

Nesse trabalho foi realizado um questionário simples, apresentado na Figura 4, 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 afirmaram ter dificuldades para encontrar o que precisavam e que gostariam em ser notificados quando houver alguma atualização de notas. Sobre o motivo do acesso cem por cento dos discentes responderam que entram no sistema *web* para consultar os resultados das avaliações.



Pesquisa sobre o portal do aluno

Qual é sua opinião sobre o portal do aluno?

☐ Ótimo  
☐ Bom  
☐ Ruim  
☐ Péssimo

Qual é sua maior dificuldade para acessar o portal do aluno?

☐ Não tenho acesso a internet  
☐ Demoro para encontrar o que preciso  
☐ O sistema não avisa quando são lançadas as notas  
☐ Outro:

A maior parte das vezes que acesso o portal do aluno é para?

☐ Ver minhas notas  
☐ Ver provas agendadas  
☐ Ver minhas faltas  
☐ Buscar contatos dos professores  
☐ Consultar financeiro  
☐ Consultar material postado pelos professores  
☐ Outro:

Você acha que um aplicativo para celular para acessar o portal seria?

☐ Ótimo  
☐ Bom  
☐ Ruim  
☐ Péssimo

100% concluído

**Figura 4** – Questionário 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 responsabilizou-se a fazer e traçar novas metas. Também foram utilizadas referências de livros, revistas, manuais e *web sites*.

### **3.4 Procedimentos e Resultados**

Após estudar as teorias de desenvolvimento de *software* e integração entre *web service* e aplicativos *Android*, iniciou-se o período de modelagem do sistema.

#### **3.4.1 Modelagem**

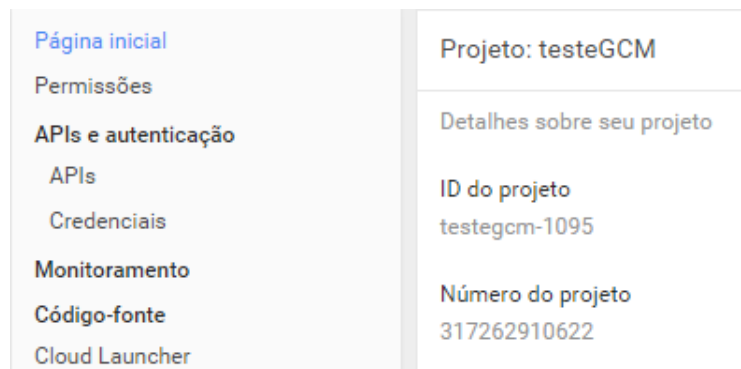
Para atender o objetivo proposto por esta pesquisa, necessitou-se antes modelar o *software* através dos diagramas de UML.

#### **3.4.2 Google Cloud Messaging**

O envio dos dados do *web service* para o aplicativo *Android*, é feito através de um serviço da *Google* conhecido como GCM.

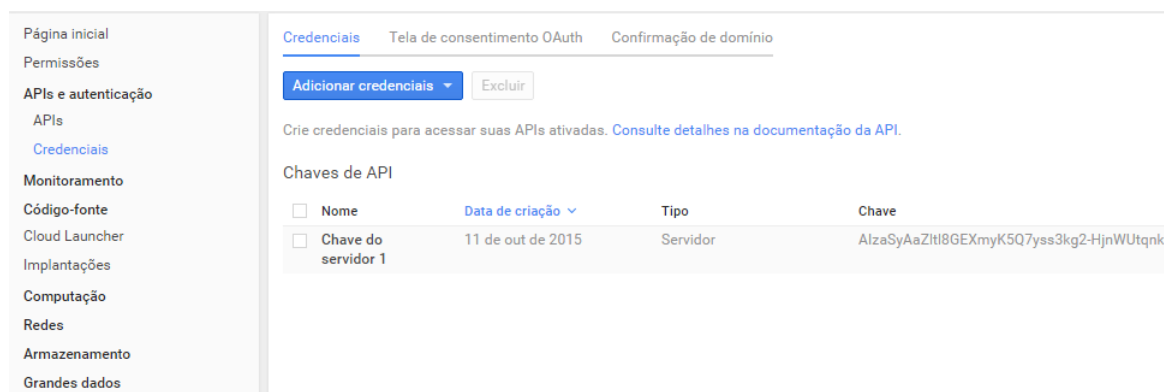
Para que o serviço apresente o resultado esperado, foi preciso acessar o *site* da *Google Developers Console* e criar um novo projeto. Ao criá-lo, foi necessário ir na aba *API's* e ativar a opção *Google Cloud Messaging for Android*.

Com a criação do projeto, a *Google* oferece um número que identificará o *software*, também chamado de *Sender ID*, conforme mostra a Figura 5.



**Figura 5** – Sender ID do GCM. **Fonte:**Elaborado pelos autores.

Por fim, acessou-se a aba Credenciais para indicar o IP do servidor. Ao informa-lo, o serviço gerou uma chave pública a qual foi inserida no *web service*. Na Figura 6, é possível ver o código de acesso criado.



**Figura 6** – Geração da credencial do GCM. **Fonte:**Elaborado pelos autores.

### 3.4.3 Aplicativo

Para iniciar a construção do aplicativo, fez-se necessário a instalação e configuração do ambiente de desenvolvimento. Primeiramente, realizou-se o *download* da IDE *Android Studio*, versão 1.1.0 e do *Android SDK*, versão 24.0.2, ambos no site *Developers Android*.

Contudo, ao executar o emulador do *Android* o sistema apresentava a seguinte mensagem: “*emulator: Failed to open the HAX device!*”. Depois de algum tempo pesquisando, percebeu-se que era necessário instalar um programa chamado *Intel Hardware Accelerated Execution Manager* (HAXM), responsável por aumentar a performance do emulador.

No entanto, ao instalá-lo ocorria o seguinte erro: “*this computer meets the requirements for haxm but intel virtualization technology (VT-x) is not turned on*”. A solução foi acessar a

BIOS da máquina e habilitar o assistente de *hardware* para virtualização. Daí em diante, foi possível executar no emulador as aplicações feitas no *Android Studio*.

Com o ambiente já configurado, criou-se um repositório no controlador de versão *Github*, cujo todos os participantes possuem acesso, para que ambos tenham a versão mais recente do aplicativo em seu dispositivo.

A partir de então, passou-se a desenvolver o *software*. A princípio, foi construída uma *activity*, que é acessível ao aluno logo que a aplicação se inicia. Essa *activity* é do tipo *Navigation Drawer Layout*, ou seja, é um painel que permite inserir as opções de navegação do aplicativo, semelhante a um menu.

Ao criar essa *activity*, o *Android Studio* gera automaticamente a classe *NavigationDrawerFragment* e um arquivo XML na pasta *layout*, chamado *fragment\_navigation\_drawer.xml*.

No arquivo *fragment\_navigation\_drawer.xml* foram inseridos três *widgets*, sendo dois do tipo *textView*, para o cabeçalho com a logomarca da Univás e para o rodapé com o seguinte texto: "Univás - Pouso Alegre - MG" e um *widget* do tipo *listView* que contém a lista com as opções que o *software* oferece ao aluno. Na Figura 7, podem ser vistos os *widgets* no arquivo *fragment\_navigation\_drawer.xml*

```

<TextView
    android:id="@+id/headerView"
    style="?android:attr/textAppearanceLarge"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:drawableLeft="@drawable/logo1"
    android:gravity="center_vertical"
    android:padding="25dp"
    android:text=""
    android:layout_alignParentTop="true"
    android:layout_alignParentStart="true" />

<TextView
    android:id="@+id/footerView"
    style="?android:attr/textAppearanceMedium"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:gravity="center"
    android:padding="20dp"
    android:text="Univás - Pouso Alegre - MG"
    android:textStyle="bold" />

<ListView
    android:id="@+id/navigationItems"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_above="@+id/footerView"
    android:layout_below="@+id/headerView"
    android:background="#cccc"
    android:choiceMode="singleChoice"
    android:divider="@android:color/transparent"
    android:dividerHeight="0dp" />

```

**Figura 7** – sem legenda. **Fonte:**Elaborado pelos autores.

A classe `NavigationDrawerFragment` representa o painel de navegação. Nela se destacam os métodos `onCreateView()`, responsável por criar o *layout* de navegação e o `selectItem()`, encarregado em identificar qual item foi escolhido pelo usuário. Na figura 8, vê-se o método `onCreateView()`, informando ao sistema operacional o *layout* a ser chamado e adicionando a um array de *String* as alternativas de navegação que serão exibidos no `listView` do arquivo `fragment_navigation_drawer.xml`.

```

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

    mDrawerListView = (ListView) view.findViewById(R.id.navigationItems);
    mDrawerListView.setOnItemClickListener((parent, view, position, id) -> {
        selectItem(position);
    });
    mDrawerListView.setAdapter(new ArrayAdapter<String>(
        getActionBar().getThemedContext(),
        android.R.layout.simple_list_item_activated_1,
        android.R.id.text1,
        new String[]{
            "Home",
            "Notas",
            "Faltas",
            "Provas Agendadas",
            "Sair"
        }
    ));
    mDrawerListView.setItemChecked(mCurrentSelectedPosition, true);
    return view;
}

```

**Figura 8** – sem legenda. **Fonte:**Elaborado pelos autores.

O próximo passo, foi criação de uma activity do tipo blank activity com finalidade de listar as notas. Ao criá-la com o nome de ListResultsActivity, o *Android Studio* gera dentro da pasta *layout* o arquivo XML referente a ela, chamado de *activity\_list\_results.xml*. Neste, foi inserido apenas o *widget* *expandableListView*, que está incumbido de apresentar a lista de disciplinas cujo o discente está cursando e ao clicar em alguma dessas matérias serão apresentadas as notas referentes aos exercícios realizados desta disciplina. Na Figura 9 é possível ver o *layout* com uma lista do tipo *expandableListView*.

```

<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="univas.edu.com.university.ListResultsActivity">

    <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 9** – sem legenda. **Fonte:**Elaborado pelos autores.

Depois, fez-se necessário a criação de uma classe encarregada por organizar e controlar todas as atualizações dos itens de uma lista. Essa classe recebeu o nome de `ListResultsAdapter` e estende da classe `BaseExpandableListAdapter`, nativa do *Android*.

Os procedimentos acima citados foram necessários também para as opções de faltas e provas agendadas. Após construídos todos os *layouts*, foi fundamental criar o banco de dados, no qual o aplicativo salva as informações recebidas do *web service*. Para que isso fosse possível, elaborou-se uma classe denominada `DatabaseHelper` que estende da classe `SQLiteOpenHelper` do *Android*, com dois métodos, um chamado `onCreate()` e outro conhecido por `onUpgrade()`.

Foi preciso criar um atributo que mantém a versão do banco de dados. Essa informação serve para que o *Android* consiga saber qual dos dois métodos devem ser executados. Ao iniciar a aplicação pela primeira vez, estando a versão em um, o sistema chamará o método `onCreate()`. Se for preciso atualizar a estrutura do banco, o atributo versão deve ser incrementado em um, de modo que ao executar o *software* o sistema operacional perceba a mudança, chamando o método `onUpgrade()`. Na figura 10 é apresentado a classe `DatabaseHelper`.



```

public class DatabaseHelper extends SQLiteOpenHelper {

    private static final String BANCO_DADOS = "univasDB_version1";
    private static int VERSAO = 1;

    public DatabaseHelper(Context context) { super(context, BANCO_DADOS, null, VERSAO); }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE disciplinas (_id LONG PRIMARY KEY, nome TEXT);");

        db.execSQL("CREATE TABLE eventos (_id LONG PRIMARY KEY, id_disciplina LONG, " +
            " tipo_evento TEXT, descricao_evento TEXT," +
            " data_evento TEXT, valor_evento INTEGER, nota INTEGER," +
            " FOREIGN KEY (id_disciplina) REFERENCES disciplinas (_id) );");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int i, int i2) {

    }
}

```

**Figura 10** – sem legenda. **Fonte:**Elaborado pelos autores.

A fim de estabelecer uma conexão entre o aplicativo e *web service* foi preciso conceder a permissão de acesso à internet no `AndroidManifest.xml` com o seguinte comando:

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

Logo após, criou-se uma classe chamada de `HttpUtil` para ler informações recebidas *web service*. Nela foram inseridos dois métodos, um para identificar as informações referentes as disciplinas cursadas e outro para captar os dados de eventos como notas, faltas e provas agendadas. Na Figura 11 é possível ver o método incumbido de interpretar os elementos das matérias.

```

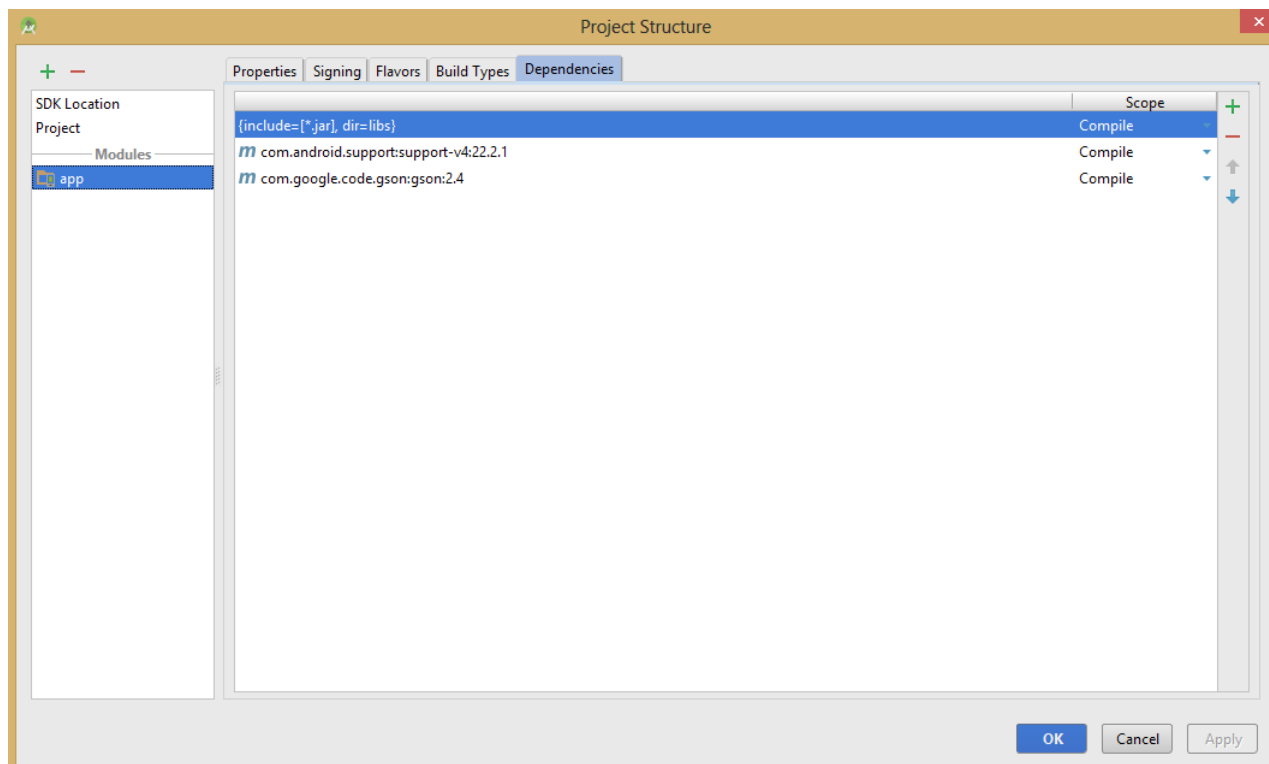
public HttpUtil(DatabaseHelper helper) { this.helper = helper; }

public void getJsonDiscipline(final String url){
    new Thread((Runnable) () -> {
        AlunoDisciplina retorno = null;
        try {
            HttpClient httpClient = new DefaultHttpClient();
            HttpGet request = new HttpGet();
            request.setURI(new URI(url));
            HttpResponse response = null;
            try {
                response = httpClient.execute(request);
            } catch (IOException e) {
                e.printStackTrace();
            }
            InputStream content = null;
            try {
                content = response.getEntity().getContent();
            } catch (IOException e) {
                e.printStackTrace();
            }
            Reader reader = new InputStreamReader(content);
            Gson gson = new Gson();
            retorno = gson.fromJson(reader, AlunoDisciplina.class);
            for (int i = 0; i < retorno.getDisciplinas().size(); i++){
                DatabaseExecute execute = new DatabaseExecute(helper);
                DisciplineTO to = new DisciplineTO();
                to.set_id(retorno.getDisciplinas().get(i).getId());
                to.setNome(retorno.getDisciplinas().get(i).getNome());
                execute.insertDiscipline(to);
            }
            content.close();
        }
    });
}

```

**Figura 11** – sem legenda. **Fonte:**Elaborado pelos autores.

Nos métodos de leitura dos dados foi preciso criar uma thread separada da thread principal do sistema, evitando travar a aplicação enquanto recebe as informações vindas do *web service*. Estes dados estão em formato JSON e foi utilizado a biblioteca Gson para convertê-las no formato esperado. Para utilizá-la foi fundamental adicioná-la como uma dependência do projeto, conforme mostra a Figura 12.



**Figura 12** – sem legenda. **Fonte:**Elaborado pelos autores.

Ao receber as informações do servidor *web* é preciso salvá-las no banco de dados do aplicativo. Para isso desenvolveu-se uma classe chamada *DatabaseExecute*, encarregada de inserir, alterar e buscar os dados dos estudantes no banco de dados. Na Figura 13, pode se ver o método pelo qual são inseridos os eventos ocorridos. Esses eventos podem ser notas, faltas ou provas agendadas.

```
public void insertEvents(EventTO to){
    SQLiteDatabase db = helper.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put("_id", to.get_id());
    values.put("id_disciplina", to.getId_disciplina());
    values.put("tipo_evento", to.getTipo_evento());
    values.put("descricao_evento", to.getDescricao_evento());
    values.put("data_evento", to.getData_evento());
    values.put("valor_evento", to.getValor_evento());
    values.put("nota", to.getNota());

    long result = db.insert("eventos", null, values);

    if(result != -1 ){
        Log.d(TAG, "Evento salvo com sucesso! Tipo evento= " + to.getTipo_evento() + " idDisciplina= " + to.getId_disciplina());
    }else{
        Log.d(TAG, "Erro ao salvar o Evento!");
    }
}
```

**Figura 13** – sem legenda. **Fonte:**Elaborado pelos autores.

O método recebe um objeto do tipo *EventTO*. Para que seja possível a inserção dos da-

dos, Monteiro (2012) afirma, que é necessário recuperar a referência da classe SQLiteDatabase, através do método `getWritableDatabase()`, logo após é instanciada a classe ContentValues, onde é informado o campo da tabela e o valor desejado. Ao concluir é chamado o insert da classe SQLiteDatabase informando o nome da tabela e o objeto da classe ContentValues.

Para listar os resultados dos exames realizados pelos discentes no painel de notas é utilizado o método `getResults()`, da classe DatabaseExecute. Ele retorna uma lista de objetos da classe EventTO. De acordo com Monteiro (2012), para conseguir recuperar as informações armazenadas no banco de dados é preciso adquirir a instância de leitura da classe SQLiteDatabase através do método `getReadableDatabase()`. Por meio dele pode-se realizar a consulta e recebe um Cursor para navegar pelos resultados. Por fim, é composto um objeto do tipo EventTO e inserido na lista. Na Figura 14 é apresentado o método `getResults()`.

```
public List<EventTO> getResults(){
    List<EventTO> notasTO = new ArrayList<>();

    SQLiteDatabase db = helper.getReadableDatabase();
    Cursor cursor =
        db.rawQuery("SELECT _id, id_disciplina, descricao_evento, valor_evento, nota FROM" +
                    " eventos WHERE tipo_evento = 'PROVA APLICADA'",
                    null);
    cursor.moveToFirst();

    for(int i = 0; i<cursor.getCount();i++){
        EventTO nota = new EventTO();

        nota.set_id(cursor.getLong(0));
        nota.setId_disciplina(cursor.getLong(1));
        nota.setDescricao_evento(cursor.getString(2));
        nota.setValor_evento(cursor.getInt(3));
        nota.setNota(cursor.getInt(4));

        notasTO.add(nota);
        cursor.moveToNext();
    }

    cursor.close();

    return notasTO;
}
```

**Figura 14** – sem legenda. Fonte:Elaborado pelos autores.

Para que as informações possam aparecer na tela, a classe `ListResultsActivity` deve informar ao sistema operacional o *layout* a ser chamado através do método `setContentView()`. Por fim, deve recuperar uma instância do *widget* que apresentará os dados, no caso `expandableListView`, e passar para a classe com função de Adapter a lista de disciplinas cursadas e as notas, para que ela possa fazer a organização das informações. Como pode-se ver na Figura 15.

```

public class ListResultsActivity extends Activity {

    private DatabaseHelper helper;
    private DatabaseExecute execute;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_list_results);
        helper = new DatabaseHelper(this);

        execute = new DatabaseExecute(helper);

        ExpandableListView listView = (ExpandableListView) findViewById(R.id.expandableListView2);
        listView.setAdapter(new ListResultsAdapter(this, execute.getDisciplinas(), execute.getResults()));
    }
}

```

**Figura 15** – sem legenda. Fonte:Elaborado pelos autores.

Na classe ListResultsAdaper as informações referentes as matérias são inseridas em um vetor de *string* enquanto as notas são inseridas em uma matriz, conforme ilustra a Figura 16. Ao findar esse processo, utiliza-se o método `getGroupView()` para apresentar os nomes das disciplinas e o método `getChildView()` para mostrar as notas da matéria desejada, como demonstra a Figura 17.

```

public ListResultsAdapter(Context context, List<DisciplineTO> disciplinas, List<EventTO> notas){
    this.context = context;
    this.disciplinas = disciplinas;
    this.notas = notas;
    nomesMateria = new String[disciplinas.size()];
    valoresMateria = new String[disciplinas.size()][notas.size()];

    for (int i = 0; i < disciplinas.size(); i++){
        Long idMateria = disciplinas.get(i).getId();
        String nomeMateria = disciplinas.get(i).getNome();
        nomesMateria[i] = nomeMateria;
        int position = 0;
        for (int y = 0; y < notas.size(); y++) {
            String descricao = notas.get(y).getDescricao_evento();
            int valor = notas.get(y).getValor_evento();
            int nota = notas.get(y).getNota();
            Long idDisciplina = notas.get(y).getId_disciplina();

            if (idMateria == idDisciplina){
                valoresMateria[i][position] = "Descrição: " + descricao + " Valor: " + valor + " Nota: " + nota;
                position++;
            }
        }
    }
}

```

**Figura 16** – sem legenda. Fonte:Elaborado pelos autores.

```

@Override
public View getGroupView(int groupPosition, boolean isExpanded,
                        View convertView, ViewGroup parent) {

    TextView textViewCategorias = new TextView(context);
    textViewCategorias.setText(nomesMateria[groupPosition]);
    textViewCategorias.setPadding(30, 5, 0, 5);
    textViewCategorias.setTextSize(20);
    textViewCategorias.setTypeface(null, Typeface.BOLD);

    return textViewCategorias;
}

@Override
public View getChildView(int groupPosition, int childPosition,
                        boolean isLastChild, View convertView, ViewGroup parent) {

    TextView textViewSubLista = new TextView(context);
    textViewSubLista.setText(valoresMateria[groupPosition][childPosition]);
    textViewSubLista.setPadding(10, 5, 0, 5);

    return textViewSubLista;
}

```

**Figura 17** – sem legenda. **Fonte:**Elaborado pelos autores.

Nesse contexto, criou-se uma classe denominada *GcmControllerUnivas*, que verifica se o aparelho em que o aplicativo está instalado é compatível com os requisitos do GCM. Caso o dispositivo esteja com as configurações recomendadas, é executado o método `registerInBackground()`, para que possa gerar a chave de registro na *Google*. Ao receber essa chave é chamado o método `sendRegistrationIdToBackend()`, encarregado de enviar esse código para o *web service*. Na Figura 18, vê-se o método `registerInBackground()`, recebendo na variável `regid`, a chave de registro do aparelho, realizado pela classe nativa *GoogleCloudMessaging*, ao ser passado o id do projeto gerado na *Google Developers Console*.

```

private void registerInBackground() {
    new AsyncTask<Void, Void, String>() {
        @Override
        protected String doInBackground(Void... params) {
            String msg = "";
            try {
                if (gcm == null) {
                    gcm = GoogleCloudMessaging.getInstance(context);
                }
                regid = gcm.register(SENDER_ID);
                msg = "Dispositivo registrado, registro ID=" + regid;

                //...
                sendRegistrationIdToBackend(regid);

                //...
                storeRegistrationId(context, regid);
            } catch (IOException ex) {
                msg = "Error :" + ex.getMessage();
                //...
            }
            return msg;
        }
    };
}

```

**Figura 18** – sem legenda. **Fonte:**Elaborado pelos autores.

Desta forma, quando o *web service* envia uma informação ao GCM, é transmitido junto aos dados o Registration ID, gerado pelo método *registerInBackground()*, possibilitando ao serviço da *Google*, identificar a qual dispositivo deve conduzir a mensagem.

Ao receber os registros enviados pelo GCM o *broadcastReceiver* chama a classe que apresenta ao usuário a notificação, no entanto, antes de notificá-lo a informação recebida é enviada à classe *HttpUtil* que faz a leitura dos dados e os salva no banco de dados. Ao findar esse processo é analisada se a informação recebida é de notas, faltas ou provas agendadas e chama o método *sendNotification()* que recebe um objeto de *EventTO* e o tipo do evento.

Por fim, adiciona em uma lista de *String* as informações que devem ser apresentadas aos estudantes, passando-as para a *activity* responsável em exibí-las. A notificação, por sua vez, é exposta através do comando retratado na Figura 19, onde identifica-se os atributos da notificação como a ícone que aparecerá, o título e a mensagem. O método *notify()*, é responsável por fazer a notificação aparecer.

```

NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this).setSmallIcon(R.drawable.notification_univas)
    .setContentTitle("Univas informa")
    .setAutoCancel(true)
    .setStyle(new NotificationCompat.BigTextStyle()
        .bigText(msg))
    .setContentText(msg);

mBuilder.setContentIntent(contentIntent);
mNotificationManager.notify(NOTIFICATION_ID, mBuilder.build());

```

**Figura 19** – sem legenda. **Fonte:**Elaborado pelos autores.

### 3.4.4 Web service

Nesse sessão serão descritos os procedimentos realizados para o desenvolvimento do *webservice* responsável por prover os dados necessários ao aplicativo. Além disso serão descritas as configuração necessárias para a montagem do ambiente de desenvolvimento e sua posterior implantação.

#### 3.4.4.1 Montagem do Ambiente de Desenvolvimento

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*.

A princípio foi instalado o *Servlet Container Apache Tomcat* em sua versão de número 7. Esse *Servlet Container* foi instalado pois implementa a API da especificação *Servlets 3.0* do *Java*. Isso era necessário pelo fato que o *framework Jersey* usa *servlets* para disponibilizar serviços REST. Além disso o *Apache Tomcat* foi escolhido, para que o *WebService* pudesse fornecer os serviços necessários para o consumo, na arquitetura REST, que sugere o uso do protocolo HTTP<sup>1</sup> para troca de mensagens, pois além da funcionalidade com *Servlets*, o *Apache Tomcat* também é um servidor HTTP.

O *Apache Tomcat* foi instalado, por meio do *download* de um arquivo compactado no site oficial do mesmo. A instalação consiste apenas em extrair os dados do arquivo em uma pasta da preferência do desenvolvedor. Esta abordagem permitiu a integração do *Apache Tomcat* com o IDE<sup>2</sup> *Eclipse*, que foi usada para o desenvolvimento. Com isto foi possível controlar e monitorar, o servidor de aplicações através da IDE. Além da configuração necessária para integrar o servidor à IDE, nenhuma outra configuração foi necessária.

Como ferramenta para desenvolvimento, foi usada a IDE *Eclipse* na versão 4.4, que é popularmente conhecida como Luna. O processo de instalação e configuração da IDE, se assemelha bastante ao processo de instalação do *Apache Tomcat*, pois somente é necessário fazer o *download* do arquivo compactado que é fornecido na página do projeto, e descompactá-lo no local preterido pelo desenvolvedor.

---

<sup>1</sup> HTTP - Hypertext Transfer Protocol

<sup>2</sup> IDE - *Integrated Development Environment*



Para armazenar os dados gerados e/ou recebidos, foi necessário fazer a instalação do Sistema Gerenciador de Banco de Dados(SGBD) *PostGreSql* na sua versão de número 9.4. Como está sendo usado um sistema operacional baseado em GNU/Linux como ambiente de desenvolvimento, o *PostGreSql* foi instalado através do gerenciador de pacotes da distribuição.

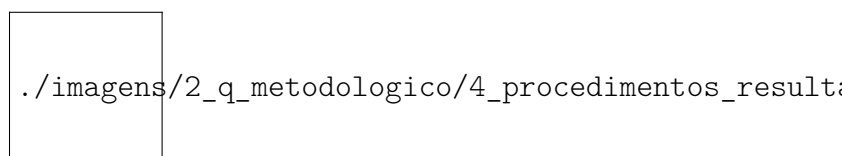
Foi necessário criar um usuário no SGDB que tivesse permissão suficiente apenas para fazer as operações referentes ao banco de dados do *web service*, evitando assim a necessidade de se trabalhar diretamente com um usuário master do SGDB. Esta medida foi tomada visando a segurança do banco de dados, pois com isto foi possível isolar e restringir as responsabilidades deste usuário.

### 3.4.4.2 Desenvolvimento

Com o ambiente de desenvolvimento pronto, podia-se começar de fato a desenvolver. Primeiramente foi necessário criar o banco de dados no SGDB. O banco foi criado, porém sua estrutura não foi definida, pois como será visto mais adiante o *Hibernate*, possui um mecanismo, que com algumas configurações, permite a estruturação do banco de dados, de acordo com o mapeamento objeto relacional. Isto permitirá mudanças na estrutura do banco de dados e suas tabelas, e até mesmo eventuais correções.

Em seguida foi criado um projeto do tipo *web* no *Eclipse* com a ajuda do plugin *Maven*. O *archetype* usado foi *Quick Star WebApp*. Esse Projeto foi criado usando o *Maven* pois depende de uma quantidade considerável de *frameworks*, e uma das principais funcionalidades deste plugin, é ajudar na resolução das dependências de um projeto Java.

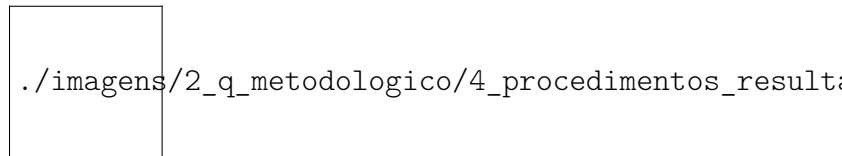
Para tal projeto foi necessário a configuração do *POM.xml* que é o arquivo utilizado pelo *Maven*. Nele estão contidas as configurações relativas à compilação do projeto bem como suas dependências. Na figura 20 a seguir pode ser visto o conteúdo do arquivo *POM.xml*.



**Figura 20** – *pom.xml*. **Fonte:**Elaborado pelos autores.

Com a estrutura do projeto devidamente criada foi possível iniciar os trabalhos com a camada de persistência de dados do projeto. Para este propósito, primeiramente foi criado um pacote, onde ficaram contidas as classes que representam as entidades do ORM. O pacote

recebeu o nome de "br.edu.univas.restapiappunivas.model", pois nele estão contidas as classes que fazem parte do modelo de negócios da aplicação. Este pacote foi criado visando a divisão das responsabilidades internas no projeto, além de contribuir positivamente com a organização do mesmo. Tal pacote e as classes que o compõe estão representados na figura 21.



**Figura 21** – Estrutura do Projeto. **Fonte:**Elaborado pelos autores.

Com este pacote criado, já era possível criar as classes do ORM. Foi criada primeiramente a classe `Student.java`. Para que esta classe pudesse ser reconhecida como um entidade e persistida ao banco de dados através do *Hibernate*, é necessário que esta classe tivesse a anotação `@Entity`. Com isso esta classe já poderia ser entendida como uma entidade e poderia ser persistida no banco de dados, porém além dessa anotação outras foram usadas para que a persistência pudesse ocorrer de forma consistente.

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<sup>3</sup>. Essas classes fazem parte dos mecanismos de persistência de dados e são simplesmente 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.

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.

<sup>3</sup> JPA - Java Persistence API

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 22.

```
<?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 22** – Arquivo `persistence.xml`. **Fonte:**Elaborado pelos autores.

Em seguida à confecção do `persistence.xml` foi criada a classe `JpaUtil` que está representada na Figura 23. 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.

```

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 23** – 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 24.

```

@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 24** – 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<sup>4</sup>. Não foi necessário fazer nenhuma implementação

<sup>4</sup> 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 da saída de dados, faz a conversão de objetos *Java* para JSON. E no caso de entrada transforma 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.

#### **3.4.4.3 Implantação**

## 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.

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.

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.

GONÇALVES, J. A. T. : **O que é pesquisa? Para que?** 2008. Disponível em: <<http://metodologiadapesquisa.blogspot.com.br/2008/06/pesquisa-para-que.html>>. Acesso em: 07 de Outubro de 2015.

GUEDES, G. T. A. : **UML 2 : uma abordagem prática**. 2<sup>a</sup>. 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](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**. 1<sup>a</sup>. ed. São Paulo: Novatec, 2014.

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

LECHETA, R. R. : **Google Android: aprenda a criar aplicações para dispositivos móveis com o android sdk**. 3<sup>a</sup>. 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**. 5<sup>a</sup>. 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](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](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.

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.** 1<sup>a</sup>. 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.** 1<sup>a</sup>. ed. São Paulo: Casa do Código, 2013.

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.