

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

Orientador: Prof. MSc. ROBERTO RI-
BEIRO ROCHA

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

2015

Nunes, Diego D'leon; Rezende, Diógenes Aparecido

APLICATIVO PARA CONSULTA DE NOTAS / Diego D'leon Nunes, Diógenes Aparecido Rezende – Pouso Alegre – MG: Univás, 2015.
103 f. : il.

Trabalho de Conclusão de Curso (graduação) – Universidade do Vale do Sapucaí, Univás, Sistemas de Informação.

Orientador: Prof. MSc. ROBERTO RIBEIRO ROCHA

1. Android. 2. web service. 3. GCM.

DIEGO D'LEON NUNES
DIÓGENES APARECIDO REZENDE

APLICATIVO PARA CONSULTA DE NOTAS

Trabalho de conclusão de curso defendido e aprovado em 07/11/2015 pela banca examinadora constituída pelos professores:

Prof. MSc. ROBERTO RIBEIRO ROCHA
Orientador

Professor ANDRÉ LUIZ MARTINS DE OLIVEIRA
Examinador

Professor ARTUR LUIS RIBAS BARBOSA
Examinador

De Diego D'leon Nunes.

Dedico este trabalho primeiramente a Deus, que me concedeu saúde e motivação para lutar e conquistar meus objetivos. Aos meus pais Mauro Nunes da Silva e Vera Lúcia da Silva, que tanto se dedicaram para que eu pudesse estudar e realizar meus sonhos. Por fim, dedico a todas as pessoas que, de um modo ou outro, contribuíram pelo meu crescimento ético, pessoal e profissional.

De Diógenes Aparecido Rezende.

Dedico este trabalho, primeiramente, a Deus e sua Santíssima Mãe que me deram força e saúde para vencer esta luta, considerada por mim, uma batalha, independente do resultado. Dedico também aos meus pais Joaquim Coutinho de Rezende e Maria José Cirino Rezende, que se dedicaram ao extremo, se privando muitas vezes de coisas necessárias, para meu conforto e para que eu pudesse progredir como pessoa e como profissional. Dedico ao meu irmão Djalma Lúcio Rezende, por ser meu amigo e pela sua ajuda valiosa. Dedico a minha vó Maria do Carmo Cirino por ser para mim um exemplo de paciência e humildade, pois posso afirmar com certeza que foram suas orações que me mantiveram de pé. E por fim, dedico a todas as pessoas do meu dia-a-dia, que passaram por essa batalha comigo.

AGRADECIMENTOS

De Diego D'leon Nunes

Agradeço primeiramente a Deus pelo dom da vida, pela saúde e oportunidade de cursar o ensino superior. Aos meus pais Mauro Nunes da Silva e Vera Lúcia da Silva pelo apoio e exemplo de caráter e honestidade. Aos meus familiares e amigos que me apoiaram e acreditaram em mim. Aos meus colegas de classe que dividiram comigo as dificuldades e as alegrias nesse período de formação acadêmica. Aos funcionários da Fundação do Vale do Sapucaí, principalmente ao seu corpo docente que me ajudou a crescer na ética profissional. Aos professores Msc. José Luiz da Silva e Msc Roberto Ribeiro Rocha que tanto contribuíram para a realização deste trabalho. Por fim, agradeço de maneira geral a todos que de um modo ou outro colaboram em minha formação.

De Diógenes Aparecido Rezende

Qualquer agradecimento que comece sem ser primeiramente a Deus considero que seja em vão. Então agradeço a Deus pela paciência e motivação que Ele me concedeu. Agradeço a minha família pelo amor e compreensão com que me conduziram até este momento. Agradeço também aos professores Msc. José Luiz da Silva e Msc Roberto Ribeiro Rocha pela enorme contribuição não só nesta pesquisa, mas também na minha formação profissional. Agradeço ao senhor Pedro Alexandre Barbosa pela ajuda nos momentos de dificuldade. Agradeço a Maria Izabel Cirino Felipe e a Raphael Ribeiro Cheu da Silva pela confiança que depositaram em mim. Agradeço aos meus amigos pela compreensão nos momentos de ausência. Agradeço a todos os colegas de sala que compartilharam da mesma luta que eu. Agradeço a Diego D'leon Nunes por me acompanhar nesta caminhada e por fim agradeço a todos que contribuíram para a concretização desta etapa da minha vida.

LISTA DE FIGURAS

Figura 1 – Participação de mercado do Android em 2016	14
Figura 2 – Ciclo de Vida de uma activity	19
Figura 3 – Código do arquivo AndroidManifest.xml indicando qual activity deve ser executada quando a aplicação iniciar	20
Figura 4 – Infraestrutura e componentes dos serviços <i>web</i>	24
Figura 5 – Questionário Aplicado	35
Figura 6 – Criando um novo projeto	37
Figura 7 – Inserindo o nome do projeto	37
Figura 8 – Tela de configuração do projeto	38
Figura 9 – Número do projeto	38
Figura 10 – Habilitando GCM para Android	39
Figura 11 – Botão para ativar o GCM ao projeto	39
Figura 12 – Criando as credenciais	40
Figura 13 – Escolhendo a opção Chave de Servidor	40
Figura 14 – Inserindo dados do servidor	41
Figura 15 – Chave de API gerada	41
Figura 16 – Servidor de banco de dados local no PgAdmin	43
Figura 17 – Opção New Database	43
Figura 18 – Tela New Database	44
Figura 19 – Opção Add to Build Path no Eclipse	45
Figura 20 – Classe Student.java	46
Figura 21 – Diagrama de classes do pacote model	48
Figura 22 – Modelo físico do banco de dados	48
Figura 23 – Opção Generate hashCode() and equals()...	49
Figura 24 – Implementação os métodos hashCode() e equals()	50
Figura 25 – Implementação da enumeração EventType	51
Figura 26 – Arquivo persistence.xml	52
Figura 27 – Classe JpaUtil.java	53
Figura 28 – Opção para conversão projeto web par projeto Maven	53
Figura 29 – Arquivo pom.xml	54
Figura 30 – Classe StudentEventsCtrl.java	56

Figura 31 – Classe StudentEvent.java	57
Figura 32 – Classe StudentEvents.java	58
Figura 33 – Classe StudentDisciplines.java	58
Figura 34 – Classe StudentDisciplinesCtrl.java	59
Figura 35 – Trecho do arquivo pom.xml com a configuração do Jersey	60
Figura 36 – Trecho do arquivo web.xml com a configuração do Jersey	60
Figura 37 – Classe StudentResouce.java	61
Figura 38 – Classe UserResouce.java	62
Figura 39 – Classe UserCtrl.java	63
Figura 40 – Classe JerseyProvider.java	64
Figura 41 – Classe ContentMessageGCM.java	66
Figura 42 – Classe PostToGCM.java	67
Figura 43 – Classe SendMessageGCM.java	68
Figura 44 – Classe ListenerAtualizations.java	69
Figura 45 – Configuração do Listener no arquivo web.xml	70
Figura 46 – Diagrama de arquitetura do aplicativo	71
Figura 47 – Diagrama de casos de uso	72
Figura 48 – Código dos widgets do arquivo fragment_navigation_drawer.xml	74
Figura 49 – Método onCreateView()	75
Figura 50 – método selectItem()	75
Figura 51 – Classe DatabaseHelper	76
Figura 52 – Método de inserção de eventos	77
Figura 53 – Método getResults()	78
Figura 54 – Código da classe HttpUtilUnivas que faz a leitura dos eventos	79
Figura 55 – Adicionando uma dependência ao projeto	80
Figura 56 – Adicionando a biblioteca GSON ao projeto	80
Figura 57 – Usando GSON para conversão dos dados	80
Figura 58 – Construtor da classe ListResultsAdapter	81
Figura 59 – Método getGroupView()	82
Figura 60 – Método getChildView()	82
Figura 61 – Código XML do layout que apresentará a lista de notas	83
Figura 62 – Método onCreate() da classe ListResultsActivity	83
Figura 63 – Método sendNotification()	85
Figura 64 – Classe GcmBroadcastReceiverUnivas	86

Figura 65 – Configuração do BroadcastReceiver no AndroidManifest.XML	86
Figura 66 – Método checkPlayServices()	87
Figura 67 – Método getRegistrationId()	87
Figura 68 – Método registerInBackground()	88
Figura 69 – Tela principal do aplicativo com as opções de navegação	91
Figura 70 – Lista de disciplinas	92
Figura 71 – Notificação na barra de notificações do dispositivo.	92
Figura 72 – Paleta de notificação estendida	93
Figura 73 – Tela exibindo os dados após clicado na notificação	93

LISTA DE SIGLAS E ABREVIATURAS

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
API	<i>Application Programming Interface</i>
CPA	Comissão Própria de Avaliação
EE	<i>Enterprise Edition</i>
EJB	<i>Enterprise Java Bean</i>
GCM	<i>Google Cloud Messaging</i>
HAXM	<i>Hardware Accelerated Execution Manager</i>
HTML	<i>HyperText Markup Language</i>
HTTP	<i>HyperText Transfer Protocol</i>
ID	<i>Identity</i>
IDE	<i>Integrated Development Environment</i>
I/O	<i>Input/Output</i>
JDBC	<i>Java Database Connectivity</i>
JPA	<i>Java Persistence Api</i>
JPQL	<i>Java Persistence Query Language</i>
JSON	<i>JavaScript Object Notation</i>
JVM	<i>Java Virtual Machine</i>
KB	<i>kilobyte</i>
OMT	<i>Object Modeling Technique</i>
OOSE	<i>Object-Oriented Software Engineering</i>
ORM	<i>Object-relational mapping</i>
POJO	<i>Plain Old Java Object</i>
POM	<i>Project Object Model</i>
REST	<i>Representational State Transfer</i>
SDK	<i>Software Development Kit</i>
SGBD	Sistema de Gerenciamento de Banco de Dados
SOAP	<i>Simple Object Access Protocol</i>

SQL	<i>Structured Query Language</i>
SSL	<i>Secure Socket Layer</i>
UML	<i>Unified Modeling Language</i>
UNIVAS	Universidade do Vale do Sapucaí
URI	<i>Uniform Resource Identifier</i>
URL	<i>Universal Resource Locator</i>
XML	<i>Extensible Markup Language</i>
WSDL	<i>Web Services Description Language</i>

NUNES, Diego D'leon; REZENDE, Diógenes Aparecido. **APLICATIVO PARA CONSULTA DE NOTAS**. 2015. Monografia – Curso de SISTEMAS DE INFORMAÇÃO, Universidade do Vale do Sapucaí, Pouso Alegre – MG, 2015.

RESUMO

Atualmente, existe uma grande quantidade de tecnologias disponíveis no mercado e sabe-se também que, com a popularização dos *smartphones* e *tablet* tornou-se comum utilizar dispositivos móveis para receber informações em tempo real. Neste contexto, nesta pesquisa foi desenvolvido um *web service* para disponibilizar as informações aos discentes da Univás através de serviços. Também foi criado um aplicativo na plataforma Android que consome o serviço do *web service*. O *app* permite aos alunos da Universidade do Vale do Sapucaí consultarem suas informações acadêmicas como notas, faltas e provas agendadas do semestre corrente. No momento em que um professor lançar um evento, o servidor transmite a mensagem para a API Google Cloud Messaging (GCM) que se responsabiliza em entregar os dados para o aplicativo Android. Este trabalho enquadra-se no tipo de pesquisa aplicada, pois foi desenvolvido um produto real com propósito de resolver um problema específico. A plataforma Android foi escolhida pelo seu destaque no mercado.

Palavras-chave: Android. web service. GCM.

NUNES, Diego D'leon; REZENDE, Diógenes Aparecido. **APLICATIVO PARA CONSULTA DE NOTAS**. 2015. Monografia – Curso de SISTEMAS DE INFORMAÇÃO, Universidade do Vale do Sapucaí, Pouso Alegre – MG, 2015.

ABSTRACT

Currently, there is a lot of technology available in the market and it is also known that with the popularization of smartphones and tablets has become common to use of mobile devices to receive information in real time. In this context, in the research was developed a web service to provide information from Univas by this service. It was also created an application on the Android platform that consumes the use of web service. The app allows students at the University of Vale do Sapucaí consult their academic information such as grades, absences and tests of the current semester. When a teacher launch an event, the server transmits the message to API Google Cloud Messaging (GCM) that is responsible for the data delivery to the Android application. This work fits in the type of applied research because it was developed a real product with purpose to solve a specific problem. The Android platform was chosen for its prominent position in the market.

Key words: Android. web service. GCM.

SUMÁRIO

INTRODUÇÃO	14
2 QUADRO TEÓRICO	17
2.1 Java	17
2.2 Android	17
2.2.1 Elementos Gráficos	21
2.3 Android Studio	22
2.4 Web Services	23
2.4.1 REST	25
2.5 Apache Tomcat	26
2.6 PostgreSQL	27
2.7 UML	28
2.8 Google Cloud Messaging(GCM)	28
2.9 Jersey	29
2.10 Hibernate	30
2.11 Maven	32
3 QUADRO METODOLÓGICO	33
3.1 Tipo de pesquisa	33
3.2 Contexto de pesquisa	33
3.3 Instrumentos	34
3.4 Procedimentos e Resultados	36
3.4.1 Lenvantamento de Requisitos	36
3.4.2 Google Cloud Messaging (GCM)	37
3.4.3 Web service	41
3.4.4 Aplicativo	70
4 DISCUSSÃO DE RESULTADOS	89
5 CONCLUSÃO	94
REFERÊNCIAS.....	98
APÊNDICES	100
ANEXOS	103

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 de aplicativos 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* terão o sistema operacional Android.

Participação no mercado

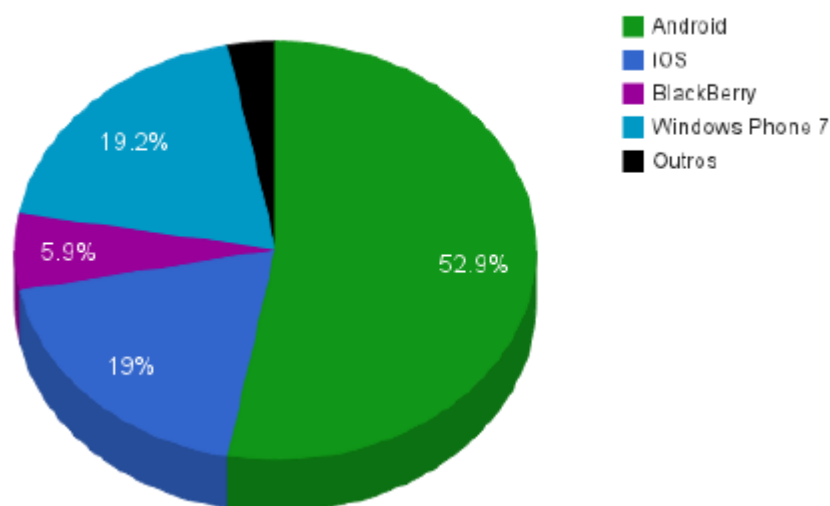


Figura 1 – Participação de mercado do Android em 2016. **Fonte:**Monteiro (2012)

Hoje em dia, há uma gama enorme de aplicativos para Android que têm 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.

Ciente das facilidades proporcionadas pelos dispositivos móveis em conseguir informação rapidamente, a qualquer hora e local e visando facilitar o acesso dos alunos às suas notas, faltas e provas agendadas, esta pesquisa tem por objetivos, desenvolver 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 do semestre corrente, bem como desenvolver um *web service* que provê uma estrutura para a Univás disponibilizar seus dados através de serviço.

Para alcançar o propósito principal do trabalho, o objetivo geral foi dividido em alguns objetivos específicos, os quais podem-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* que possibilitará à Univás disponibilizar suas informações através de serviços e implementar um serviço para que o aplicativo Android consulte as informações relativas a notas, faltas e provas agendadas.

Desta maneira, este trabalho contribui socialmente com os graduandos, pois espera-se agilizar o processo para que eles consultem os resultados dos exercícios avaliativos, faltas e as provas agendadas. O software também irá notificá-los no momento em que houver algum lançamento referente às 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 com o trabalho, tendo em vista o aumento na procura por profissionais habilitados com tecnologias atuais como Java, Android, REST¹, Hibernate entre outros.

Para a universidade, esta pesquisa a coloca como pioneira neste quesito e demonstra a sua preocupação com o bem estar de seus alunos, pois existem pessoas que não têm computadores, mas possuem *smartphones*, com isso eles também conseguirão ter suas in-

¹ REST - *Representational State Transfer* ou Transferência de Estado Representativo.

formações. O *web service* coloca ao alcance da Univás a estrutura para disponibilizar seus dados através de serviço.

Na área acadêmica, este trabalho contribui com os alunos do curso de Sistemas de Informação servindo lhes como referência para pesquisas futuras ou usá-lo para adicionar funcionalidades a este projeto.

Este trabalho é composto de cinco capítulos, sendo que o atual faz uma breve introdução sobre o tema proposto e seus objetivos. No segundo capítulo são descritas as teorias sobre as tecnologias utilizadas para o desenvolvimento do projeto. O terceiro, relata as metodologias usadas nesta pesquisa, bem como que tipo de pesquisa foi desenvolvido assim como o contexto da pesquisa, instrumentos e procedimentos. No quarto capítulo é realizada a discussão de resultados ressaltando o que se obteve com a realização do trabalho. Por fim, se apresenta a conclusão.

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 a objetos, baseada na linguagem C++, desenvolvida pela empresa Sun Microsystems 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 JVM¹, 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 a 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 ascensão 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 softwa-

¹ JVM - Java Virtual Machine

res, 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 existem 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 sejam formalmente aceitos pelo usuário nos termos de uso, ao instalá-los.

As configurações de uma aplicação na plataforma Android ficam salvas em um arquivo XML² denominado `AndroidManifest.xml`, que se localiza na pasta raiz do projeto. Para Lecheta (2010), as informações devem estar entre *tags* correspondentes ao recurso.

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), "Intents 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 é 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

² XML - Extensible Markup Language.

a ligação será executado o método `onRestart()` e voltar para a *activity* na qual o usuário se encontrava.

No arquivo `AndroidManifest.xml` as *activities* devem estar contidas dentro das 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 ficará disponível junto aos outros aplicativos no dispositivo. Na figura 3 é apresentado o código do arquivo `AndroidManifest.xml`. Nela, pode-se ver o nome da classe que será iniciada e no atributo *label* o nome que aparecerá na tela para o usuário.

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

Figura 3 – Código do arquivo `AndroidManifest.xml` indicando qual *activity* deve ser executada quando a aplicação iniciar. **Fonte:**Elaborado pelos autores

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 ficam normalmente escondidas aparecendo apenas quando clicado no canto superior esquerdo.

Segundo Lecheta (2010), a classe *Service* existe com o intuito de 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), um *Content Provider* provê conteúdos de forma pública para todas as aplicações, possibilitando aos aplicativos consultar, salvar, deletar e alterar informações no *smartphone*. Assim afirma Lecheta (2010, p.413) “o Android tem uma série de provedores de conteúdos 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.

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-se comunicar com o usuário, o aplicativo cria uma notificação. Normalmente as notificações aparecem na barra superior, as quais podem ser acessadas arrastando-as para baixo a partir da parte superior da tela. Assim que o usuário clica na notificação ela cria uma *activity* para abrir aplicação em questão.

O Android traz embarcado em sua plataforma o banco de dados `SQLite`, que armazena tabelas, *views*, índices, *triggers* em apenas um arquivo em disco. Somente é possível acessá-lo pela aplicação a qual o criou e é deletado caso o aplicativo seja removido.

Na seção abaixo serão descritos os elementos gráficos presentes no Android.

2.2.1 Elementos Gráficos

Em uma aplicação, um elemento fundamental é a interface gráfica, que deverá ser organizada, simples e elegante. O Android organiza os elementos gráficos (*widgets*) através de *layouts*. Conforme Monteiro (2012) esses são os principais *layouts* do sistema operacional Android:

- *LinearLayout*: permite posicionar os elementos em forma linear. Dessa forma se o *layout* for configurado com orientação vertical os itens ficaram um abaixo do outro, porém se estiver configurado com orientação horizontal eles ficaram um ao lado do outro.
- *RelativeLayout*: permite posicionar elementos de forma relativa, ou seja um item com relação a outro.
- *TableLayout*: permite criar *layouts* em formato de tabelas. O elemento *TableRow* representa uma linha da tabela e seus filhos as células. Dessa maneira, caso um *TableRow* possua dois itens significa que essa linha tem duas colunas.

- *DatePicker*: *widget* desenvolvido para a seleção de datas que podem ser usadas diretamente no *layout* ou através de caixas de diálogo.
- *Spinner*: *widget* que permite a seleção de itens, similar ao *combobox*.
- *ListView*: permite exibir itens em uma listagem. Dessa forma, em uma lista de compras, clicando em uma venda, é possível ver os detalhes do item selecionado.
- *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 feita uma animação informando ao usuário o progresso da operação.
- *ExpandableListView*: exibe os itens em forma de uma lista 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.

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

Um 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 (2015c), 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 transita pela internet, é necessário um nível muito alto de integração entre as diversas plataformas, tecnologias e sistemas. Como uma prováveis soluções 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 HTTP³. Sendo uma tecnologia não amarrada a fornecedor, o SOAP disponibilizou uma alternativa atrativa em relação aos protocolos proprietários tradicionais, tais como CORBA e DCOM.

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.

³ HTTP - *HyperText Transfer Protocol*

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 4.

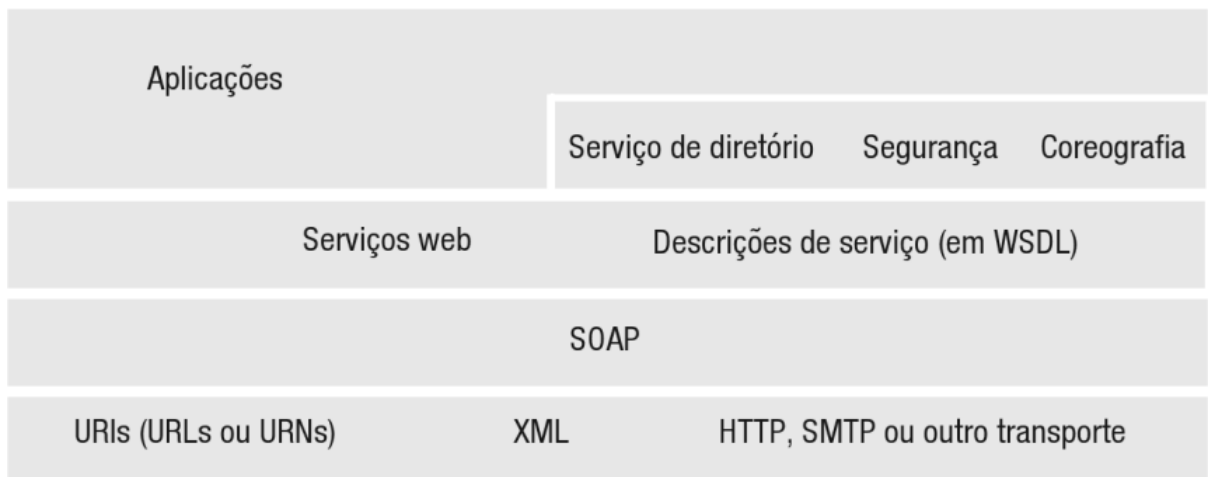


Figura 4 – Infraestrutura e componentes dos serviços *web*. **Fonte:**Coulouris et al. (2013)

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 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 *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, provêm transparência na troca de informações. Isso contribui para que as novas aplicações consigam se comunicar com aplicações mais antigas ou aplicações construídas 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 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, fornece uma interface simples e uniforme para fornecimento de serviços⁴.

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 construído na arquitetura REST baseia-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 do protocolo mais usados na *web*, o HTTP, pode-se dizer que o REST foi concebido para rodar sobre este

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

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 efetiva de *media types*, sendo o mais comum JSON;
- 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 são o XML e o JSON⁵. O REST é o mais indicado para aplicações em dispositivos móveis, devido à agilidade que proporciona na comunicação entre cliente e servidor. Além disso outra característica importante é a simplicidade que o mesmo proporciona no manuseio das informações.

2.5 Apache Tomcat

De acordo com Tomcat (2015), Apache Tomcat é uma implementação de código aberto das especificações Java Servlet e Java Server Pages. O Apache Tomcat é um *Servlet Container*, 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 EE⁶.

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.

⁵ JSON - *JavaScript Object Notation*.

⁶ EE - Sigla para enterprise edition

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 SGBD⁷ 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 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.

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 SSL⁸ e criptografia.

⁷ SGDB - Sistema Gerenciador de Banco de Dados.

⁸ SSL -*Secure Socket Layer*

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.

A linguagem de modelagem UML não exige um processo muito 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 uso de um projeto, será utilizada a linguagem de modelagem UML para o desenvolvimento desta pesquisa.

2.8 Google Cloud Messaging(GCM)

Para que os alunos sejam notificados quando houver alguma mudança no portal do aluno, será utilizada uma API oferecida pela Google denominada Google Cloud Messaging ou simplesmente GCM, 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 KB⁹ 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*¹⁰: 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*.

2.9 Jersey

Atualmente o padrão arquitetural REST para desenvolvimento de serviços *web* vem sendo bastante adotado. 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"¹¹. Através desta especificação torna-se mais fácil e ágil a construção de serviços *web* baseados em REST.

⁹ KB - Kilobytes

¹⁰ Identity

¹¹ Tradução e resumo de informações de responsabilidade dos autores da pesquisa.

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 construir serviços *web RESTful* usando a linguagem de programação Java"¹². 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 vários 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 construir 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 se perdia muito tempo com a confecção de *queries* SQL usadas nas consultas em bancos de dados relacionais e com a construção do código JDBC¹³, 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 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* ORM¹⁴ 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 JPA¹⁵ 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"¹⁶.

¹² Tradução e resumo de informações de responsabilidade dos autores da pesquisa.

¹³ JDBC - *Java Database Connectivity*

¹⁴ ORM - *Object-relational Mapping*

¹⁵ JPA - *Java Persistence API*

¹⁶ Tradução de responsabilidade dos autores da pesquisa.

A especificação JPA foi concebida sendo a terceira parte da especificação EJB¹⁷, e deveria atender ao propósito de persistência de dados desta especificação. De acordo com Keith e Schincariol (2009, p.12), JPA é um *framework* leve baseado em POJO's¹⁸, 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 escaláveis¹⁹.

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) "o 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 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.
- *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.

¹⁷ EJB - *Enterprise Java Bean*

¹⁸ POJO - *Plain Old Java Object*

¹⁹ Tradução e resumo de informações de responsabilidade dos autores da pesquisa.

- Extensível: Hibernate é altamente configurável e extensível²⁰.

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.

2.11 Maven

Maven é uma famosa ferramenta de automação de compilação, e empacotamento e gerência de dependências para projetos Java. Nasceu com o objetivo de simplificar o processo de compilação do projeto Jakarta *Turbine*. Possui inúmeras funcionalidades, sendo que as mais comuns são gerência de dependências de um projeto e os processos de *build* do mesmo.

Apache (2015) afirma que o Maven é baseado no conceito de um *Project Object Model* (POM), e pode gerenciar um projeto de construção, elaboração de relatórios e documentação de um software. Lecheta (2015) ainda complementa ao afirmar que , "a configuração de um projeto Maven é feita no arquivo pom.xml, que descreve os dados de seu projeto, suas dependências e várias outras configurações".

A principal funcionalidade do Maven é a gerência de dependências de um projeto Java. Lecheta (2015) afirma que "ao adicionar uma dependência ao projeto, o Maven faz o *download* desta dependência diretamente de um repositório mundial de projetos conhecido com Maven *Central Repository*". Se necessário ainda resolve dependências transitivas, ou seja, dependências de dependências do projeto. De acordo com Apache (2015) existem três ciclos de vida de compilação para um projeto que usa Maven:

- Ciclo de vida *default*: lida com a implantação do projeto;
- Ciclo de vida *clean*: lida com a limpeza do projeto;
- Ciclo de vida *site*: lida com a criação de documentação do projeto.

Com o intuito de facilitar o desenvolvimento do *web service*, haja vista a quantidade muito elevada de dependências, a ferramenta Maven foi usada no projeto Java *web*.

²⁰ Tradução e resumo de informações de responsabilidade dos autores da pesquisa.

3 QUADRO METODOLÓGICO

Neste 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), a 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 trabalho enquadra-se no tipo de pesquisa aplicada, pois foi desenvolvido um produto real com intuito de resolver um problema específico, que no caso foi um *web service* para que a Univás possa disponibilizar seus dados através de serviços e 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 que eles acessem o portal do aluno para consultá-las.

O *web service*, criado através desta pesquisa, tem por objetivo fornecer a estrutura para que a Univás possa disponibilizar suas informações através de serviços, bem como desenvolver um serviço para que o aplicativo consulte as notas, faltas e provas agendadas dos alunos.

A aplicação Android, por sua vez, tem por finalidade facilitar aos estudantes o acesso às suas informações escolares mais procuradas.

Os alunos acessarão o aplicativo com o mesmo usuário e senha do portal do aluno, e quando houver o lançamento de alguma nota ou prova agendada, eles serão notificados em seu dispositivo. Ao clicar na notificação o sistema apresentará a informação recebida.

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.

Neste trabalho foi realizado um questionário simples, apresentado na Figura 5, contendo quatro perguntas e enviado para *e-mails* de alguns alunos da universidade. O foco desse questionário foi 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 afirmou ter dificuldades para encontrar as informações de que necessitam, e que gostariam de ser notificados quando houvesse 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.

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 as quais cada integrante responsabilizou-se a fazer, além de traçar novas metas. Também foram utilizadas referências de livros, revistas, manuais e *web sites*.



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:

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

[Enviar](#)

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

3.4 Procedimentos e Resultados

Após o estudo das teorias de desenvolvimento de software e integração entre serviços e aplicativos Android, iniciou-se o período de desenvolvimento do sistema.

3.4.1 Lenvantamento de Requisitos

Ao decidir-se por esta pesquisa foi preciso entender os pontos necessários para o bom funcionamento do software. As soluções geradas por este trabalho tendem a atender os seguintes requisitos:

- Disponibilidade: O *web service* foi projetado para criar uma estrutura pela qual a universidade pudesse disponibilizar informações através de serviços. Por isso, o servidor não pode ficar impossibilitado de responder as requisições por causa de falhas no sistema desenvolvido.
- Simplicidade: Segundo as respostas obtidas através do questionário aplicado, os estudantes encontram dificuldades para encontrar o que procuram quando acessam o portal do aluno. Portanto, o aplicativo Android, deve possuir uma interface simples e objetiva para levar as informações aos usuários.
- Notificação: Geralmente, o aluno acessa várias vezes ao dia o portal do aluno para saber se foi postada sua nota. Pensando nisso, o software precisa avisá-lo através de uma notificação quando uma nova informação for lançada no portal do aluno e ao clicar nesta notificação, devem ser apresentados a ele os dados lançados.
- Velocidade: Os dispositivos móveis conseguem informações em tempo real. Também, estando ciente de que as pessoas utilizam os dispositivos móveis para ganhar tempo, é indispensável que as informações cheguem ao aluno o mais breve possível.

Tendo estes paradigmas em mente, passou-se a desenvolver o software, como pode ser acompanhado nas seções seguintes.

3.4.2 Google Cloud Messaging (GCM)

No momento em que é lançada alguma nota, falta ou prova agendada, o *web service* precisa transmitir esta informação para o aplicativo Android. Para que esta comunicação aconteça foi utilizado o serviço da Google chamado de Google Cloud Messaging (GCM).

Neste contexto, o servidor *web* envia uma mensagem para o GCM com as informações que precisa passar para a aplicação *mobile*. A partir daí, a entrega dos dados para os dispositivos móveis fica por conta da Google.

Para que o GCM apresentasse o resultado esperado, foi preciso acessar o site Google *Developers Console* através do endereço <<https://console.developers.google.com>> e construir um novo projeto. Para criá-lo, bastou clicar no botão *Create Project* que está na página inicial, conforme pode se ver na Figura 6. Logo após, foi adicionado um nome ao projeto e clicado no botão criar, como mostra a Figura 7.

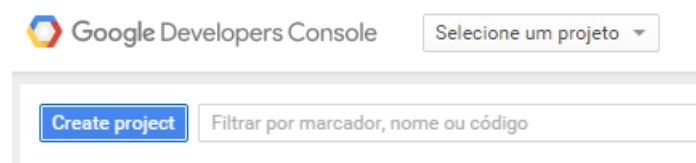


Figura 6 – Criando um novo projeto. **Fonte:**Elaborado pelos autores.

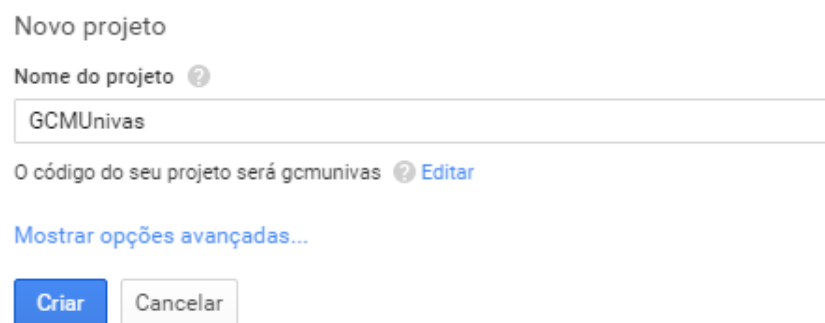


Figura 7 – Inserindo o nome do projeto. **Fonte:**Elaborado pelos autores.

Ao criar o projeto foi aberta uma tela para sua configuração, ilustrada na Figura 8.

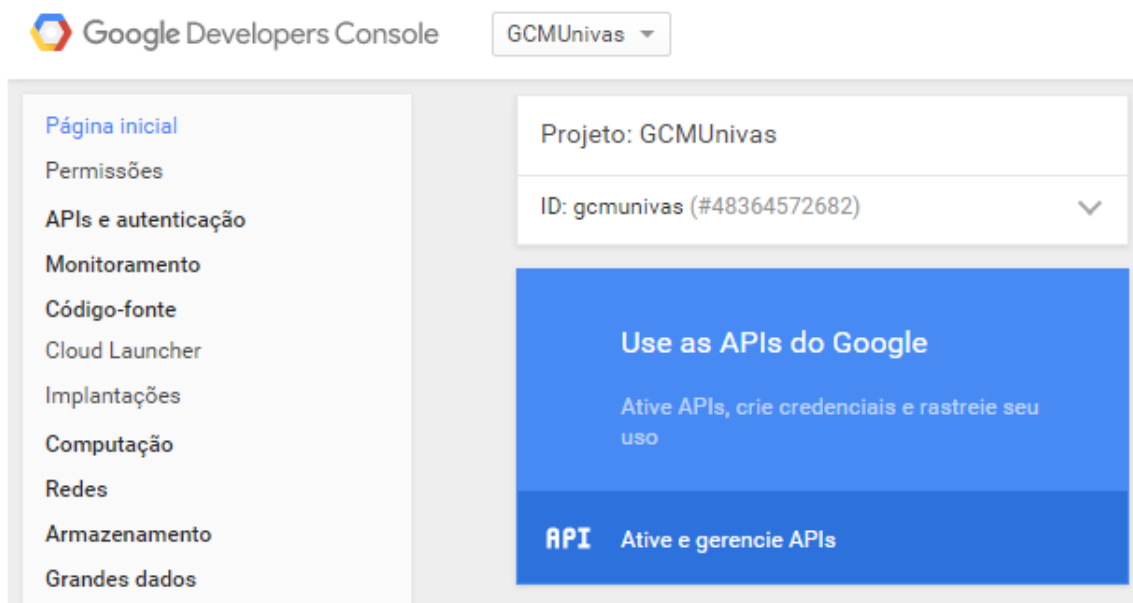


Figura 8 – Tela de configuração do projeto. **Fonte:**Elaborado pelos autores.

O primeiro dado que se obteve foi o número do projeto, também chamado de *Sender ID*. Este código serve para que a Google reconheça a aplicação que enviou a mensagem. Para visualizar este identificador, foi preciso clicar nos detalhes do projeto na página inicial, como se vê na Figura 9.

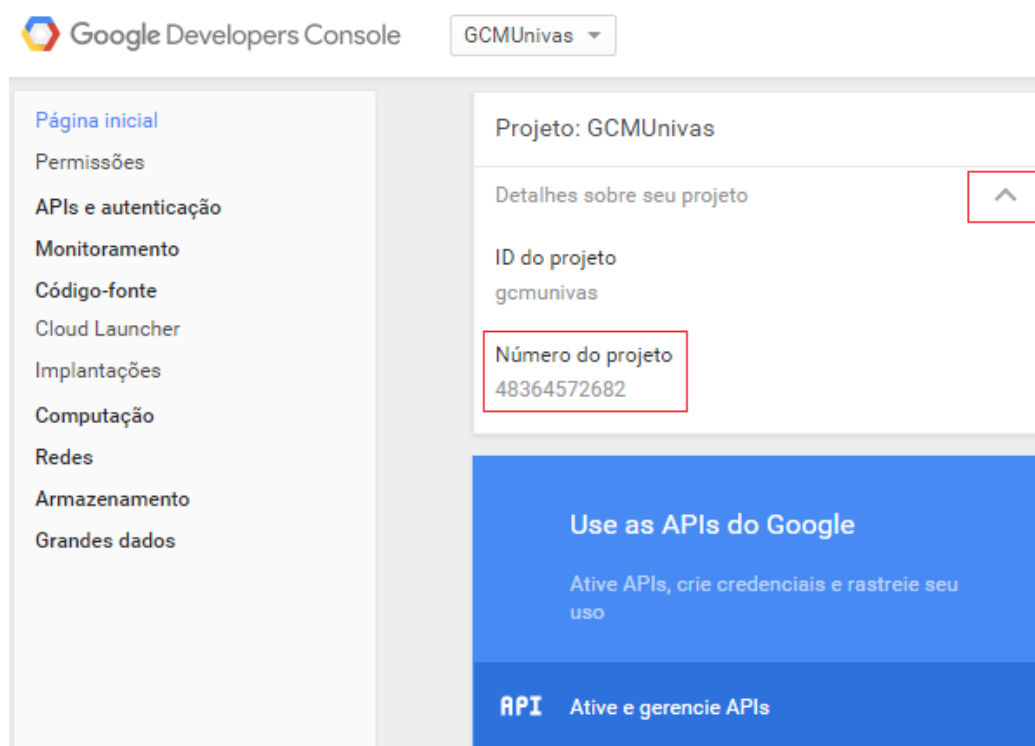


Figura 9 – Número do projeto. **Fonte:**Elaborado pelos autores.

O próximo passo, foi habilitar a API GCM para trabalhar com o projeto. Para essa etapa, foi necessário ir à aba APIs e autenticação, selecionando a opção APIs, conforme indica a Figura 10. Na tela presente aparecem os serviços fornecidos pelo Google. Neste caso optou-se por *Cloud Messaging for Android*, também ilustrado na Figura 10.

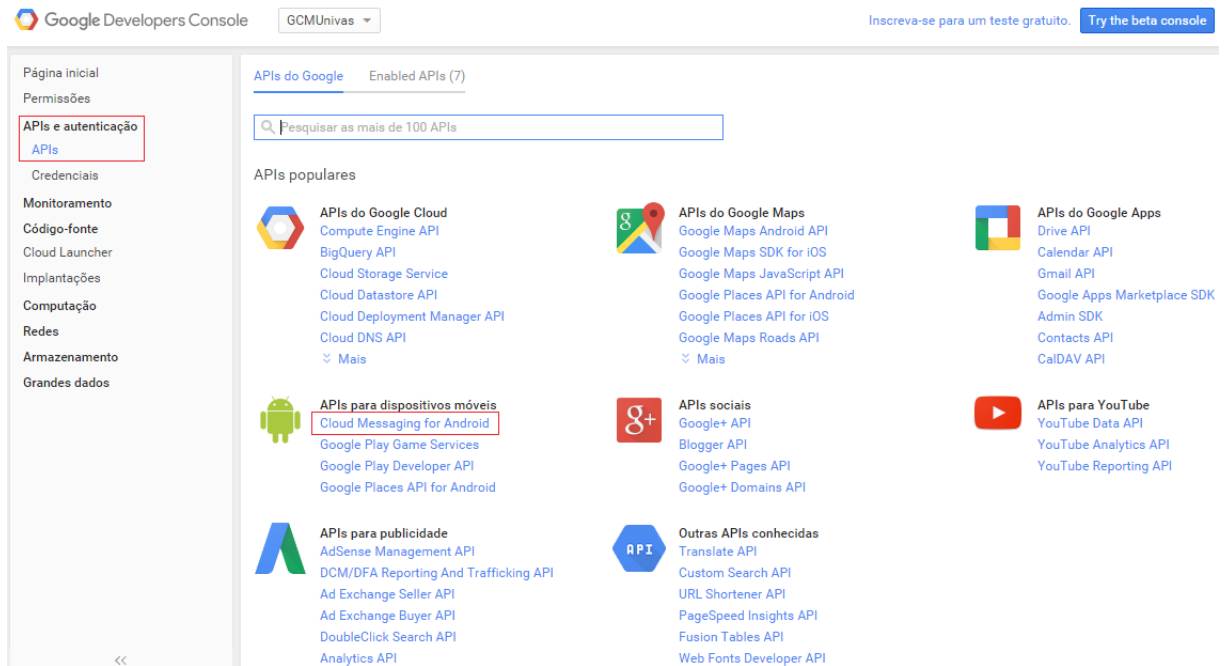


Figura 10 – Habilitando GCM para Android. **Fonte:**Elaborado pelos autores.

Ao selecionar *Cloud Messaging for Android*, foi apresentada a tela com a opção de ativar o GCM ao projeto, como mostra a Figura 11.

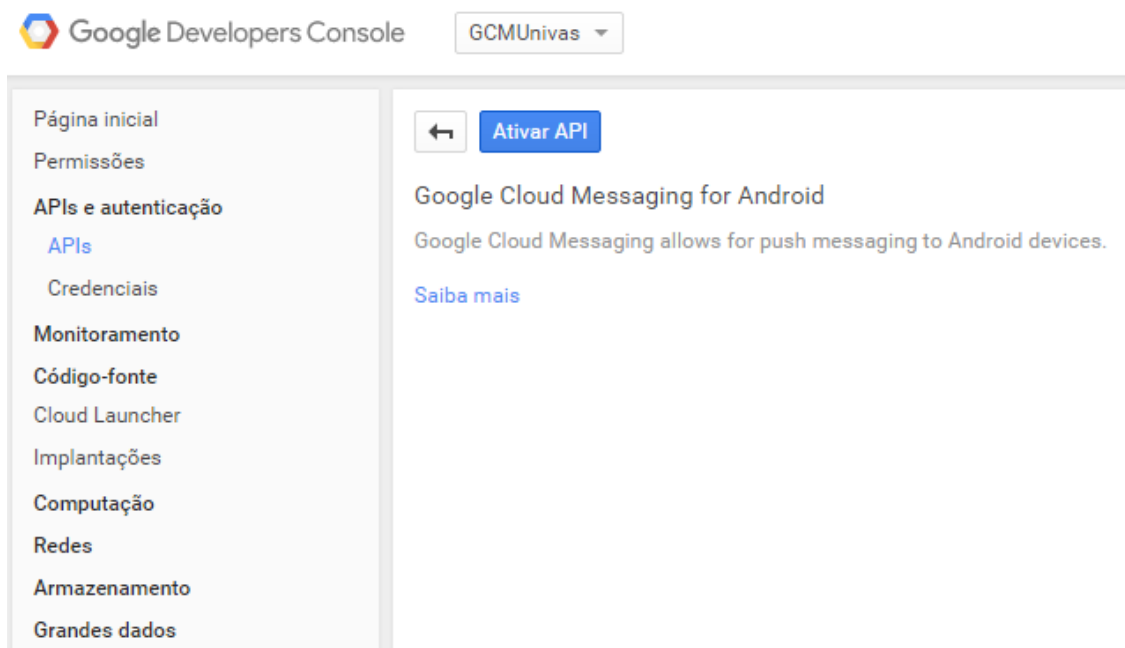


Figura 11 – Botão para ativar o GCM ao projeto. **Fonte:**Elaborado pelos autores.

Para concluir a configuração, foi preciso acessar a aba APIs e autenticação novamente, escolhendo a alternativa Credenciais, como mostra a Figura 12. Na página apresentada, foi selecionada a opção Chave de API, igualmente exibido na Figura 12.

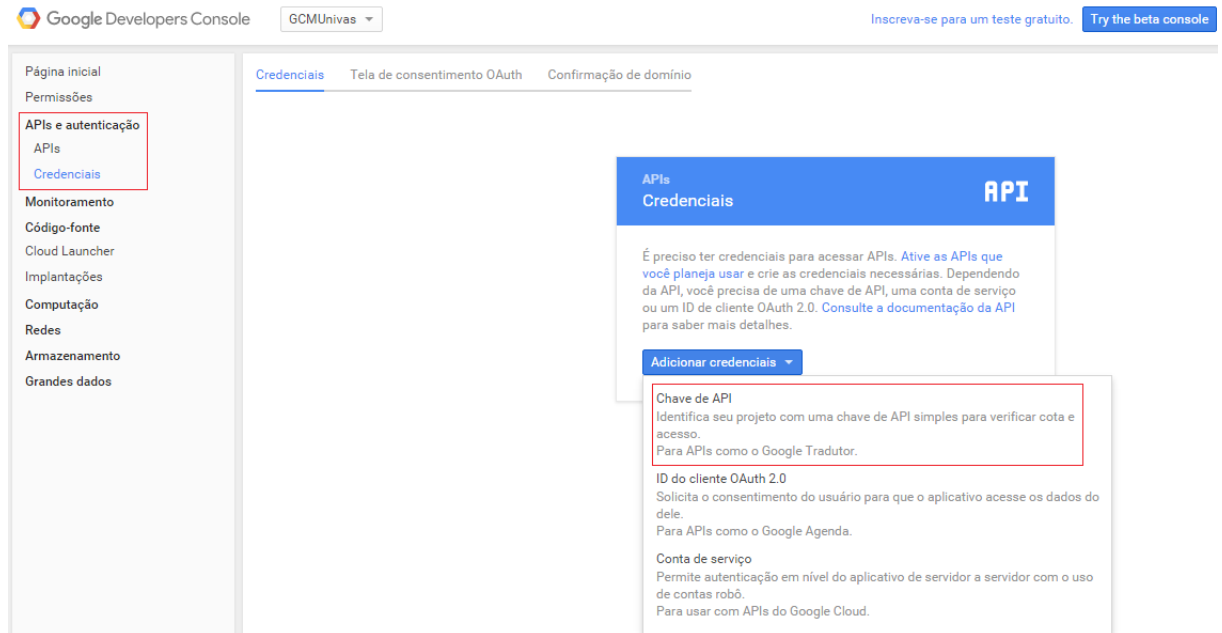


Figura 12 – Criando as credenciais. **Fonte:**Elaborado pelos autores.

Ao escolher Chave de API, foi exibida uma tela na qual se escolheu a opção Chave de Servidor, demonstrado na Figura 13.

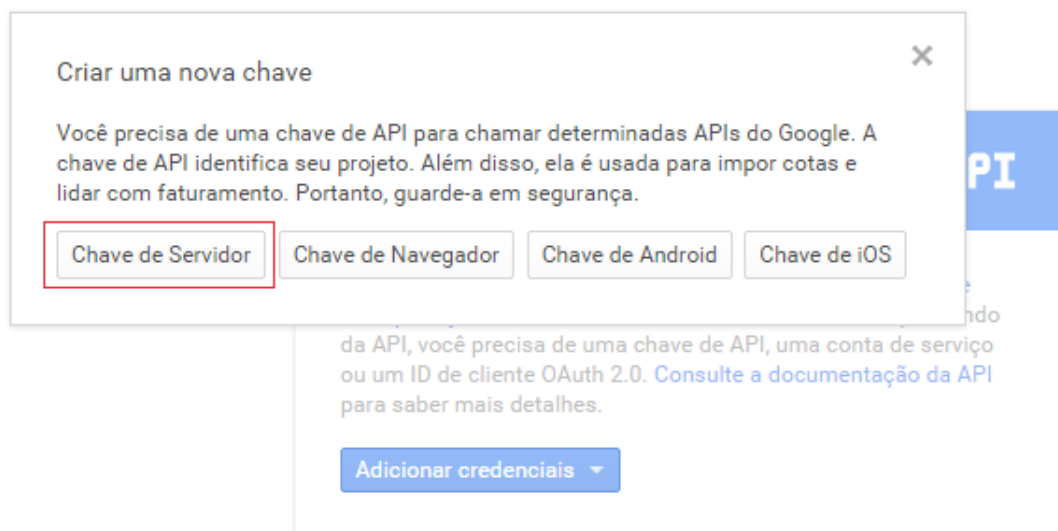
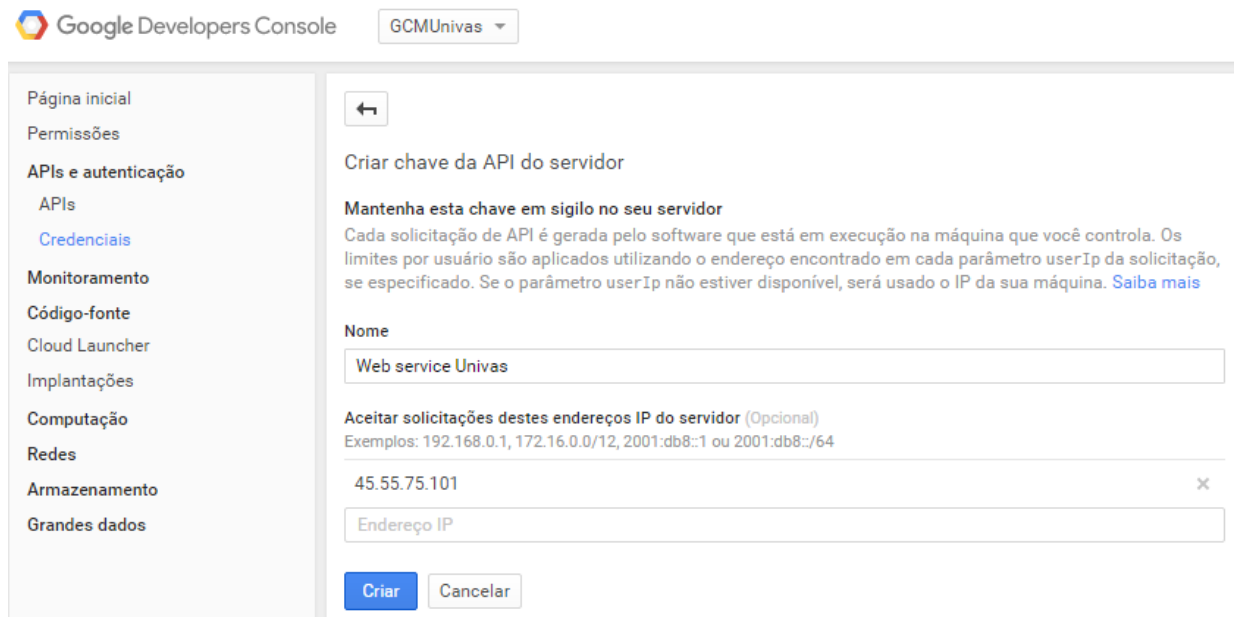


Figura 13 – Escolhendo a opção Chave de Servidor. **Fonte:**Elaborado pelos autores.

Ao decidir-se por Chave de Servidor, foi apresentada uma tela onde é criada a chave pública, também conhecida por *Sender Auth Token*. Esta identificação é transmitida no cabeçalho das mensagens enviadas do servidor ao GCM. Para que esse código fosse gerado

foi fundamental adicionar um nome e o IP do *web service*, como mostra a Figura 14. Ao clicar no botão criar, o Google apresentou a chave gerada, como ilustra a Figura 15.



Google Developers Console GCMUnivas

Página inicial
Permissões
APIs e autenticação
APIs
Credenciais
Monitoramento
Código-fonte
Cloud Launcher
Implantações
Computação
Redes
Armazenamento
Grandes dados

←

Criar chave da API do servidor

Mantenha esta chave em sigilo no seu servidor

Cada solicitação de API é gerada pelo software que está em execução na máquina que você controla. Os limites por usuário são aplicados utilizando o endereço encontrado em cada parâmetro `userIp` da solicitação, se especificado. Se o parâmetro `userIp` não estiver disponível, será usado o IP da sua máquina. [Saiba mais](#)

Nome

Web service Univas

Aceitar solicitações destes endereços IP do servidor (Opcional)

Exemplos: 192.168.0.1, 172.16.0.0/12, 2001:db8::1 ou 2001:db8::/64

45.55.75.101

Endereço IP

Criar Cancelar

Figura 14 – Inserindo dados do servidor. **Fonte:**Elaborado pelos autores.



Chave de API

Esta é sua chave de API

AIzaSyDBGsXMZSoMGNgDdVOh9B5WLP050o_9_SQ

OK

Figura 15 – Chave de API gerada. **Fonte:**Elaborado pelos autores.

3.4.3 Web service

Nesta seção serão descritos os procedimentos realizados para o desenvolvimento do *web service* responsável por prover os dados necessários ao aplicativo. Além disso serão descritas as configurações necessárias para a montagem do ambiente de desenvolvimento.

3.4.3.1 Montagem do Ambiente de Desenvolvimento

No que diz respeito à construção do *web service*, 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. Este *Servlet Container* foi instalado, pois implementa a API da especificação *Servlets* 3.0 do Java. Este passo foi necessário pelo fato de que o *framework* Jersey usa *servlets* para disponibilizar serviços REST. Além disso o Apache Tomcat foi escolhido, para que o *web service* pudesse fornecer os serviços necessários para o consumo do aplicativo, na arquitetura REST, que sugere o uso do protocolo HTTP¹ 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, de seu site oficial. 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 a IDE² 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 preferido pelo desenvolvedor.

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.

¹ HTTP - Hypertext Transfer Protocol

² IDE - Integrated Development Environment

3.4.3.2 Desenvolvimento

Com o ambiente de desenvolvimento pronto, começou de fato o desenvolvimento. Primeiramente foi necessário criar o banco de dados no SGDB. Este por sua vez foi criado com a ajuda do PgAdmin que é um software gráfico para administração do SGDB, e que fornece uma interface visual de apoio para o PostgreSQL. Para criar o banco era necessário já estar com o PgAdmin aberto e conectado a um servidor de banco de dados que neste caso era um servidor local como pode ser visto na Figura 16.

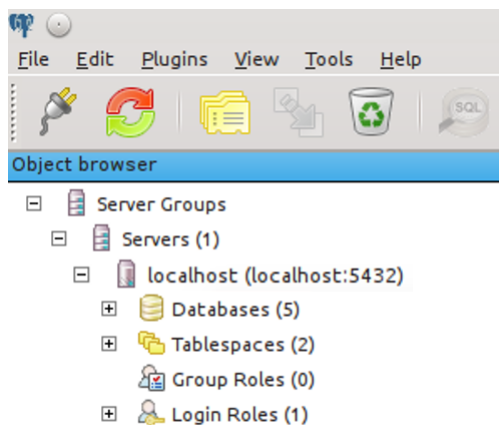


Figura 16 – Servidor de banco de dados local no PgAdmin. **Fonte:**Elaborado pelos autores.

Para a efetiva criação do banco de dados era necessário clicar com o botão direito do *mouse*, sobre a opção **Databases -> New Database...** no PgAdmin, apresentada na Figura 17.

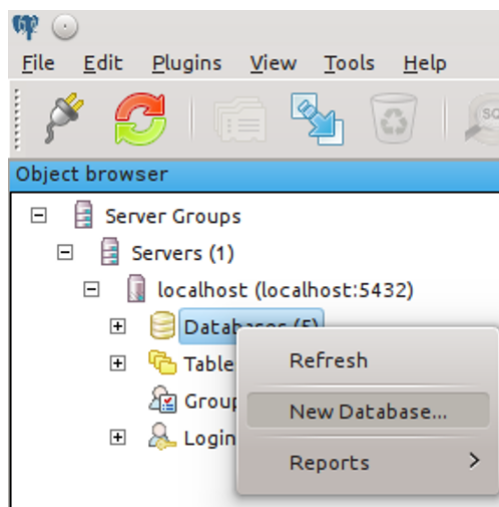


Figura 17 – Opção *New Database...* **Fonte:**Elaborado pelos autores.

Em seguida foi necessário preencher os dados da janela apresentada, como está demonstrado na Figura 18.

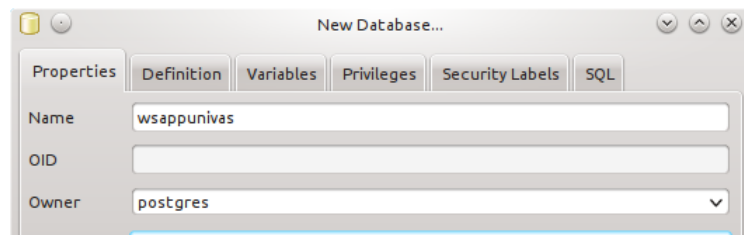


Figura 18 – Tela *New Database...*... **Fonte:**Elaborado pelos autores.

Como pode ser visto, foram preenchidos os campos **Name** e **Owner**. O campo **Name** se refere ao nome do banco de dados que foi definido com *wsappunivas*, e o campo **Owner**, o responsável pelo banco de dados que, para este caso, foi usuário padrão do SGDB, que é o *postgres*. Além destas configurações mais nenhuma opção foi necessária. O banco de dados foi criado, porém sua estrutura não está 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 e de acordo com a evolução do projeto. Isto permitirá mudanças na estrutura do banco de dados e suas tabelas, e até mesmo eventuais correções no decorrer do desenvolvimento da aplicação.

Em seguida foi criado um projeto do tipo *Dynamic Web Project* no Eclipse conforme Apêndice I. Antes de começar o desenvolvimento foi necessário ainda criar uma pasta a qual foi a responsável por conter todos os arquivos *.jar* das bibliotecas que foram usadas para o desenvolvimento do *web service*. Em seguida foram copiados todos os arquivos *.jar* da biblioteca Hibernate que eram necessários ao projeto, para dentro desta pasta e também o *jar* do *driver* JDBC do PostgreSQL, que seria responsável por fazer a comunicação entre o banco de dados e a aplicação.

Além disso era necessário mais uma configuração para que as bibliotecas pudessem ser reconhecidas como parte do projeto. Era necessário fazer a inclusão destas bibliotecas para dentro do *build path* do projeto. Para isso foi necessário selecionar todos os arquivos *.jar* que estavam dentro da pasta *libs*, clicar com o botão direito do mouse sobre eles e escolher a opção **Build Path -> Add to Build Path**, que pode ser visto na Figura 19.

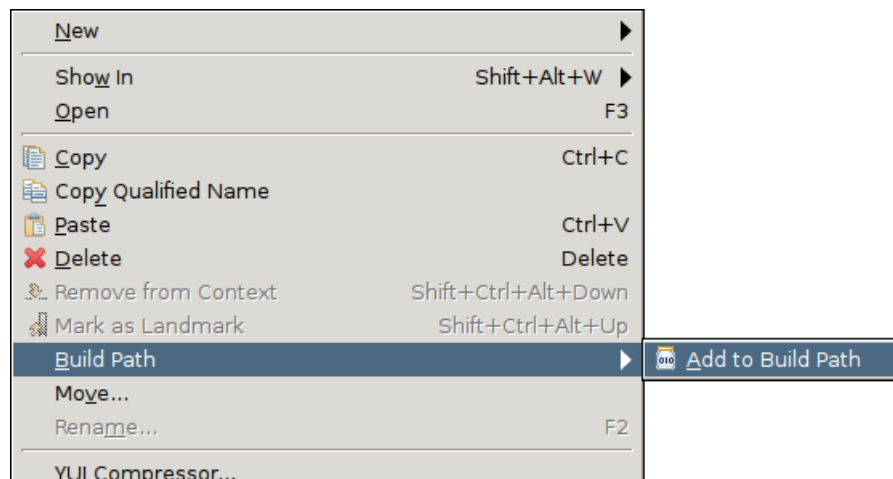


Figura 19 – Opção *Add to Build Path* no Eclipse. **Fonte:**Elaborado pelos autores.

Com o projeto devidamente configurado, começou-se de fato a desenvolver a camada de persistência da aplicação. Para este propósito, primeiramente, foi criado um pacote, em que ficaram contidas as classes que representam as entidades do ORM. Este 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.

Com este pacote criado, já era possível criar as classes do ORM. Foi criada primeiramente a classe `Student.java`. Esta classe foi definida para representar as informações referente aos alunos. O código fonte desta classe pode ser visto na Figura 20.

```

1  package br.edu.univas.restapiappunivas.model;
2  /**
3   *imports omitidos
4   */
5
6  @Entity
7  @Table(name = "student")
8  public class Student {
9
10     @Id
11     @SequenceGenerator(name = "id_student", sequenceName = "
        seq_id_student",
12         allocationSize = 1)
13     @GeneratedValue(generator = "id_student", strategy =
        GenerationType.IDENTITY)
14     @Column(name = "id_student", nullable = false)
15     private Long idStudent;
16
17     @Column(name = "id_external", nullable = false)
18     private Long idDatabaseExternal;
19
20     @Column(length = 100, nullable = false)
21     private String name;
22
23     @Column(length = 100, nullable = false)
24     private String email;
25
26     @OneToMany(mappedBy="student", fetch = FetchType.EAGER)
27     private List<Event> events;
28
29     @OneToOne(optional = false, fetch = FetchType.LAZY)
30     @JoinColumn(name = "id_user")
31     private User user;
32
33     /**
34     * Omitidos todos Getters e Setters
35     */
36
37     @Override
38     public int hashCode() {
39         /**
40         * Omitido
41         */
42     }
43
44     @Override
45     public boolean equals(Object obj) {
46         /**
47         * Omitido
48         */
49     }
50 }
51

```

Figura 20 – Classe Student.java. **Fonte:**Elaborado pelos autores.

É válido lembrar que esta classe possui anotações para que possa ser reconhecida como uma entidade do JPA, e assim persistida no banco de dados quando necessário. Além disso, estas anotações possuem outras finalidades específicas. A seguir estão listadas todas

as anotações que foram usadas na classe `Student.java` e nas outras classes que fazem parte do mapeamento objeto relacional da aplicação e definidas suas funcionalidades.

- `@Entity`: esta anotação foi necessária para que esta classe pudesse ser reconhecida como uma entidade do JPA e assim persistida no banco de dados;
- `@Table`: anotação que possui algumas configurações relativas a uma tabela no banco de dados, a qual esta entidade representa, no caso da classe mostrada anteriormente é configurado o nome da tabela;
- `@Id`: esta anotação fica sobre o atributo da entidades, o qual representa a chave primária no banco de dados;
- `@GeneratedValue`: esta anotação indica qual será a estratégia usada para incrementar a chave primária da tabela.
- `@SequenceGenerator`: esta anotação define o mecanismo com que a chave primária será incrementada.
- `@Column`: define algumas propriedades do campo da tabela do banco de dados, do atributo que ele anota representa. Estas configurações podem são:
 - `name`: mapeia o nome do campo;
 - `length`: determina o tamanho em caracteres que o campo aceitará;
 - `nullable`: define se o preenchimento do campo é obrigatório;
 - `unique`: este atributo define se o campo aceitará valores únicos;
- `@OneToMany`: representa um relacionamento um-para-muitos no banco de dados. Anotam coleções de outras entidades com as quais esta entidade possui relacionamento;
- `@ManyToOne`: representa um relacionamento muitos-para-um no banco de dados. Este é a contraparte da anotação um-para-muitos;
- `@OneToOne`: representa um relacionamento um-para-um no banco de dados.

Esta classe faz parte do mecanismo de persistência de dados e é simplesmente um POJO ou seja, um objeto simples que contém somente atributos privados e os métodos *getters* e *setters* que servem apenas para encapsular estes atributos, e não possui nenhuma

lógica de negócios. Além desta classe, foram criadas outras com os mesmos propósitos, que podem ser vistas no diagrama de classes que está apresentado na Figura 21.

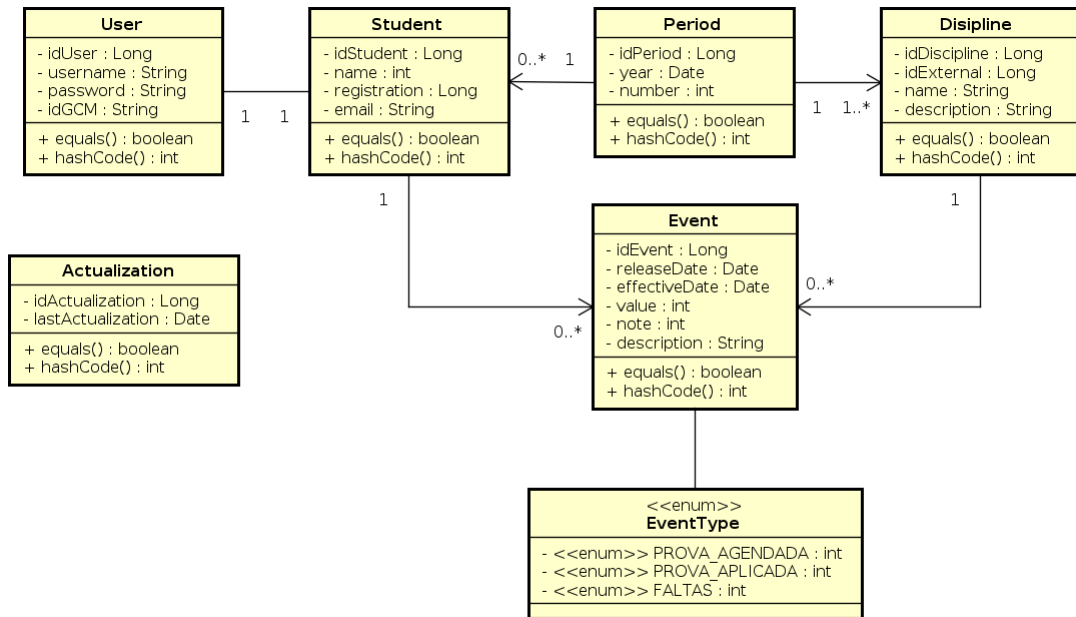


Figura 21 – Diagrama de classes do pacote model. **Fonte:**Elaborado pelos autores.

Estas classes tinham a mesma finalidade da anterior, porém com pequenas diferenças no que diz respeito aos atributos, métodos e anotações. Estas classes representam, de maneira individual, as tabelas no banco de dados. O modelo físico do banco de dados pode ser visto na Figura 22.

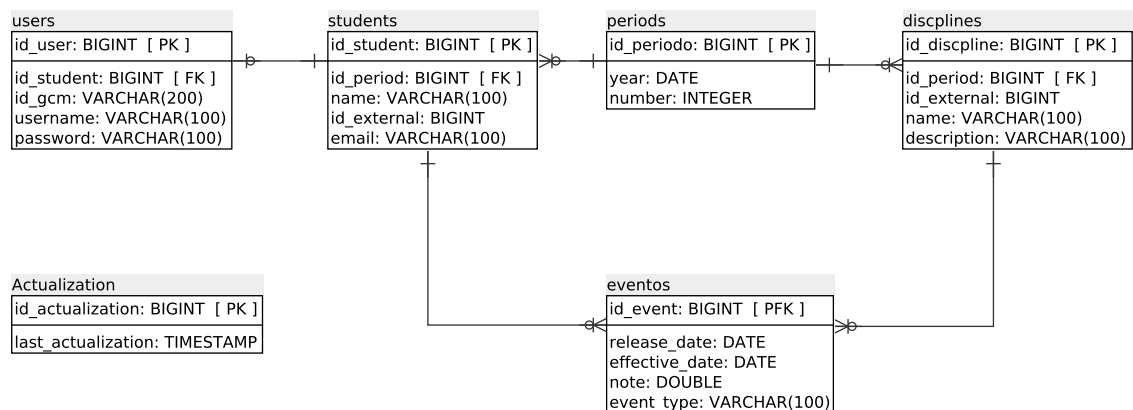


Figura 22 – Modelo físico do banco de dados. **Fonte:**Elaborado pelos autores.

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 vista que cada instância destas classes representa um registro no banco de dados. A própria IDE provê uma forma fácil para criar este métodos, bastando para isso clicar com o botão direito do mouse sobre o código da classe e escolher a opção **Source -> Generate hashCode() and equals()...** como pode ser visto na Figura 23.

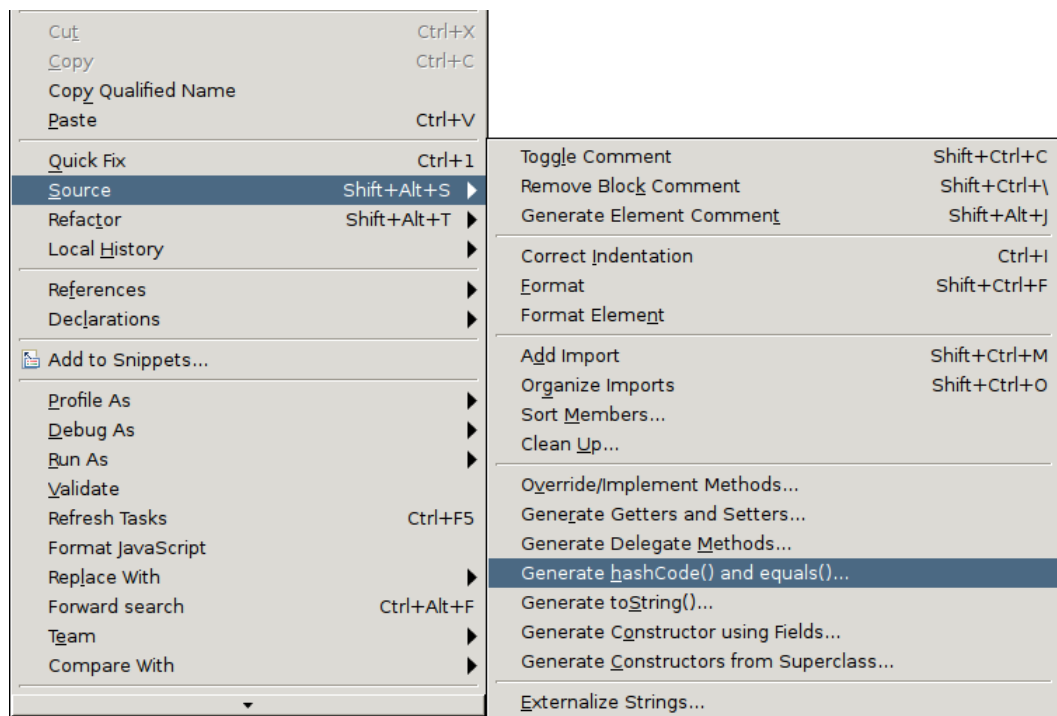


Figura 23 – Opção Generate hashCode() and equals()... **Fonte:**Elaborado pelos autores.

Os métodos `hashCode()` e `equals()` da classe `Student.java` foram implementados usando o atributo `idStudent` e podem ser vistos na Figura 24.

```

1  ...
2
3  @Override
4  public int hashCode() {
5      final int prime = 31;
6      int result = 1;
7      result = prime * result
8          + ((idStudent == null) ? 0 : idStudent.hashCode());
9      return result;
10 }
11
12 @Override
13 public boolean equals(Object obj) {
14     if (this == obj)
15         return true;
16     if (obj == null)
17         return false;
18     if (getClass() != obj.getClass())
19         return false;
20     Student other = (Student) obj;
21     if (idStudent == null) {
22         if (other.idStudent != null)
23             return false;
24     } else if (!idStudent.equals(other.idStudent))
25         return false;
26     return true;
27 }
28 ...

```

Figura 24 – Implementação os métodos hashCode() e equals(). **Fonte:**Elaborado pelos autores.

Além destas classes, foi necessário criar um tipo enumerado (ou enum), para definir quais seriam os tipos dos eventos, haja vista que estes teriam um numero limitado de possibilidades. Para esta enumeração foi definido o nome EventType. Os tipos de eventos definidos foram três:

- PROVA_AGENDADA: que define um evento como agendamento de uma atividade avaliativa;
- PROVA_APLICADA: que define um evento como a efetiva realização de uma atividade avaliativa;
- FALTAS: representa o lançamento de faltas;

A implementação da enumeração pode ser vista na Figura 25.

```
1
2 package br.edu.univas.restapiappunivas.model;
3
4 public enum EventType {
5     PROVA_AGENDADA, PROVA_APLICADA, FALTAS
6 }
```

Figura 25 – Implementação da enumeração EventType. **Fonte:**Elaborado pelos autores.

Após a criação das entidades da camada de persistência, foi necessário configurar o arquivo `persistence.xml`. Foi necessário criar a pasta META-INF dentro da pasta de códigos fontes do projeto que também é conhecida como *classpath*, com a finalidade de abrigar este arquivo. Em seguida foi criado o arquivo dentro desta pasta.

O arquivo `persistence.xml` 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. Este por sua vez recebeu o nome de `WsAppUnivas`.

Uma das configurações do `persistence.xml`, merece uma atenção especial. Trata-se da configuração `<property name="hibernate.hbm2ddl.auto" value="create"/>` que é a responsável por definir que o próprio Hibernate irá criar a estrutura do banco de dados (tabelas, sequences, etc.) através do mapeamento objeto relacional das classes. O arquivo `persistence.xml` está exposto na Figura 26.

```

<?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="omitido" />
      <property name="javax.persistence.jdbc.driver"
        value="org.postgresql.Driver" />
      <property name="hibernate.dialect"
        value="org.hibernate.dialect.PostgreSQLDialect" />
      <property name="hibernate.format_sql"
        value="true" />
      <property name="hibernate.temp.use_jdbc_metadata_defaults"
        value="false" />
      <property name="hibernate.show_sql"
        value="true" />
      <property name="hibernate.hbm2ddl.auto"
        value="create" />
    </properties>
  </persistence-unit>
</persistence>

```

Figura 26 – Arquivo persistence.xml. **Fonte:**Elaborado pelos autores.

Em seguida à confecção do persistence.xml foi criada a classe JpaUtil.java que está representada na Figura 27. Para isso foi necessário criar um pacote que seria responsável por armazená-la, para que a organização do projeto pudesse ser mantida. Tal pacote recebeu o nome de de br.edu.univas.restapiappunivas.util e poderia se necessário abrigar outras classes de utilidades do projeto assim como a classe JpaUtil.java.

A classe JpaUtil.java é responsável por criar uma EntityManagerFactory. Este por sua vez é uma fábrica de instâncias de EntityManager que é quem cria a *persistence unit* ou unidade de persistência. O *persistence unit* desta aplicação foi configurado através do arquivo persistence.xml. Este por sua vez tem a responsabilidade de prover um modo de comunicação entre a aplicação e o banco de dados. No entanto a classe JpaUtil.java cria uma única instância de EntityManagerFactory, que é responsável por disponibilizar e gerenciar as instâncias de EntityManager de acordo com a necessidade da aplicação.

```

1 package br.edu.univas.restapiappunivas.util;
2
3 /**
4  *Imports Omitidos
5  */
6
7 public class JpaUtil {
8     private static EntityManagerFactory factory;
9
10    static {
11        factory = Persistence.createEntityManagerFactory("WsAppUnivas")
12        ;
13    }
14
15    public static EntityManager getEntityManager() {
16        return factory.createEntityManager();
17    }
18
19    public static void close() {
20        factory.close();
21    }
22 }

```

Figura 27 – Classe JpaUtil.java. **Fonte:**Elaborado pelos autores.

Depois de finalizada a criação da camada de persistência do projeto foi necessário uma configuração adicional. Percebeu-se que, além das bibliotecas já usadas no projeto, seriam necessárias mais algumas bibliotecas tais como Jersey JSON e Jersey *Servlet*, para que se pudesse chegar ao resultado final esperado. Por esse motivo tornava-se inviável ficar controlando as bibliotecas de maneira manual no projeto. Foi necessário, então fazer a conversão do projeto para um projeto que fosse controlado pelo Maven. Para tanto foi necessário clicar com o botão direito do mouse sobre o projeto e navegar até a opção **Configure -> Convert to Maven Project**, como pode ser visto na Figura 28

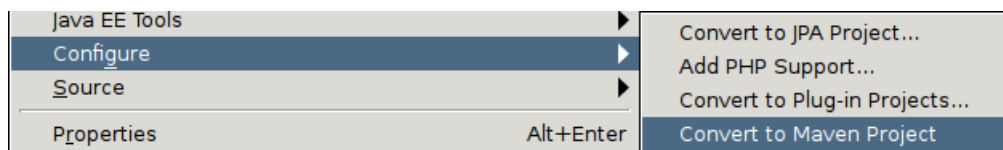


Figura 28 – Opção para conversão projeto web par projeto Maven. **Fonte:**Elaborado pelos autores.

Na janela que foi apresentada na sequência, foi necessário preencher os campos **Group id**, **Artifact id**, **Version** e **Packaging**, da seguinte forma:

- **Group id:** representa id do grupo ao qual pertence o projeto, que recebeu o nome de "br.edu.univas".
- **Artifact id:** foi preenchido com o nome do próprio projeto já criado anteriormente pois, este seria o nome do artefato final do gerado pelo Maven.

- **Version:** versão na qual está o projeto. Neste caso manteve-se o que já veio por padrão.
- **Packaging:** a forma como o projeto seria empacotado após a compilação do mesmo. Foi escolhido ao empacotamento do tipo `war` por se tratar de um projeto Java para *web*.

Com a conclusão da conversão do projeto foi gerado o arquivo `pom.xml`. Este arquivo provê as informações necessárias para que o Maven faça a gerência do projeto. Este arquivo possui, além das informações apresentadas anteriormente (**Group id**, **Artifact id**, **Version** e **Packaging**), algumas informações a respeito da compilação do projeto e um recurso importante, que é a gerência das dependências do projeto. Inicialmente foi incluída somente as dependências referentes ao *driver* JDBC do PostgreSQL e a biblioteca Hibernate como pode ser visto na Figura 29.

```
...
<!--driver JDBC postgresql -->
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <version>9.2-1003-jdbc4</version>
</dependency>
<!--hibernate implementation -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>4.3.11.Final</version>
</dependency>
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-entitymanager</artifactId>
  <version>4.3.11.Final</version>
</dependency>
...
```

Figura 29 – Arquivo `pom.xml`. **Fonte:**Elaborado pelos autores.

Em seguida a essa configuração foi necessário apagar a pasta `libs` do projeto, pois esta já não era mais necessária, haja vista que as bibliotecas usadas no projeto já estavam sendo gerenciadas pelo Maven. A partir daí foi necessário criar o código de disponibilização dos serviços REST. Para isto foram criados três novos pacotes para abrigar as classes que formaram a parte de serviços do *web service*, são eles:

- `br.edu.univas.restapiappunivas.controllers`: pacote responsável por conter as classes responsáveis por realizar as consultas na camada de persistência do

projeto e prover o resultados para publicação nos serviços.

- `br.edu.univas.restapiappunivas.entities`: pacote responsável por agrupar as classes que serviriam para abrigar os dados retornados nas consultas dos *controllers* e servir como retorno aos métodos das classes de serviços.
- `br.edu.univas.restapiappunivas.resources`: pacote responsável por agrupar as classes responsáveis por prover os serviços REST.

Foi necessário criar algumas classes que ficaram responsáveis por fazer as consultas no banco de dados usando o Hibernate. Estas classes tinham alguns métodos que com a responsabilidade de fazer a consulta e retornar os dados que seriam usados pelas classes que continham os serviços do *web service*. Uma destas classes é a `StudentEventsCtrl.java`, que pode ser vista na Figura 30.

```

1 package br.edu.univas.restapiappunivas.controller;
2 /**
3  * Imports omitidos
4  */
5 public class StudentEventCtrl {
6
7     public StudentEvent getEventsByRegistration(Long idStudent) {
8
9         EntityManager em = JpaUtil.getEntityManager();
10
11         String jpql = "select distinct e.idEvent, e.date, e.value, e.
12             note,";
13         jpql += " e.description, e.eventType, d.idDiscipline, d.
14             idDatabaseExternal from Discipline d ";
15         jpql += " right outer join d.events e right outer join e.
16             student s where s.idDatabaseExternal = :id ";
17
18         try {
19             Query query = em.createQuery(jpql);
20             query.setParameter("id", idStudent);
21
22             List<Object[]> resultSet = query.getResultList();
23             List<StudentEvent> studentEvents = new ArrayList<StudentEvent>
24                 <>();
25
26             for (Object[] obj : resultSet) {
27                 StudentEvent se = new StudentEvent();
28                 se.setIdEvent((Long) obj[0]);
29                 se.setDate((Date) obj[1]);
30                 se.setValue((int) obj[2]);
31                 se.setNote((int) obj[3]);
32                 se.setDescription((String) obj[4]);
33                 se.seteventType((TipoEvento) obj[5]);
34                 se.setIdDiscipline((Long) obj[6]);
35                 se.setidDatabaseExternal((Long) obj[7]);
36
37                 studentEvents.add(se);
38             }
39
40             StudentEvents students = new StudentEvents();
41             students.setEvents(studentEvents);
42
43             return students;
44         } catch (Exception e) {
45             e.printStackTrace();
46
47             throw new WebApplicationException(Status.NOT_FOUND);
48         } finally {
49             em.close();
50         }
51 }

```

Figura 30 – Classe StudentEventsCtrl.java. **Fonte:**Elaborado pelos autores.

Esta classe é composta unicamente pelo método `getEventsByRegistration()` o qual é responsável por fazer a busca de todos os eventos de um determinado aluno pela

sua matrícula. Ainda na Figura 30 são usadas instâncias das classes `StudentEvent` e `StudentEvents`. A classe `StudentEvent` foi criada apenas para servir como um simples POJO que contém o retorno da consulta JPQL³. O código desta classe pode ser visto na Figura 31.

```
1 package br.edu.univas.restapiappunivas.entities;
2
3 /**
4  * Imports Omitidos
5  */
6 public class StudentEvent {
7
8     private Long idEvent;
9
10    private Date effectiveDate;
11
12    private int value;
13
14    private int note;
15
16    private String description;
17
18    private TipoEvento eventType;
19
20    private Long idDiscipline;
21
22    private Long idExternalDiscipline;
23
24    private Long idStudent;
25
26    /**
27     * Metodos getters e setter Omitidos
28     */
29
30 }
```

Figura 31 – Classe `StudentEvent.java`. **Fonte:**Elaborado pelos autores.

Por outro lado a classe `StudentEvents` recebe uma coleção de `StudentEvent` para que a mesma pudesse ser convertida corretamente para JSON, nas classes que disponibilizam o serviço. O código fonte desta classe pode ser visto na Figura 32.

³ JPQL - *Java Persistence Query Language*

```

1 package br.edu.univas.restapiappunivas.entities;
2 /**
3  *Imports omitidos
4  */
5
6 public class StudentEvents {
7
8     private List<StudentEvent> events = new ArrayList<StudentEvent>()
9         ;
10
11     public List<StudentEvent> getEvents() {
12         return events;
13     }
14
15     public void setEvents(List<StudentEvent> events) {
16         this.events = events;
17     }
18 }

```

Figura 32 – Classe StudentEvents.java. **Fonte:**Elaborado pelos autores.

Estas duas classes mostradas nas Figuras 31 e 32 foram criadas dentro do pacote `br.edu.univas.restapiappunivas.entities`. Além delas foi criada também a classe `StudentDisciplines` no mesmo pacote. Esta classe foi usada com o mesmo propósito da `StudentEvents`, que é encapsular um resultado e posterior conversão para JSON. O código fonte desta classe pode ser visto na Figura 33.

```

1 package br.edu.univas.restapiappunivas.entities;
2 /**
3  *Imports omitidos
4  */
5
6 public class StudentDisciplines {
7
8     private List<Discipline> disciplines = new ArrayList<Discipline>
9         >();
10
11     public List<Discipline> getDisciplines() {
12         return disciplines;
13     }
14
15     public void setDisciplines(List<Discipline> disciplines) {
16         this.disciplines = disciplines;
17     }
18 }

```

Figura 33 – Classe StudentDisciplines.java. **Fonte:**Elaborado pelos autores.

Ainda foi criada mais uma classe que tinha basicamente a mesma função da classe `StudentEventsCtrl.java` que era de fazer a consulta e retornar os dados que seriam usados pelas classes que iam tornar disponíveis os serviços do *web service*. Esta classe foi

chamada de `StudentDisciplinesCtrl.java` e seu código fonte pode ser visto na Figura 34.

```
1 package br.edu.univas.restapiappunivas.controller;
2 /**
3  *Imports omitidos
4  */
5 public class StudentDisciplinesCtrl {
6
7     public AlunoDisciplinas getDisciplineByRegistrationStudent(Long
        idStudent) {
8         EntityManager em = JpaUtil.getEntityManager();
9
10        String jpql = "select distinct d.idDiscipline, d.idExternal, d.
            name from Period p ";
11        jpql += "inner join p.disciplines d inner join p.studentss a
            where a.idExternal=:id";
12
13        try {
14            Query query = em.createQuery(jpql);
15            query.setParameter("id", idStudent);
16
17            List<Object[]> resultSet = query.getResultList();
18            List<Discipline> disciplines = new ArrayList<Discipline>();
19            for (Object[] obj : resultSet) {
20                Discipline d = new Discipline();
21                d.setIdDiscipline((Long) obj[0]);
22                d.setIdExternal((Long) obj[1]);
23                d.setName((String) obj[2]);
24                disciplines.add(d);
25            }
26            StudentDisciplines studentDisciplines = new
                StudentDisciplines();
27            studentDisciplines.setDisciplines(disciplines);
28            return studentDisciplines;
29        } catch (Exception e) {
30            e.printStackTrace();
31        } finally {
32            em.close();
33        }
34    }
35 }
36 }
```

Figura 34 – Classe `StudentDisciplinesCtrl.java`. **Fonte:**Elaborado pelos autores.

Em seguida à construção das classes que fazem a busca e retorno dos dados na camada de persistência, foram desenvolvidas as classes responsáveis pela disponibilização de serviços *RESTful*, fazendo uso do *framework* Jersey em sua versão de número 1.19. Para isso, primeiramente foi necessário incluir, no arquivo `pom.xml`, a dependência referente ao *framework* Jersey como pode ser visto na Figura 35.

```

...
<!--jersey framework -->
<dependency>
  <groupId>com.sun.jersey</groupId>
  <artifactId>jersey-server</artifactId>
  <version>1.19</version>
</dependency>
<dependency>
  <groupId>com.sun.jersey</groupId>
  <artifactId>jersey-servlet</artifactId>
  <version>1.19</version>
</dependency>
<dependency>
  <groupId>com.sun.jersey</groupId>
  <artifactId>jersey-json</artifactId>
  <version>1.19</version>
</dependency>
...

```

Figura 35 – Trecho do arquivo pom.xml com a configuração do Jersey. **Fonte:**Elaborado pelos autores.

Além desta configuração, foi necessário apontar no arquivo web.xml da aplicação, que pode ser encontrado dentro da pasta WebContent -> WEB-INF do projeto, qual seria o *servlet* responsável pelos serviços, que para esta aplicação foi a classe do *framework* Jersey com.sun.jersey.spi.container.servlet.ServletContainer. Mais uma configuração foi acrescentada no web.xml, que foi a definição dos pacotes onde se encontrariam as classes que fornecem os serviços REST da aplicação, que no caso foi o pacote br.edu.univas.restapiappunivas.resources. Estas duas configurações podem ser vistas na Figura 36.

```

...
<servlet-class>
  com.sun.jersey.spi.container.servlet.ServletContainer
</servlet-class>
<init-param>
  <param-name>jersey.config.server.provider.packages</param-name>
  <param-value>br.edu.univas.restapiappunivas.resources</param-value>
</init-param>
...

```

Figura 36 – Trecho do arquivo web.xml com a configuração do Jersey. **Fonte:**Elaborado pelos autores.

Com isso pode-se construir as classes que representam de fato os serviços do *web service*, que são as classes StudentResource.java e UserResource.java. Estas classes representam contextos REST desta aplicação que, segundo Saudate (2012, p.6), são "a raiz pela qual a aplicação está sendo fornecida para o cliente". Dentro de um contexto pode haver inúmeros recursos que, segundo Saudate (2012, p.5), podem ser definidos como

"conjuntos de dados que são trafegados pelo protocolo".

Inicialmente foi contruída a classe `StudentResouce.java`, que representa o contexto `students` desta aplicação, representada na Figura 37.

```
1 package br.edu.univas.restapiappunivas.resources;
2 /**
3  *Imports Omitidos
4  */
5 @Path("/students")
6 @Produces({ MediaType.APPLICATION_JSON })
7 @Consumes({ MediaType.APPLICATION_JSON })
8 public class StudentsResource {
9
10     @GET
11     @Path("events/{registration}")
12     public StudentEvents getEventsByStudentRegistration(@PathParam("registration") Long registration) {
13
14         StudentEventsCtrl ctrl = new StudentEventsCtrl();
15         return ctrl.getEventsByRegistration(registration);
16     }
17
18     @GET
19     @Path("disciplinas/{id}")
20     public StudentDisciplines getDisciplinesByStudentRegistration(@PathParam("id") Long id) {
21
22         StudentDisciplinesCtrl ctrl = new StudentDisciplinesCtrl();
23         return ctrl.getDisciplineByRegistration(idStudent);
24     }
25
26 }
```

Figura 37 – Classe `StudentResouce.java`. **Fonte:**Elaborado pelos autores.

A classe `StudentResouce.java`, representa o contexto `students` e é composto por alguns métodos. O primeiro deles é `getEventsByStudentRegistration()` que representa o recurso `events` e faz o retorno de todos os eventos de um determinado aluno através de sua matrícula. Já o método `getDisciplinesByStudentRegistration()` representa o recurso `disciplines`, que tem, por responsabilidade, retornar todas as disciplinas cursadas por um aluno em um período. A URL⁴ para acessar o recurso relativo ao método `getEventsByStudentRegistration()` usando REST tem o seguinte formato:

`http://localhost:8080/WebServiceAppUnivas/students/events/{$registration}`

Esta pode ser dividida da seguinte forma:

- `localhost:8080`: indica o endereço e porta do servidor que fornece este serviço;

⁴ URL - Uniform Resource Locator

- `WebServiceAppUnivas`: representa o contexto universal da aplicação;
- `students`: representa um contexto específico dentro da aplicação;
- `events`: representa um recurso;
- `registration`: representa um parâmetro passado para o serviço, que neste caso é o número da matrícula do aluno.

Além da classe `StudentResouce.java`, foi desenvolvida também a classe responsável pelo contexto `users`, que é `UserResources.java`. Esta por sua vez representa o serviço mais importante do *web service* que é responsável por receber os valores de `registration_id` do dispositivo gerado pelo GCM. Sem estes id's não seria possível enviar as mensagens aos servidores do GCM e consequentemente manter os aplicativos dos alunos constantemente atualizados. O código fonte desta classe pode ser visto na Figura 38.

```

1 package br.edu.univas.restapiappunivas.resources;
2 /**
3  *Imports Omitidos
4  */
5 @Path("/users")
6 @Produces({ MediaType.APPLICATION_JSON })
7 @Consumes({ MediaType.APPLICATION_JSON })
8 public class UserResource {
9     @PUT
10    public Status receiveGoogleId(User user) {
11        UserCtrl ctrl = new UserCtrl();
12        try {
13            ctrl.receiveGCMId(user);
14        } catch (Exception e) {
15            e.printStackTrace();
16            throw new WebApplicationException(Status.
17                INTERNAL_SERVER_ERROR);
18        }
19        return Status.OK;
20    }
21 }

```

Figura 38 – Classe `UserResouce.java`. **Fonte:**Elaborado pelos autores.

A classe `UserResources.java` recebe um objeto do tipo `User.java` que tem em seu conteúdo o id gerado pelo GCM preenchido e usa um objeto da classe `UserCtrl.java` para buscar o usuário que é o dono do id recebido e em seguida persiste o mesmo no banco de dados para o posterior uso. A classe `UserCtrl.java` está apresentada na Figura 39.

```

1 package br.edu.univas.restapiappunivas.controller;
2 /**
3  *Imports Omitidos
4  */
5 public class UserCtrl {
6     public void receiveGCMId(Usuario user) {
7         EntityManager em = JpaUtil.getEntityManager();
8         try {
9             String jpql = "select u from User u where u.username=:id";
10            TypedQuery<User> query = em.createQuery(jpql, User.class)
11                .setParameter("id", user.getUsername());
12            Usuario userNew = query.getSingleResult();
13            userNew.setIdGCM(user.getIdGCM());
14            em.getTransaction().begin();
15            em.persist(userNew);
16            em.getTransaction().commit();
17        } catch (Exception e) {
18            e.printStackTrace();
19        } finally {
20            em.close();
21        }
22    }
23 }

```

Figura 39 – Classe UserCtrl.java. **Fonte:**Elaborado pelos autores.

Nas classes que foram mostradas anteriormente e que fazem parte da disponibilização dos serviços REST, é perceptível o uso de algumas anotações. Estas anotações são parte do *framework* Jersey e são, geralmente, a forma que este usa para identificar e disponibilizar o serviços. A seguir será feita uma lista com as principais anotações usadas nessa aplicação e suas funcionalidades.

- @Path: mapeia para qual URL⁵ um recurso ou contexto deve responder. Foi usado tanto em classes quanto em métodos;
- @PathParam: permite o recebimento de parâmetros através URL de chamada do serviço;
- @Produces: determina qual o tipo de conteúdo produzido pela classe ou método que ele anota, no momento da chamada do serviço. Aceita variados tipos sendo os principais XML e JSON;
- @Consumes: determina qual o tipo de conteúdo é aceito pela classe ou método que ele anota.
- @GET: Representa que método anotado deverá, responder somente ao método HTTP GET;

⁵ URL - *Uniform Resource Locator*

- @PUT: Representa que método anotado deverá, responder somente ao método HTTP PUT;

De acordo com o desenvolvimento dos serviços, percebeu-se que com certa frequência, ocorria um situação que se mostrava um tanto quanto estranha. Trata-se do seguinte caso: muitas vezes em que um serviço era invocado, todas as consultas eram realizadas pela camada de persistência da aplicação, porém os resultados não eram apresentados. Além disso o *servlet* responsável por este serviço retornava um cabeçalho com um erro HTTP 500, que se trata de um código genérico de erro e que representa a ocorrência de um erro interno da aplicação servidora. Isto acontecia, porém ao examinar os *log's* de execução da aplicação não era possível encontrar nem um tipo erro e nem mesmo algum tipo de *Exception* lançada pela aplicação.

Porém após algum tempo de pesquisa, descobriu-se que era necessário contruir uma classe responsável por disponibilizar, em tempo de execução, uma forma de converter corretamente os objetos usados na aplicação por um formato que a aplicação usava como resposta, no caso, o JSON. A classe construída foi `JerseyProvider.java`, que pode ser vista na Figura 40.

```
1 package br.edu.univas.restapiappunivas.resources;
2 /**
3  *Imports omitidos
4  */
5 @Provider
6 public class JerseyProvider implements ContextResolver<JAXBContext>
7 {
8     private JAXBContext context;
9
10    @SuppressWarnings("rawtypes")
11    private Class[] types = {StudentDisciplinas.class, StudentEventos
12        .class };
13
14    public AlunosProvider() throws Exception {
15        this.context = new JSONJAXBContext(JSONConfiguration.mapped()
16            .arrays("events","disciplines").build(), types);
17    }
18
19    @SuppressWarnings("rawtypes")
20    public JAXBContext getContext( Class objectType) {
21        for (Class type : types) {
22            if (type == objectType) {
23                return context;
24            }
25        }
26        return null;
27    }
28 }
```

Figura 40 – Classe `JerseyProvider.java`. **Fonte:**Elaborado pelos autores.

A principal função desta classe era a de ser uma provedora de um contexto para que se pudesse ocorrer a serialização de objetos e a conversão dos mesmos para JSON que é o formato usado no *web service*. E assim foi concluída a parte de fornecimento de serviços da aplicação.

Era necessário também que, a cada novo evento lançado no banco de dados do *web service*, este fosse enviado ao GCM para que este pudesse enviar uma notificação aos dispositivos dos alunos. Com a finalidade de sanar esta necessidade de forma que fosse mais agíl foram construídas três classes, que eram responsáveis por tratar as mensagens a serem enviadas e posteriormente enviá-las ao GCM. Estas classes ficaram contidas no pacote `br.edu.univas.restapiappunivas.gcm`.

A primeira classe a ser criada foi `ContentMessageGCM.java`. Esta classe tinha por finalidade estruturar o conteúdo da mensagem que seria entregue ao GCM. Dentre os conteúdos da mensagem estão os `registration_ids`, que são os ids dos dispositivos dos alunos, que foram registrados nos servidores do GCM e `data` que é o conteúdo da mensagem. O código fonte desta classe pode ser visto na Figura 41.

```

1 package br.edu.univas.restapiappunivas.gcm;
2 /**
3  * Imports omitidos
4  */
5 public class ContentMessageGCM implements Serializable {
6
7     private static final long serialVersionUID = 1L;
8     private List<String> registration_ids;
9     private Map<String, Event> data;
10
11     public void addRegId(String regId) {
12         if (registration_ids == null)
13             registration_ids = new LinkedList<String>();
14         registration_ids.add(regId);
15     }
16
17     public void createData(String title, Event event) {
18         if (data == null)
19             data = new HashMap<String, Event>();
20
21         data.put("event", event);
22     }
23
24     public List<String> getRegistration_ids() {
25         return registration_ids;
26     }
27
28     public void setRegistration_ids(List<String> registration_ids) {
29         this.registration_ids = registration_ids;
30     }
31
32     public Map<String, String> getData() {
33         return data;
34     }
35
36     public void setData(Map<String, String> data) {
37         this.data = data;
38     }
39 }

```

Figura 41 – Classe ContentMessageGCM. java. **Fonte:**Elaborado pelos autores.

Além disso foi criada a classe estática PostToGCM. java, a qual é responsável por converter o conteúdo da mensagem para o formato aceito pelo GCM e enviá-lo através do método POST do HTTP. Para que a mensagem pudesse ser convertida em JSON um objeto do tipo ObjectMapper foi usado, e para enviar a mensagem para o GCM foi usado um objeto do tipo HttpURLConnection, em conjunto com o ObjectMapper. Esta classe está apresentada na Figura 42.

```

1 package br.edu.univas.restapiappunivas.gcm;
2 /**
3  *Imports omitidos
4  */
5
6 public class PostToGCM {
7
8     public static void post(String apiKey, ConteudoMensagemGCM
        content) {
9
10        try {
11
12            URL url = new URL("https://android.googleapis.com/gcm/send");
13
14            HttpURLConnection conn =
15                (HttpURLConnection) url.openConnection();
16
17            conn.setRequestMethod("POST");
18
19            conn.setRequestProperty("Content-Type", "application/json");
20            conn.setRequestProperty("Authorization", "key=" + apiKey);
21
22            conn.setDoOutput(true);
23
24            ObjectMapper mapper = new ObjectMapper();
25
26            DataOutputStream wr = new DataOutputStream(conn.
                getOutputStream());
27
28            mapper.writeValue(wr, content);
29
30            wr.flush();
31
32            wr.close();
33
34            int responseCode = conn.getResponseCode();
35            System.out.println("\nSending 'POST' request to URL : " + url
                );
36            System.out.println("Response Code : " + responseCode);
37
38            BufferedReader in = new BufferedReader(new InputStreamReader(
39                conn.getInputStream()));
40            String inputLine;
41            StringBuffer response = new StringBuffer();
42
43            while ((inputLine = in.readLine()) != null) {
44                response.append(inputLine);
45            }
46            in.close();
47
48            System.out.println(response.toString());
49
50        } catch (MalformedURLException e) {
51            e.printStackTrace();
52        } catch (IOException e) {
53            e.printStackTrace();
54        }
55    }
56 }

```

Figura 42 – Classe PostToGCM. java. **Fonte:**Elaborado pelos autores.

Além disso esta classe coleta a resposta retornada pelo GCM e lança este retorno nos *log's* da aplicação e por fim, para que estas duas classes pudessem trabalhar conjuntamente, foi necessário criar a classe `SendMessageGCM.java` que foi responsável por receber o conteúdo a ser enviado e os ids dos dispositivos aos quais seriam enviadas estas mensagens, e invocar o método `post()` da classe `PostToGCM.java`. Foi nessa classe também que estava contida a chave de autorização de envio de mensagens ao GCM, que foi atribuída ao atributo `apiKey` e que foi obtida através da criação de um projeto no site do Google *Developers*, já mostrado na seção 3.4.2. Esta classe pode ser vista na Figura 43.

```
1 package br.edu.univas.restapiappunivas.gcm;
2 /**
3  * Imports omitidos
4  */
5 public class SendMessageGCM {
6
7     private void sendMessage(List<Users> users) {
8
9         String apiKey = "AIzaSyC58w1R-ODzfpTzZx3e4WUKSXwE_VoDvqU";
10        ContentMessageGCM content = createContent(users);
11
12        if (!content.equals(null)) {
13            POST2GCM.post(apiKey, content);
14        } else {
15            System.out.println("Nenhum usuario registrado!");
16        }
17    }
18
19    private ContentMessageGCM createContent(List<users> users) {
20
21        ContentMessageGCM c = new ContentMessageGCM();
22
23        for (Users user : user) {
24            if (user.getIdGCM() == null) {
25                System.out.println("User - " + usuario.getUsername()
26                    + " id not have the registered gcm");
27            } else {
28                c.addRegId(usuario.getIdGCM());
29                System.out.println("Usuario - " + usuario.getUsername()
30                    + " added to the batch shipping!");
31            }
32        }
33
34        if (c.getRegistration_ids().size() == 0) {
35            return null;
36        } else {
37            c.createData("event", user.event);
38            return c;
39        }
40    }
41 }
```

Figura 43 – Classe `SendMessageGCM.java`. **Fonte:**Elaborado pelos autores.

Era necessário também que, de tempos em tempos, o próprio *web service* fizesse uma varredura, buscando por novos eventos lançados para os alunos, para que as classes responsáveis por enviar as mensagens ao GCM pudessem enviar tais notificações. Para isto primeiramente foi criada a classe `ListenerAtualizations.java` que pode ser vista na Figura 44.

```
1 /**
2  *Imports Omitidos
3  */
4
5 public class ListenerAtualizations implements
    ServletContextListener {
6
7     public void contextInitialized(ServletContextEvent arg0) {
8         ServletContext servletContext = arg0.getServletContext();
9
10        int delay = 1000;
11        Timer timer = new Timer();
12        timer.scheduleAtFixedRate(
13            new TimerTask() {
14                public void run() {
15                    /**
16                     * Tarefas a serem executadas
17                     */
18                }
19            }, delay, 5000);
20        servletContext.setAttribute("timer", timer);
21    }
22
23     public void contextDestroyed(ServletContextEvent arg0) {
24         ServletContext servletContext = arg0.getServletContext();
25         Timer timer = (Timer) servletContext.getAttribute("timer");
26         if (timer != null)
27             timer.cancel();
28         servletContext.removeAttribute("timer");
29     }
30 }
31
32 }
```

Figura 44 – Classe `ListenerAtualizations.java`. **Fonte:**Elaborado pelos autores.

Para que esta classe pudesse funcionar como um *timer* que dispare uma rotina, foram necessários dois detalhes que, foram de extrema importância. O primeiro era fazê-la implementar a *interface* `ServletContextListener`, para que esta pudesse funcionar como um *listener* que inicializasse essa classe assim que aplicação fosse ativada. O segundo foi acrescentar no arquivo `web.xml` a configuração referente ao *listener* que pode ser visto na Figura 45.

```

<listener>
  <listener-class>
    br.edu.univas.restapiappunivas.atualizations.ListenerAtualizations
  </listener-class>
</listener>

```

Figura 45 – Configuração do *Listener* no arquivo *web.xml*. **Fonte:**Elaborado pelos autores.

Além disso essa classe internamente acionava um *timer* que era atualizado a cada 5000 milissegundos. Dentro deste *timer* era criado um objeto do tipo *TimerTask* que era o criador de um nova *thread* para a execução das tarefas relativas à busca por novos eventos lançados e a notificações pelo GCM.

E ainda, 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 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. Para esta pesquisa, este módulo foi simulado.

Estes procedimentos foram os passos realizados com o propósito de se alcançar os resultados esperados para essa pesquisa.

3.4.4 Aplicativo

O primeiro passo realizado para a construção do aplicativo Android, foi a modelagem do software através dos diagramas de UML, que permitiram nortear o desenvolvimento.

A princípio projetou-se um diagrama para se ter uma visão geral do aplicativo. Nele pode-se ver a arquitetura da aplicação, bem como a comunicação com o GCM e o *web service*.

Quando o servidor precisa enviar uma informação para os dispositivos móveis, ele envia a mensagem ao GCM, que a transfere para aplicativo. Ao receber os dados, a classe *GcmBroadcastReceiverUnivas* executa a classe *GcmIntentServiceUnivas* que, por sua vez, irá notificar o usuário. Contudo antes de notificá-lo, ela envia as informações para a classe *HttpUtil*, que faz a conversão dos dados vindos em JSON para o formato da classe *EventTO*. Ao finalizar a leitura e conversão dos elementos, as informações são

enviadas para a classe DatabaseExecute que realiza a gravação dos dados no banco de dados.

Quando o usuário clica na notificação, é apresentada a ele uma *activity* a qual exibe as informações recebidas. Neste contexto, quando o evento recebido se tratar de uma nota será aberta a *activity* NotificationResultsActivity, no caso de ser uma falta é executada a *activity* NotificationFoulsActivity e caso for um evento de provas agendadas então é mostrada a *activity* NotificationAgendasActivity.

Para o estudante visualizar suas informações, ele acessará a classe MainActivity. Se desejar ver as suas notas, então é apresentada a *activity* ListResultsActivity, que utiliza a classe ListResultsAdapter para gerenciar as informações e apresentá-las. No entanto, se escolher a opção de faltas, é apresentada a *activity* ListFoulsActivity, que receberá as informações da classe ListFoulsAdapter e caso decida-se pela opção provas agendadas será apresenta a *activity* ListAgendasActivity, gerenciada pela classe ListAgendasAdapter.

A classe GcmControllerUnivas, tem por finalidade cadastrar o dispositivo no GCM e enviar a chave gerada para o *web service*. Na Figura 46, é apresentado o diagrama de arquitetura.

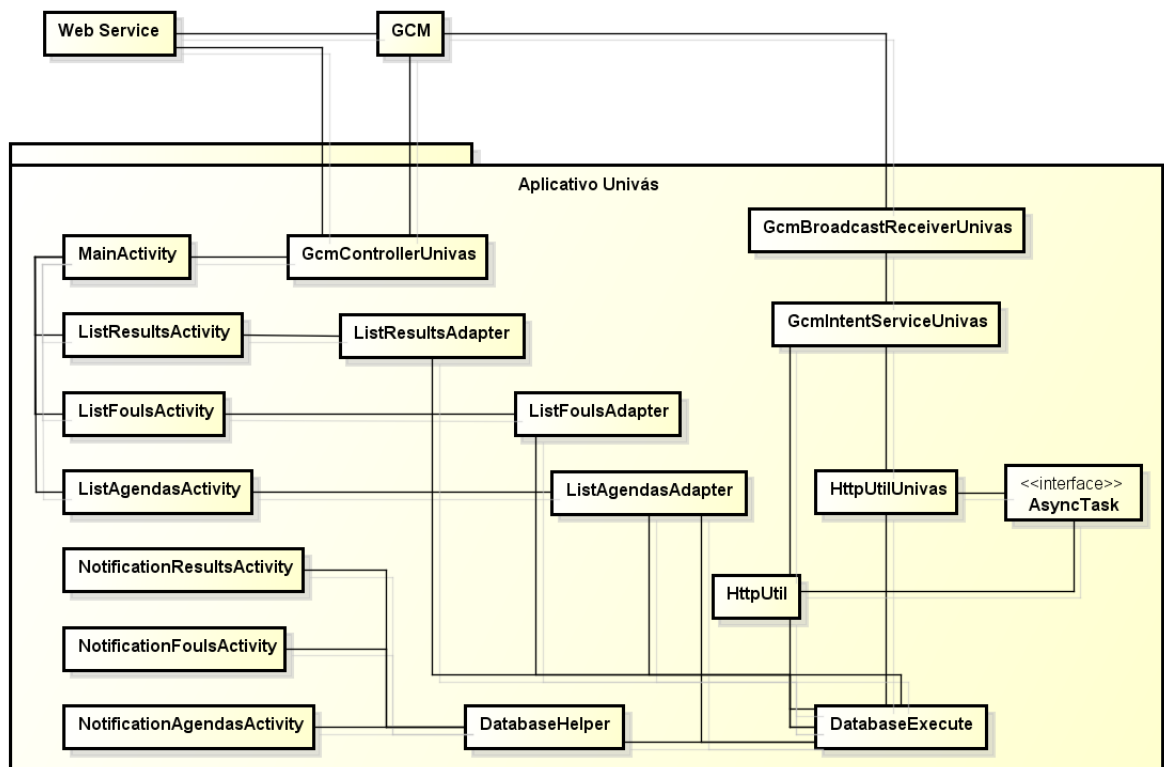


Figura 46 – Diagrama de arquitetura do aplicativo. **Fonte:**Elaborado pelos autores.

Posteriormente, foi desenvolvido o diagrama de caso de usos, com finalidade em ter uma visão das funcionalidades do software pelo lado do usuário. O utilitário permite ao aluno receber notificações quando for lançada uma nova nota, falta ou prova agendada e visualizar estas informações. Na Figura 47 é possível ver o diagrama de casos de uso.

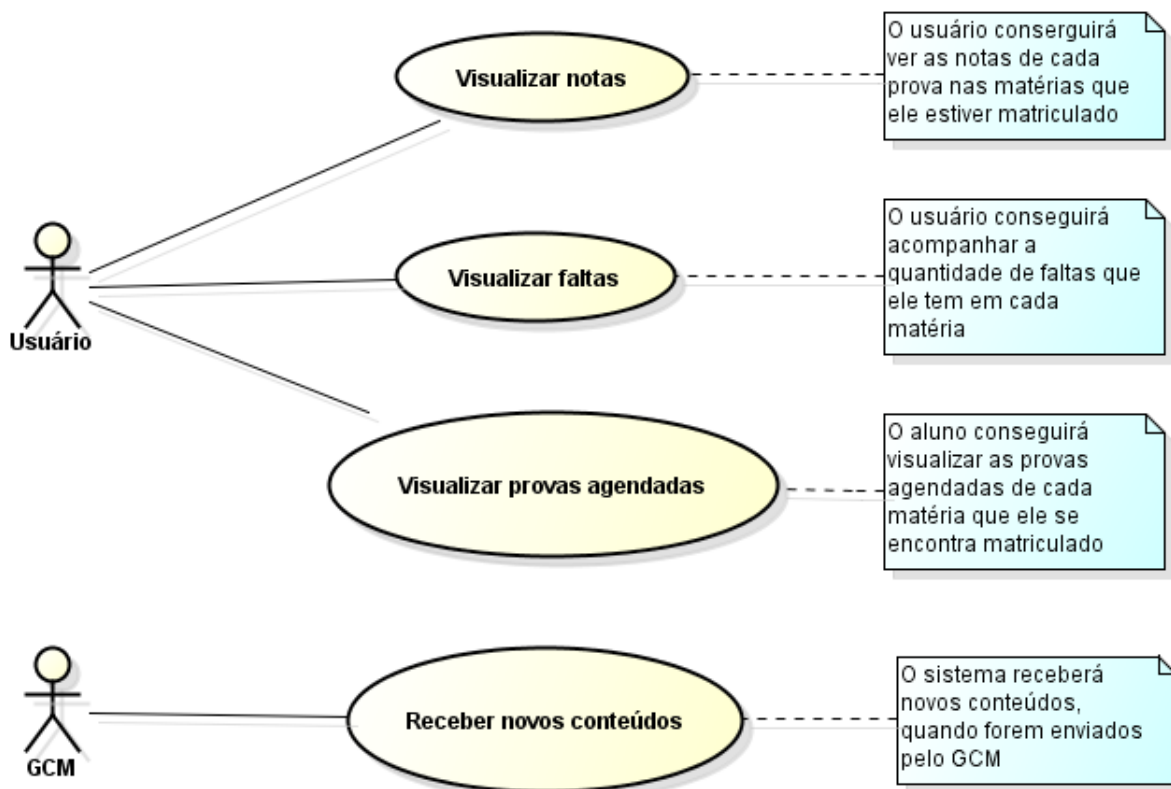


Figura 47 – Diagrama de casos de uso. **Fonte:**Elaborado pelos autores.

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* através do endereço <<https://developer.android.com/intl/pt-br/sdk/index.html>>.

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), que permite a execução do emulador Android mais rápido.

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, houve a necessidade de se criar um repositório

no controlador de versão Github, o qual pode ser acessado através do endereço <<https://github.com/diegodnunes12/AppTCC>> e compartilhado entre os participantes do projeto.

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.

O *layout* desta *activity* chama-se *relativeLayout*, que permite a um elemento se posicionar em relação a outro. Desta forma o *widget listView* do Android, utiliza a propriedade `android:layout_below="@+id/headerView"` para se posicionar abaixo do componente de id `headerView` e a propriedade `android:layout_above="@+id/footerView"` indicando que ela deve preceder o *widget* com id `footerView`. Na Figura 48, pode ser visto o código XML dos *widgets* desta tela.

```

<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:background="@android:color/white">
    <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:paddingTop="5dp"
        android:paddingBottom="5dp"
        android:paddingLeft="25dp"
        android:paddingRight="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:textColor="#228B22"
        android:text="Univas -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="#228B22"
        android:choiceMode="singleChoice"
        android:divider="@android:color/transparent"
        android:dividerHeight="1dp" />
</RelativeLayout>

```

Figura 48 – Código dos *widgets* do arquivo `fragment_navigation_drawer.xml`. **Fonte:**Elaborado pelos autores.

A classe `NavigationDrawerFragment` representa o painel de navegação. Nela se destaca o método `onCreateView()`, responsável por criar o *layout* de navegação. Na Figura 49, é mostrado o método `onCreateView()` informando ao sistema operacional o *layout* a ser carregado e adicionando a um *array* de *String* as alternativas de navegação que serão exibidos no *listView* da tela principal. Pode-se perceber também o método `onItemClick()`, que é executado no momento em que o usuário clica em algum item da lista.

```

1 @Override
2 public View onCreateView(
3     LayoutInflater inflater, ViewGroup container, Bundle
4         savedInstanceState) {
5     View view = inflater.inflate(
6         R.layout.fragment_navigation_drawer, container, false);
7
8     mDrawerListView = (ListView) view.findViewById(R.id.
9         navigationItems);
10    mDrawerListView.setOnItemClickListener(new AdapterView.
11        OnItemClickListener() {
12        @Override
13        public void onItemClick(AdapterView<?> parent, View view,
14            int position, long id) {
15            selectItem(position);
16        }
17    });
18    mDrawerListView.setAdapter(new ArrayAdapter<String>(
19        getActionBar().getThemedContext(),
20        android.R.layout.simple_list_item_activated_1,
21        android.R.id.text1,
22        new String[]{
23            getString(R.string.title_section1),
24            getString(R.string.title_section2),
25            getString(R.string.title_section3),
26            getString(R.string.title_section4),
27            getString(R.string.title_section5)
28        }));
29    mDrawerListView.setItemChecked(mCurrentSelectedPosition, true);
30    return view;
31 }
32
33 public boolean isDrawerOpen() {
34     return mDrawerLayout != null && mDrawerLayout.isDrawerOpen(
35         mFragmentManager.getView());
36 }

```

Figura 49 – Método onCreateView(). Fonte:Elaborado pelos autores.

Nesse caso, quando for selecionada alguma opção da tela principal será executado o método `selectItem()` apresentado na Figura 50, o qual é responsável por retornar a posição do *array* em que se encontra a opção selecionada pelo aluno.

```

1 private void selectItem(int position) {
2     mCurrentSelectedPosition = position;
3     if (mDrawerListView != null) {
4         mDrawerListView.setItemChecked(position, true);
5     }
6     if (mDrawerLayout != null) {
7         mDrawerLayout.closeDrawer(mFragmentManager.getView());
8     }
9     if (mCallbacks != null) {
10        mCallbacks.onNavigationDrawerItemSelected(position);
11    }
12 }

```

Figura 50 – método selectItem(). Fonte:Elaborado pelos autores.

O próximo passo foi criar o banco de dados do aplicativo para salvar as informações

recebidas do *web service*. Para que isso fosse possível, elaborou-se uma classe denominada *DatabaseHelper* que se estende da classe *SQLiteOpenHelper* do Android, com dois métodos, um chamado *onCreate()* que cria a estrutura do banco de dados e outro conhecido por *onUpgrade()*, usado se for necessário atualizar a estrutura do banco de dados.

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 1 (um), o sistema chamará o método *onCreate()*. Se for preciso atualizar a estrutura do banco, o atributo versão deve ser incrementado em 1 (um), de modo que ao executar o software o sistema operacional perceba a mudança, chamando o método *onUpgrade()*. Na Figura 51 é apresentado a classe *DatabaseHelper*.

```
1 public class DatabaseHelper extends SQLiteOpenHelper {
2
3     private static final String BANCO_DADOS = "univasDB";
4     private static int VERSAO = 1;
5
6     public DatabaseHelper(Context context) {
7         super(context, BANCO_DADOS, null, VERSAO);
8     }
9     @Override
10    public void onCreate(SQLiteDatabase db) {
11        db.execSQL("CREATE TABLE disciplinas (_id LONG PRIMARY KEY,
12                    nome TEXT);");
13
14        db.execSQL("CREATE TABLE eventos (_id LONG PRIMARY KEY,
15                    id_disciplina LONG, " +
16                    " tipo_evento TEXT, descricao_evento TEXT," +
17                    " data_evento TEXT, valor_evento INTEGER, nota
18                    INTEGER," +
19                    " FOREIGN KEY (id_disciplina) REFERENCES
20                    disciplinas (_id) );");
21    }
22    @Override
23    public void onUpgrade(SQLiteDatabase db, int i, int i2) {
24        //Nao ha atuaizacoes no momento
25    }
26 }
```

Figura 51 – Classe *DatabaseHelper*. **Fonte:**Elaborado pelos autores.

Em seguida foi criada a classe responsável por executar as consultas SQL, denominada *DatabaseExecute*. Nela foram inseridos os métodos responsáveis por inserir, alterar e buscar os dados dos alunos no banco de dados local do aplicativo. Na Figura 52, pode-se observar o método que possibilita a inserção dos eventos ocorridos. Esses eventos podem ser notas, faltas ou provas agendadas.

```

1 public void insertEvents(EventTO to){
2     SQLiteDatabase db = helper.getWritableDatabase();
3
4     ContentValues values = new ContentValues();
5     values.put("_id", to.getId());
6     values.put("id_disciplina", to.getId_discipline());
7     values.put("tipo_evento", to.getType_event());
8     values.put("descricao_evento", to.getDescription_event());
9     values.put("data_evento", to.getDate_event());
10    values.put("valor_evento", to.getValue_event());
11    values.put("nota", to.getResult());
12
13    long result = db.insert("eventos", null, values);
14
15    if(result != -1 ){
16        Log.d(TAG, " Evento salvo com sucesso!");
17    }else{
18        Log.d(TAG, " Erro ao salvar o Evento!");
19    }
20 }

```

Figura 52 – Método de inserção de eventos. **Fonte:**Elaborado pelos autores.

Este método recebe um objeto da classe EventTO com os elementos necessários para inserir o evento no banco de dados. Para que seja possível a inserção dos dados, Monteiro (2012), diz que é necessário recuperar a referência da classe SQLiteDatabase através do método `getWritableDatabase()`, logo após é instanciada a classe ContentValues, onde são informados os campos da tabela e os respectivos valores. 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()` que 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, que devolve um *Cursor* para navegar pelos resultados. Por fim, é composto um objeto do tipo EventTO e inserido na lista. Na Figura 53 é apresentado o método `getResults()`.

Foram inseridos mais dois métodos semelhantes ao `getResults()`, que são os métodos de `getFouls()` e `getAgendas()` para recuperar as faltas e provas agendadas respectivamente. O que os diferencia é a consulta SQL, já que no `getFouls()` foram buscados os dados onde o `tipo_evento = 'FALTAS'` e no `getAgendas()` onde o `tipo_evento = 'PROVA_AGENDADA'`.

```

1 public List<EventT0> getResults(){
2     List<EventT0> notasT0 = new ArrayList<>();
3
4     SQLiteDatabase db = helper.getReadableDatabase();
5     Cursor cursor =
6         db.rawQuery("SELECT _id, id_disciplina,
7                     descricao_evento, valor_evento, nota FROM" +
8                     " eventos WHERE tipo_evento = '
9                     PROVA_APLICADA'",
10                    null);
11     cursor.moveToFirst();
12
13     for(int i = 0; i<cursor.getCount();i++){
14         EventT0 nota = new EventT0();
15
16         nota.set_id(cursor.getLong(0));
17         nota.setId_discipline(cursor.getLong(1));
18         nota.setDescription_event(cursor.getString(2));
19         nota.setValue_event(cursor.getInt(3));
20         nota.setResult(cursor.getInt(4));
21
22         notasT0.add(nota);
23         cursor.moveToNext();
24     }
25     cursor.close();
26     return notasT0;
27 }

```

Figura 53 – Método getResults(). **Fonte:**Elaborado pelos autores.

A fim de estabelecer uma conexão entre o aplicativo e o *web service*, foi preciso conceder a permissão de acesso à internet no arquivo `AndroidManifest.xml` da seguinte forma: `<uses-permission android:name="android.permission.INTERNET"/>`.

Logo após, criou-se uma classe chamada de `HttpUtilUnivas` para ler informações recebidas do *web service*. Ela estende da classe `AsyncTask` que executa a consulta em paralelo com a *thread main*, evitando travar a aplicação enquanto recebe as informações vindas do *web service*. Estes dados estão em formato JSON e foi utilizada a biblioteca GSON para convertê-los para o formato da classe `EventT0`. Após a leitura, o objeto da classe `EventT0` é enviado para a classe `DatabaseExecute`, a fim de realizar a inserção os dados no banco. A Figura 54, mostra a classe `HttpUtilUnivas`.

```

1  try{
2      HttpClient httpClient = new DefaultHttpClient();
3      HttpGet request = new HttpGet();
4      request.setURI(new URI(urlEvents));
5
6      HttpResponse response = null;
7      try {
8          response = httpClient.execute(request);
9      } catch (IOException e) {
10         e.printStackTrace();
11     }
12     InputStream content = null;
13     try {
14         content = response.getEntity().getContent();
15     } catch (IOException e) {
16         e.printStackTrace();
17     }
18     Reader reader = new InputStreamReader(content);
19     Gson gson = new Gson();
20     returnEvents = gson.fromJson(reader, Events.class);
21
22     for (int i = 0; i < returnEvents.getEventos().size(); i++){
23         DatabaseExecute execute = new DatabaseExecute(helper);
24         EventTO to = new EventTO();
25         to.set_id((long) returnEvents.getEventos().get(i).
26             getId_evento());
27         to.setValue_event(returnEvents.getEventos().get(i).
28             getValor());
29         to.setDescription_event(returnEvents.getEventos().get(i).
30             getDescricao());
31         to.setId_discipline(returnEvents.getEventos().get(i).
32             getId_disciplina());
33         to.setDate_event(returnEvents.getEventos().get(i).
34             getData());
35         to.setResult(returnEvents.getEventos().get(i).getNota()
36             );
37         to.setType_event(returnEvents.getEventos().get(i).
38             getTipoEvento());
39
40         if(execute.existingEvent(to.get_id()) == false){
41             execute.insertEvents(to);
42         }else{
43             execute.updateEvent(to);
44         }
45     }
46     content.close();
47 } catch (URISyntaxException e) {
48     e.printStackTrace();
49 } catch (IOException e) {
50     e.printStackTrace();
51 }

```

Figura 54 – Código da classe HttpUtilUnivas que faz a leitura dos eventos. **Fonte:**Elaborado pelos autores.

Para usufruir da biblioteca GSON, foi fundamental adicioná-la como uma dependência do projeto. Para isso, foi preciso ir ao Menu do Android Studio, clicando em **File** e depois em **Project Structure**. Com a janela da estrutura do projeto aberta, foi selecionada a aba **Dependencies** e depois foi escolhido o ícone de mais (+) para adicionar novas

dependências, conforme mostra a Figura 55.

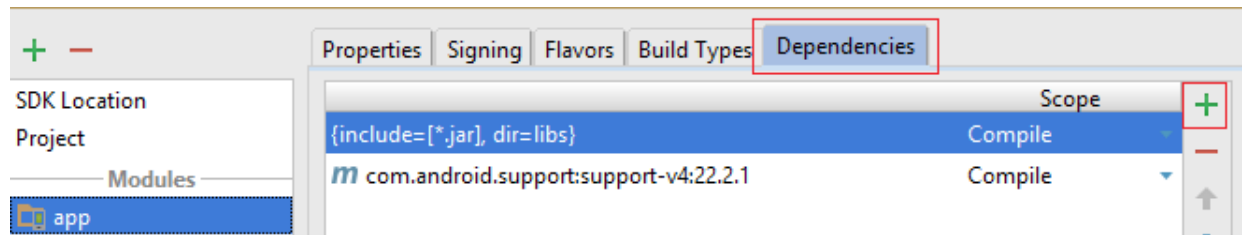


Figura 55 – Adicionando uma dependência ao projeto. **Fonte:**Elaborado pelos autores.

Na tela que foi aberta localizou-se a biblioteca GSON com o endereço da Google, logo após ela foi selecionada e clicou-se no botão Ok para adicioná-la, como apresenta a Figura 56.

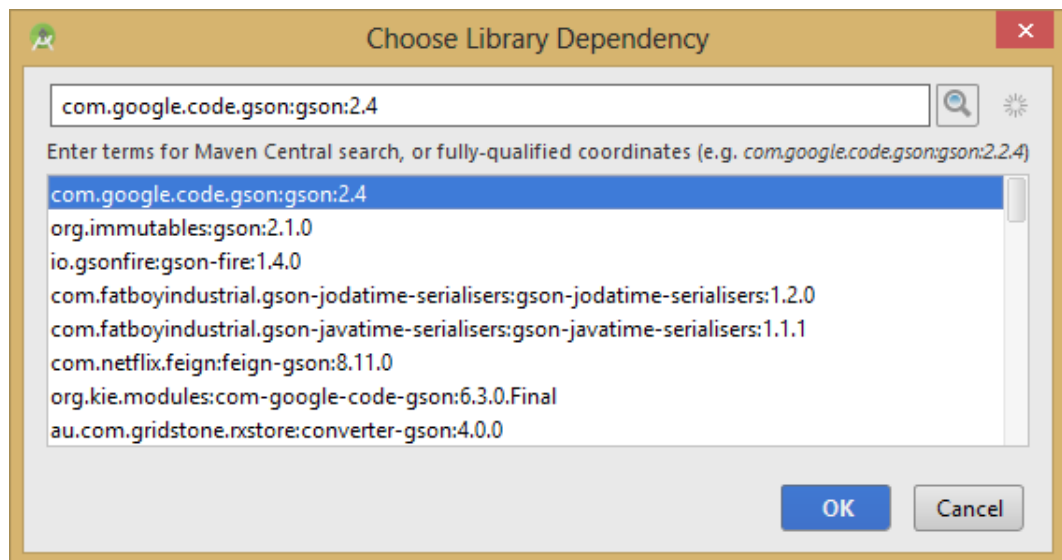


Figura 56 – Adicionando a biblioteca GSON ao projeto. **Fonte:**Elaborado pelos autores.

Na Figura 57, é mostrado o código em que é utilizada a biblioteca GSON. O sistema lê os dados vindos em JSON e envia para o GSON que faz a conversão dos dados no formato da classe EventTO.

```
1      ...
2      Reader reader = new InputStreamReader(content);
3      Gson gson = new Gson();
4      retornoEventos = gson.fromJson(reader, Events.class);
5      ...
```

Figura 57 – Usando GSON para conversão dos dados. **Fonte:**Elaborado pelos autores.

Depois, fez-se necessário construir uma classe que fizesse o gerenciamento dos dados vindos do banco de dados com a interface que listará as informações aos usuários. Essa

classe recebeu o nome de `ListResultsAdapter` e se estende da classe nativa do Android denominada `BaseExpandableListAdapter`.

Nesta classe foi criado um construtor que recebe a lista de disciplinas cursadas pelo aluno e uma lista com as notas de cada matéria. Os nomes das disciplinas foram inseridos em um *array* de *Strings*, já as notas foram inseridas em uma matriz. Este procedimento foi necessário devido à estrutura dos métodos `getGroupView()` e `getChildView()` responsável por apresentar na tela do dispositivo os nomes das disciplinas e as notas das matérias respectivamente. A Figura 58 apresenta o construtor da classe `ListResultsAdapter`.

```
1 public ListResultsAdapter(Context context, List<DisciplineTO>
   disciplines, List<EventTO> results){
2     this.context = context;
3     this.disciplines = disciplines;
4     this.results = results;
5     namesDisciplines = new String[disciplines.size()];
6     EventsDisciplines = new String[disciplines.size()][results.size
       ()];
7
8     for (int i = 0; i < disciplines.size(); i++){
9         Long idDiscipline = disciplines.get(i).get_id();
10        String nameDiscipline = disciplines.get(i).getName();
11        namesDisciplines[i] = nameDiscipline;
12        int position = 0;
13        int totalResults = 0;
14        for (int y = 0; y < results.size(); y++) {
15            String description = results.get(y).
               getDescription_event();
16            int value = results.get(y).getValue_event();
17            int result = results.get(y).getResult();
18            Long disciplineId = results.get(y).getId_discipline();
19
20            if (idDiscipline == disciplineId){
21                EventsDisciplines [i][position] = "Descricao: " +
                   description +
22                " Valor: " + value + " Nota: " + result;
23                position++;
24                totalResults += result;
25            }
26        }
27        EventsDisciplines [i][position] = "SOMA DAS NOTAS: " +
            totalResults;
28    }
29 }
```

Figura 58 – Construtor da classe `ListResultsAdapter`. **Fonte:**Elaborado pelos autores.

Após adicionados os dados no *array* e na matriz é preciso exibi-los ao estudante. Na Figura 59, pode se ver o método `getGroupView()` criando um *widget* do tipo *textView* e inserindo nele o nome da matéria, os espaçamentos, o tamanho da fonte e informando que as palavras serão escritas em negrito.

```

1 @Override
2     public View getView(int groupPosition, boolean isExpanded,
3                           View convertView, ViewGroup parent) {
4
5         TextView textViewDiscipline = new TextView(context);
6         textViewDiscipline.setText(namesDisciplines[groupPosition])
7         ;
8         textViewDiscipline.setPadding(40, 10, 0, 10);
9         textViewDiscipline.setTextSize(20);
10        textViewDiscipline.setTypeface(null, Typeface.BOLD);
11
12        return textViewDiscipline;
13    }

```

Figura 59 – Método getView(). **Fonte:**Elaborado pelos autores.

O método getChildView() segue a mesma lógica do método getView(), como ilustra a Figura 60.

```

1 @Override
2     public View getChildView(int groupPosition, int childPosition,
3                             boolean isLastChild, View convertView,
4                               ViewGroup parent) {
5
6         TextView textViewSubList = new TextView(context);
7         textViewSubList.setText(EventsDisciplines[groupPosition][
8             childPosition]);
9         textViewSubList.setPadding(10, 10, 10, 5);
10        textViewSubList.setTextSize(15);
11
12        return textViewSubList;
13    }

```

Figura 60 – Método getChildView(). **Fonte:**Elaborado pelos autores.

O próximo passo foi a 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 gerou dentro da pasta *layout* o arquivo XML *activity_list_results.xml* referente a ela. Neste, foi inserido apenas o *widget* *expandableListView*, que está incumbido de apresentar a lista de disciplinas que o discente está cursando e ao clicar em alguma dessas matérias serão apresentadas as notas referentes às atividades realizadas nesta disciplina. Na Figura 61 é possível ver o código XML de 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="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:theme="@style/AppTheme"
    tools:context="univas.edu.com.university.ListResultsActivity">

    <ExpandableListView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/expandableListView2"
        android:divider="#FFFFFF"
        android:dividerHeight="1dp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true" />

</RelativeLayout>

```

Figura 61 – Código XML do *layout* que apresentará a lista de notas. **Fonte:**Elaborado pelos autores.

Na classe `ListResultsActivity`, é preciso informar o *layout* a ser chamado através do método `setContentView()`. Além disso, também foi necessário passar para a classe `ListResultsAdapter` a lista de disciplinas que o discente está cursando e a lista com as notas referentes a cada matéria vindas do banco de dados. Na Figura 62 é exibido o método `onCreate()` da classe `ListResultsActivity`.

```

1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3      super.onCreate(savedInstanceState);
4      setContentView(R.layout.activity_list_results);
5      helper = new DatabaseHelper(this);
6
7      execute = new DatabaseExecute(helper);
8
9      ExpandableListView listView =
10         (ExpandableListView) findViewById(R.id.expandableListView2)
11         ;
12      listView.setAdapter(
13         new ListResultsAdapter(
14             this,
15             execute.getDisciplines(),
16             execute.getResults())
17     );
18 }

```

Figura 62 – Método `onCreate()` da classe `ListResultsActivity`. **Fonte:**Elaborado pelos autores.

Estes procedimentos que foram realizados para as classes `ListResultsActivity`

e `ListResultsAdapter` são necessários para se apresentar as notas dos exercícios resolvidos. Desta mesma forma foi preciso criar uma *activity* e um adapter tanto para faltas quanto para provas agendadas seguindo a mesma lógica.

No momento em que algum professor lançar notas, faltas ou provas agendadas no portal do aluno, é indispensável notificar o estudante. Com esse intuito desenvolveu-se uma classe chamada de `GcmIntentServiceUnivas` que estende `IntentService`, que recebe a mensagem vinda do GCM.

Esta classe recebe os dados em formato JSON, por isso ela transfere estas informações para o método `getJsonEvents()` da classe `HttpUtilUnivas`, o qual será responsável por ler os dados e realizar os procedimentos de gravação no banco de dados.

Ao salvar o evento, é chamado o método `sendNotification()`, que receberá um objeto da classe `EventTO`. Ele realiza uma análise do tipo de evento, para saber qual *activity* deve ser executada quando o usuário clicar na notificação. Logo após de adicionados os atributos da notificação, como o ícone, o título e a mensagem, a notificação é exibida ao usuário. Na Figura 63 é visível o método `sendNotification()`.

```

1 private void sendNotification(EventTO to) {
2     String msg;
3     DisciplineTO disciplineTO = execute.getDisipline(to.getId_discipline());
4     String nameDispline = disciplineTO.getName();
5     mNotificationManager = (NotificationManager)
6         this.getSystemService(Context.NOTIFICATION_SERVICE);
7     PendingIntent contentIntent;
8     if(to.getType_event().equals("PROVA_AGENDADA")){
9         List<String> data = new ArrayList<String>();
10        data.add(nameDispline);
11        data.add(to.getDescription_event());
12        data.add(String.valueOf(to.getValue_event()));
13        data.add(to.getDate_event());
14        Intent intent = new Intent(this, NotificationAgendasActivity.class);
15        intent.putExtra("dados", (ArrayList<String>)data);
16        contentIntent = PendingIntent.getActivity(this, 0, intent, 0);
17
18        msg = "Prova agendada dia" + to.getDate_event();
19    }else{
20        if(to.getType_event().equals("FALTAS")){
21            List<String> data = new ArrayList<String>();
22            data.add(nameDispline);
23            data.add(to.getDate_event());
24            data.add(String.valueOf(to.getValue_event()));
25            Intent intent = new Intent(this, NotificationFoulsActivity.class);
26            intent.putExtra("dados", (ArrayList<String>)data);
27
28            contentIntent = PendingIntent.getActivity(this, 0, intent, 0);
29
30            msg = to.getValue_event() + " falta(s) recebidas";
31        }else{
32            List<String> data = new ArrayList<String>();
33            data.add(nameDispline);
34            data.add(to.getDescription_event());
35            data.add(String.valueOf(to.getValue_event()));
36            data.add(String.valueOf(to.getResult()));
37            Intent intent = new Intent(this, NotificationResultsActivity.class);
38            intent.putExtra("dados", (ArrayList<String>)data);
39
40            contentIntent = PendingIntent.getActivity(this, 0, intent, 0);
41            msg = "Nova nota " + to.getResult();
42        }
43    }
44
45    Uri soundUri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
46
47    NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)
48        .setSmallIcon(R.drawable.notification_univas)
49        .setContentTitle("Univas informa")
50        .setAutoCancel(true)
51        .setStyle(new NotificationCompat.BigTextStyle()
52            .bigText(msg))
53        .setContentText(msg)
54        .setSound(soundUri);
55
56    Vibrator vibrator = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
57    long milliseconds = 30;
58    vibrator.vibrate(milliseconds);
59
60    mBuilder.setContentIntent(contentIntent);
61    mNotificationManager.notify(NOTIFICATION_ID, mBuilder.build());
62 }

```

Figura 63 – Método sendNotification(). **Fonte:**Elaborado pelos autores.

A notificação acontece toda vez que o GCM envia uma informação ao dispositivo. Para tratar essas ocorrências foi projetada uma classe para ser o BroadcastReceiver chamada de GcmBroadcastReceiverUnivas. Esta classe se estende da classe nativa do Android, WakefulBroadcastReceiver e possui apenas método onReceive(), o qual receberá a *intent* a ser chamada quando chegar algum dado do GCM. Na Figura 64, vê-se a

classe `GcmBroadcastReceiverUnivas` que, através do método `onReceive()` inicializará a classe `GcmIntentServiceUnivas`.

```
1 public class GcmBroadcastReceiverUnivas extends
    WakefulBroadcastReceiver {
2
3     @Override
4     public void onReceive(Context context, Intent intent) {
5         ComponentName comp = new ComponentName(context.
            getPackageName(), GcmIntentServiceUnivas.class.getName
            ());
6         startWakefulService(context, (intent.setComponent(comp)));
7         setResultCode(Activity.RESULT_OK);
8     }
9 }
```

Figura 64 – Classe `GcmBroadcastReceiverUnivas`. **Fonte:**Elaborado pelos autores.

No entanto para configurar esta classe como um `BroadcastReceiver`, foi preciso adicioná-la na tag `<receiver>` do arquivo `AndroidManifest.XML`, como mostra a Figura 65.

```
<receiver
    android:name=".model.GcmBroadcastReceiverUnivas"
    android:permission="com.google.android.c2dm.permission.SEND" >
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE" />
        <category android:name="univas.edu.com.university.model" />
    </intent-filter>
</receiver>
```

Figura 65 – Configuração do `BroadcastReceiver` no `AndroidManifest.XML`. **Fonte:**Elaborado pelos autores.

Por fim, foi construída uma classe chamada de `GcmControllerUnivas` que tem por objetivo configurar o dispositivo para trabalhar com o GCM.

Nesta classe existe um método denominado `checkPlayServices()` como intuito de verificar se o dispositivo possui os requisitos necessários para o GCM. Na Figura 66, é apresentado o método `checkPlayServices()`.

```

1 public boolean checkPlayServices() {
2
3     int resultCode = GooglePlayServicesUtil.
        isGooglePlayServicesAvailable(context);
4
5     if (resultCode != ConnectionResult.SUCCESS) {
6
7         if (GooglePlayServicesUtil.isUserRecoverableError(
            resultCode)) {
8
9             GooglePlayServicesUtil.getErrorDialog(
10                 resultCode,
11                 new MainActivity(),
12                 PLAY_SERVICES_RESOLUTION_REQUEST
13             ).show();
14
15         } else {
16             Log.i(TAG, "Este dispositivo nao suporta o GCM.");
17         }
18         return false;
19     }
20     return true;
21 }

```

Figura 66 – Método checkPlayServices(). **Fonte:**Elaborado pelos autores.

Logo após é chamado o método `getRegistrationId()`, que é incumbido de retornar o *Registration ID*. Na Figura 67 é possível ver este método, no entanto, se o registro retornado for nulo, isso significa que o aparelho não está cadastrado nos servidores da Google, então é executado o método `registerInBackground()` que fará esse cadastro, como demonstra a Figura 68.

```

1 private String getRegistrationId(Context context) {
2
3     final SharedPreferences prefs = getGcmPreferences(context);
4
5     String registrationId = prefs.getString(PROPERTY_REG_ID, "");
6     if (registrationId.isEmpty()) {
7         Log.i(TAG, "Falha ao registrar.");
8         return "";
9     }
10
11     int registeredVersion = prefs.getInt(PROPERTY_APP_VERSION,
        Integer.MIN_VALUE);
12     int currentVersion = getAppVersion(context);
13     if (registeredVersion != currentVersion) {
14         Log.i(TAG, "Versao alterada do aplicativo.");
15         return "";
16     }
17     return ;
18 }

```

Figura 67 – Método getRegistrationId(). **Fonte:**Elaborado pelos autores.

```

1 private void registerInBackground() {
2
3     new AsyncTask<Void, Void, String>() {
4         @Override
5         protected String doInBackground(Void... params) {
6             String msg = "";
7             try {
8                 if (gcm == null) {
9                     gcm = GoogleCloudMessaging.getInstance(
10                         context);
11                 }
12                 regid = gcm.register(SENDER_ID);
13                 msg = "Dispositivo registrado, registro ID=" +
14                     regid;
15
16                 sendRegistrationIdToBackend(regid);
17
18                 storeRegistrationId(context, regid);
19             } catch (IOException ex) {
20                 msg = "Error : " + ex.getMessage();
21             }
22             return msg;
23         }
24
25         @Override
26         protected void onPostExecute(String msg) {
27             Toast.makeText(new MainActivity(), msg, Toast.
28                 LENGTH_SHORT).show();
29         }
30     }.execute(null, null, null);
31 }

```

Figura 68 – Método registerInBackground(). **Fonte:**Elaborado pelos autores.

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

4 DISCUSSÃO DE RESULTADOS

Este capítulo tem por finalidade descrever os resultados obtidos nesta pesquisa através de uma explicação teórico-prática.

O presente trabalho teve por objetivos desenvolver uma solução que permitisse disponibilizar informações da Universidade do Vale do Sapucaí através de serviços e a construção de um aplicativo que utilize o serviço criado, permitindo aos seus alunos consultarem suas notas, faltas e provas agendadas do semestre corrente através de seus dispositivos móveis com sistema operacional Android.

O *web service* disponibiliza as informações via REST. Uma documentação básica da API pode ser vista a seguir.

1. Contexto da Aplicação - <http://<enderecoDoServidor>/WebServiceAppUnivas/>

- **Contextos internos**

- **students** - contexto relacionados aos dados dos alunos

- * **events** - recurso referente aos dados sobre eventos relacionados a alunos;

- **GET** - recebe como parâmetro a matrícula do aluno através da url, e devolve um JSON, com todos os eventos do semestre corrente relacionados ao aluno;

- Ex.: <http://<enderecoDoServidor>/WebServiceAppUnivas/students/events/98004095>

- **POST** - Não Implementado

- **PUT** - Não Implementado

- **DELETE** - Não Implementado

- * **disciplines** - recurso referente aos dados sobre disciplinas de alunos;

- **GET** - recebe como parâmetro a matrícula do aluno através da url, e devolve um JSON, com todas as disciplinas do semestre corrente relacionadas ao aluno;

- Ex.: <http://<enderecoDoServidor>/WebServiceAppUnivas/students/disciplines/98004095>

- **POST** - Não Implementado

- **PUT** - Não Implementado
- **DELETE** - Não Implementado
- **users** - contexto relacionado aos dados dos usuários para ser utilizado no futuro para login
 - * **GET** - Não Implementado
 - * **POST** - Não Implementado
 - * **PUT** - recebe um JSON com os dados dos usuário e retorna um cabeçalho com o código de *status* 200;
 - * **DELETE** - Não Implementado

O aplicativo Android, consome o serviço do *web service* e do GCM trazendo as informações aos estudantes de forma hábil, além de ser mais cômodo para o usuário, uma vez que ele recebe os dados onde quer que esteja, desde que possua acesso à internet.

Além do mais, o software notifica o discente no momento em que é lançado um novo evento beneficiando-o, tendo em vista que ele não precisa ficar verificando se a nota já foi postada pelo professor, pois o sistema se encarrega de avisá-lo.

Como o aplicativo foi desenvolvido para a plataforma Android, ele pode ser instalado em dispositivos de diferentes fabricantes, evitando ficar preso a um hardware específico.

Durante o período de desenvolvimento, foram realizados vários testes através do emulador Android, presente na IDE Android Studio e em um *smartphone* Samsung S3 Mini. Notou-se aí, que no dispositivo real a velocidade da execução do aplicativo é bem maior se comparada ao emulador.

Como os testes foram realizados sempre nos mesmos modelos de dispositivos, há a possibilidade de ocorrer alguns problemas em relação ao *layout* dependendo do tamanho da tela do equipamento, contudo o fator de lógica da aplicação não será afetado.

O aplicativo desenvolvido possui fácil usabilidade, evitando com que o aluno fique perdido ao buscar alguma informação. Para a organização das opções que o software oferece aos discentes, foi criada uma tela do tipo *Navigation Drawer Layout* que, de acordo com Android (2015b), é um painel que normalmente fica escondido e aparece quando clicado no ícone do aplicativo no canto superior esquerdo, o qual contém os dados de navegação do software semelhante a um menu. Na Figura 69, é ilustrado o painel de navegação do aplicativo.



Figura 69 – Tela principal do aplicativo com as opções de navegação. **Fonte:**Elaborado pelos autores.

Para apresentar as informações, foi utilizada um `ExpandableListView` que segundo Android (2015d), é um *widget* que exibe uma lista de itens e ao selecionar um desses elementos, a tela é estendida apresentando os subitens da opção escolhida. Na Figura 70 é mostrada a tela que lista as disciplinas sendo que Tópicos Avançados está selecionada.

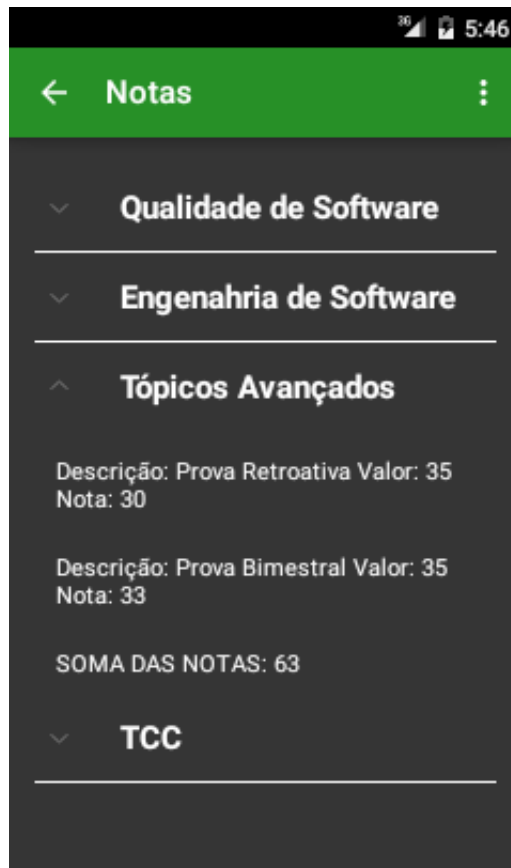


Figura 70 – Lista de disciplinas. **Fonte:**Elaborado pelos autores.

O GCM foi utilizado neste projeto para fazer a transmissão dos eventos do servidor para o aplicativo. Com ele, a troca de dados tornou-se mais rápida e simples, pois toda a lógica de entrega fica por conta da Google.

Quando se trata de apenas uma informação para um único dispositivo, o controle é simples, mas a partir do momento em que há vários equipamentos para receber informações distintas, o gerenciamento torna-se mais complexo. Por isso, pode-se afirmar que o GCM solucionou este problema, mostrando-se eficaz na transmissão dos dados, além de ser de fácil configuração. A Figura 71, ilustra uma notificação logo após o GCM ter entregue uma mensagem ao aplicativo. Ao baixar a paleta de notificação será apresentado um resumo da notificação, como apresenta a Figura 72.



Figura 71 – Notificação na barra de notificações do dispositivo. **Fonte:**Elaborado pelos autores.

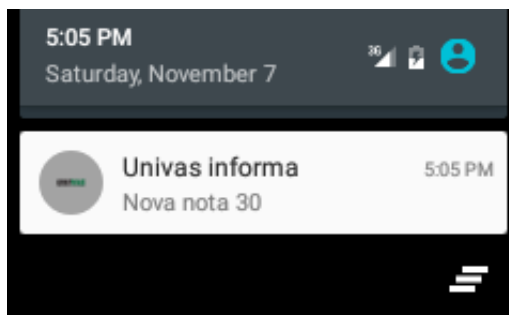


Figura 72 – Paleta de notificação estendida. **Fonte:**Elaborado pelos autores.

Ao clicar na notificação é apresentada para o aluno a tela que exibe os dados da notificação. Na Figura 73, pode-se ver a tela exibindo as informações de um evento de notas.

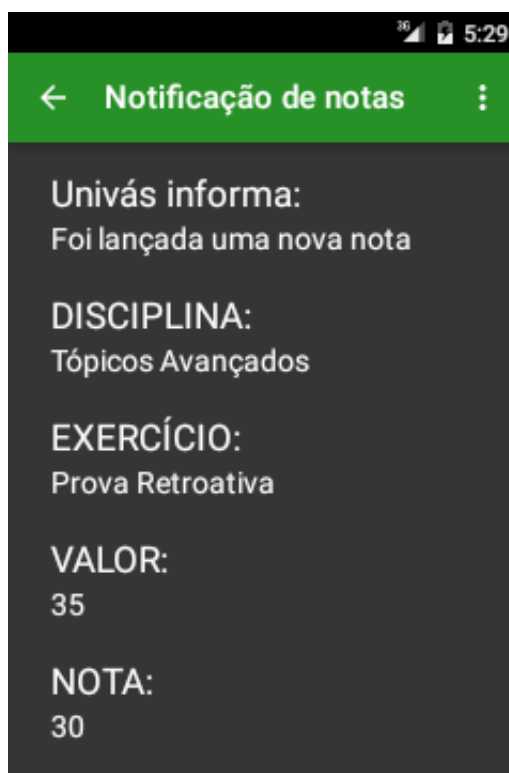


Figura 73 – Tela exibindo os dados após clicado na notificação. **Fonte:**Elaborado pelos autores.

Portanto, sabendo-se que esta pesquisa enquadra-se no tipo de pesquisa aplicada, a qual tem por finalidade desenvolver um produto real para resolver um determinado problema e que a solução construída auxilia tanto a Univás, que terá um nova forma de disponibilizar suas informações, quanto aos alunos, que possuem a opção de consultarem suas informações acadêmicas através de dispositivos móveis, entende-se que este trabalho atingiu suas expectativas.

5 CONCLUSÃO

Com a realização desta pesquisa, foi criado um *web service* que responde à requisições via REST, oferecendo à Univás uma oportunidade de disponibilizar suas informações acadêmicas através de serviços. No presente momento, o serviço possível de se utilizar é a consulta de notas, faltas e provas agendadas.

Além disso foi construído um aplicativo que tem por finalidade exemplificar uma possibilidade de uso do serviço criado. Além do mais, no momento em que algum professor lançar alguma informação acadêmica referente ao aluno, o aplicativo do mesmo deve ser notificado.

As mensagens enviadas do servidor para o dispositivos móveis são transmitidas através da API Google Cloud Messaging (GCM), da Google, que oferece o recurso gratuitamente e que mostrou-se muito eficaz, solucionando o problema do envio de notificações aos dispositivos dos alunos e de transmissão dos dados.

Deve-se também destacar o grande número de materiais disponibilizados por desenvolvedores e estudiosos da área, os quais possibilitaram aos autores desta pesquisa o estudo e aprendizagem das teorias apresentadas nesta pesquisa, bem como suas implementações.

Portanto, apesar das dificuldades encontradas para a realização deste trabalho, como realizar a comunicação entre o *web service* e o aplicativo Android, percebeu-se que o software apresentado nesta pesquisa é de grande utilidade aos alunos da Universidade do Vale do Sapucaí, pois conseguem acompanhar seus desempenhos escolares através de seus equipamentos *mobile* e a Univás que tem, agora, uma estrutura pronta para disponibilizar suas informações através de serviços.

Devido ao crescente número de dispositivos móveis é possível perceber uma época favorável para explorar essas tecnologias. Sendo assim, este projeto possibilita aos graduandos em Sistemas de Informação uma oportunidade para acrescentar novas funcionalidades para esta aplicação em trabalhos futuros.

Devido ao tempo escasso esse trabalho não trata a parte de segurança, podendo ser implementado em outra oportunidade. São exibidas, apenas as disciplinas do semestre corrente, sendo possível acrescentar a funcionalidade para exibir todas as matérias já cursadas. Também pode-se criar o serviço para que os alunos realizem a CPA, consultas de livros da biblioteca, permitir aos professores lançarem notas no portal do aluno ou publicarem materiais, além de possibilitar o acesso a outras plataformas como Windows Phone e IOS.

Por fim, pode-se afirmar que o presente trabalho realizou seus objetivos, os quais eram desenvolver uma estrutura para a universidade poder disponibilizar informações em forma de serviço, o que hoje ainda não acontece, e possibilitar aos alunos da Univás consultarem suas notas, faltas e provas agendadas, além de serem notificados quando estes eventos ocorrem. Ainda, esta pesquisa foi de grande relevância aos participantes do projeto, pois contribuiu por uma ampla visão de resolução de problemas e um conhecimento vasto nas tecnologias utilizadas.

REFERÊNCIAS

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

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

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

ANDROID. **ExpandableListView.** 2015. Disponível em: <<http://developer.android.com/reference/android/widget/ExpandableListView.html>>. Acesso em: 24 Agosto de 2015.

APACHE. **What is maven.** 2015. Disponível em: <<http://maven.apache.org/what-is-maven.html>>. Acesso em: 29 de Outubro 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**. 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, 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.

LECHETA, R. R. **Web services RESTful: aprenda a criar web services restful em java na nuvem do google**. 1ª. ed. São Paulo: Novatec, 2015.

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.

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.

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.

Apêndices

CRIAÇÃO DE UM PROJETO DYNAMIC WEB PROJECT NO ECLIPSE LUNA

Para proceder com a criação de um projeto do tipo *Dynamic Web Project* no Eclipse, é necessário acessar na IDE, a opção **File -> New-> Dynamic Web Project** como pode ser visto na Figura 74.

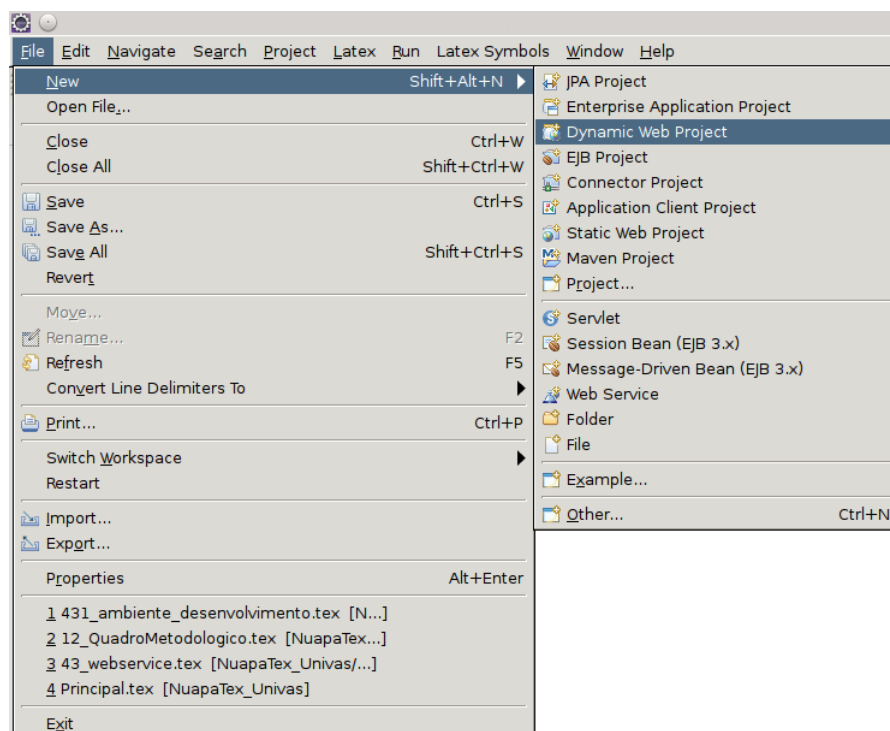


Figura 74 – Tela New Database. . . **Fonte:**Elaborado pelos autores.

Em seguida foi apresentada uma tela para o preenchimento de alguns dados requeridos para a criação do projeto. Destas informações somente foi preenchido o nome do projeto. As outras informações continuaram sendo as que vem por padrão da IDE. A janela apresentada e as informações preenchidas podem ser vistas na Figura 75.

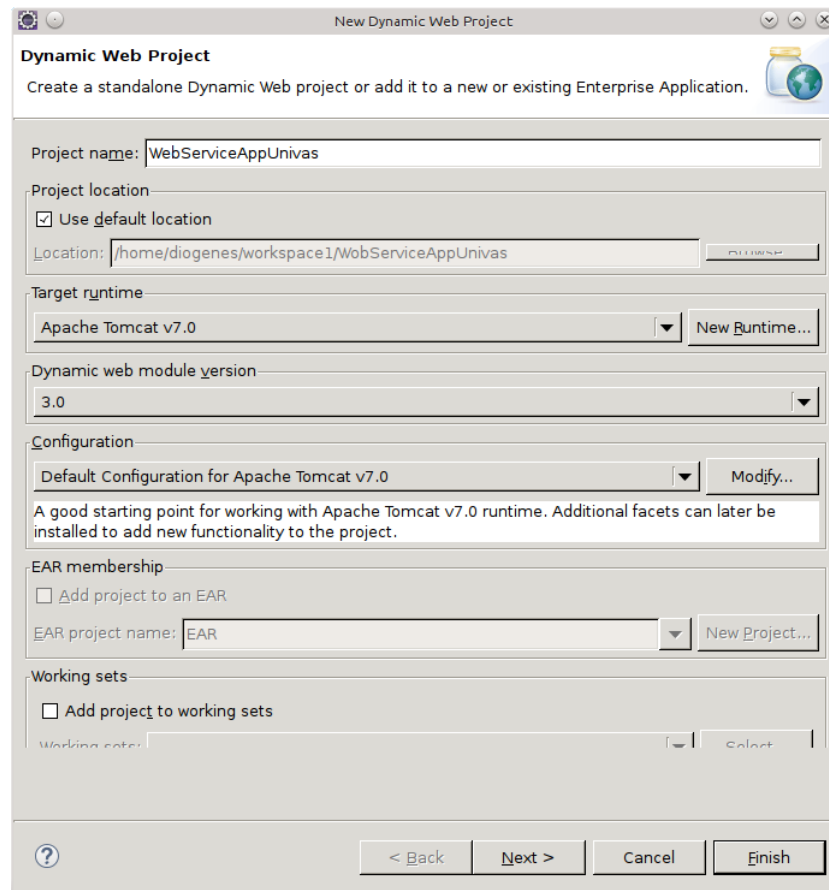


Figura 75 – Tela para criação de um novo projeto no Eclipse. **Fonte:**Elaborado pelos autores.

Na próxima janela apresentada, que têm por função configurar a pasta de códigos do projeto manteve-se a configuração apresentada pela IDE, como mostra a Figura 76.

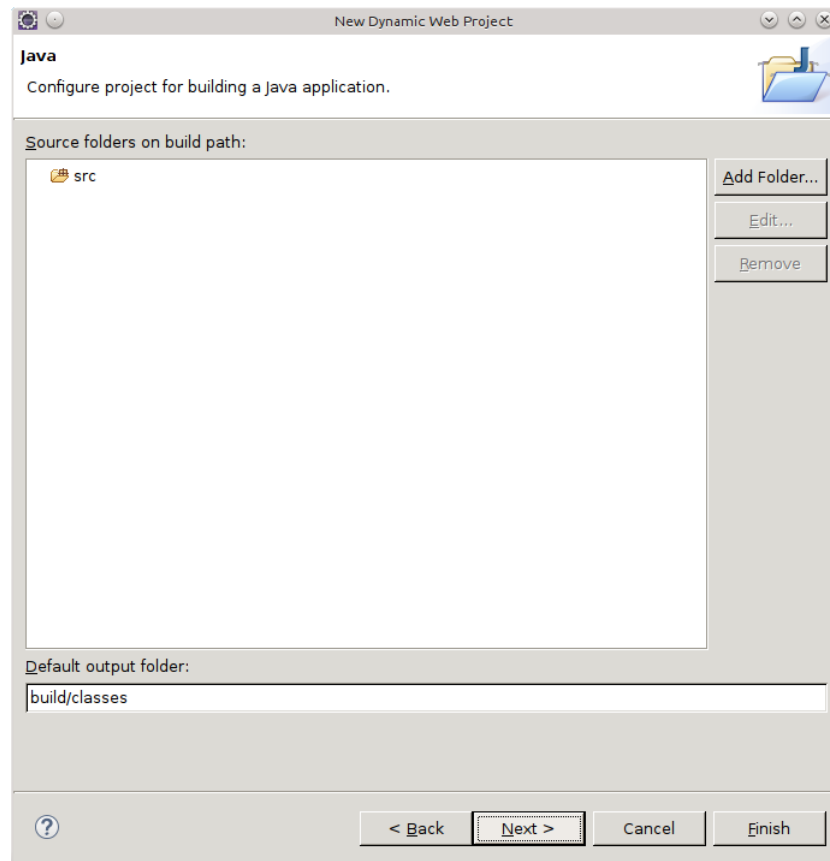


Figura 76 – Tela para criação de um novo projeto no Eclipse. **Fonte:**Elaborado pelos autores.

Na sequência, na tela que foi apresentada era necessário preencher o campo **Context root** com o contexto principal da aplicação web que acabou mantendo o próprio nome da aplicação. Além disso foi marcado a opção **Generate web.xml deployment descriptor**, para que ao criar o projeto, a própria IDE criasse o arquivo `web.xml`, arquivo responsável por algumas configurações da aplicação web. Esta tela está apresentada na Figura 77.

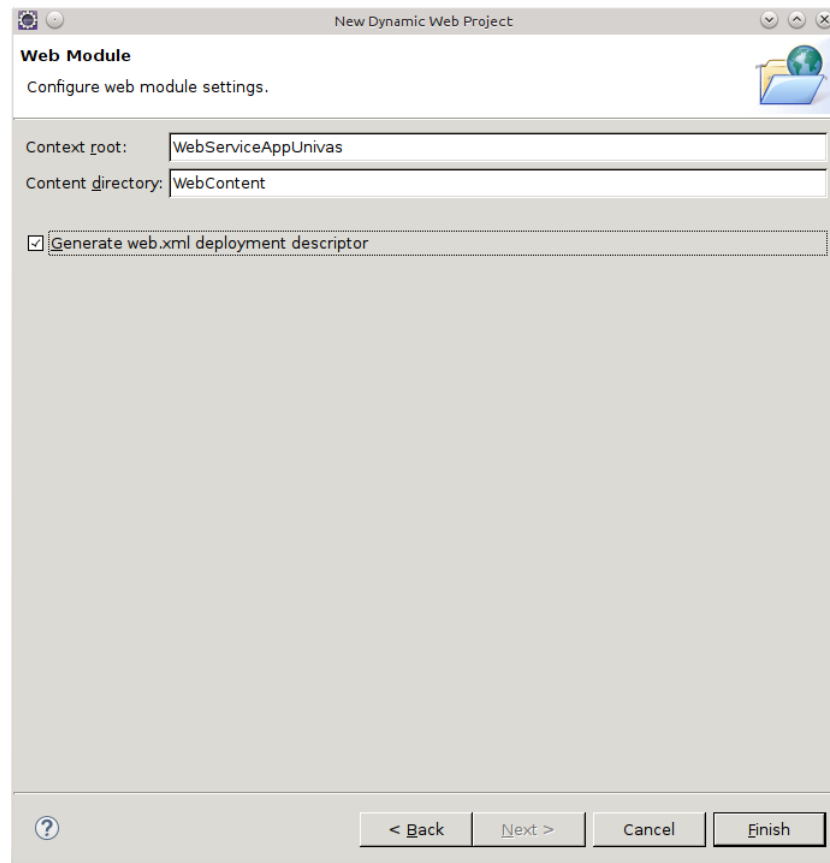


Figura 77 – Tela para criação de um novo projeto no Eclipse. **Fonte:**Elaborado pelos autores.

Após este passo foi concluído a criação do projeto.