

**DIEGO D'LEON NUNES
DIÓGENES APARECIDO REZENDE**

APLICATIVO PARA CONSULTA DE NOTAS

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

2015

**DIEGO D'LEON NUNES
DIÓGENES APARECIDO REZENDE**

APLICATIVO PARA CONSULTA DE NOTAS

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

Orientador: Prof. MSc. Roberto Rocha Ribeiro

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

2015

SUMÁRIO

1	QUADRO METODOLÓGICO	3
1.1	Tipo de pesquisa	3
1.2	Contexto de pesquisa	3
1.3	Instrumentos	4
1.4	Procedimentos e Resultados	5
1.4.1	Modelagem	6
1.4.2	Aplicativo	9
1.4.3	<i>Webservice</i>	12
REFERÊNCIAS		18

1 QUADRO METODOLÓGICO

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

1.1 Tipo de pesquisa

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

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

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

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

1.2 Contexto de pesquisa

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


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

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

1.3 Instrumentos

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

Foi realizado um questionário simples, que está apresentado na Figura 1, contendo quatro perguntas e enviado para *e-mails* de alguns alunos da universidade. O foco desse questionário era saber o motivo pelo qual os usuários mais acessavam o portal do aluno e se tinham alguma dificuldade em encontrar o que procuravam. Obteve-se um total de treze respostas, no qual pode-se perceber que a maioria dos entrevistados afirmam terem dificuldades para encontrar o que precisam e que o sistema não avisa quando ocorre alguma alteração. Sobre o motivo do acesso cem por cento respondeu que entram no sistema web para consultar suas notas.



Pesquisa sobre o portal do aluno

Qual é sua opinião sobre o portal do aluno?

☐ Ótimo
☐ Bom
☐ Ruim
☐ Péssimo

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

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

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

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

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

☐ Ótimo
☐ Bom
☐ Ruim
☐ Péssimo

100% concluído

Figura 1 – Questionário Aplicado. **Fonte:**Elaborado pelos autores.

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

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

1.4 Procedimentos e Resultados

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

1.4.1 Modelagem

Sabendo-se que o planejamento é um passo importante, além do levantamento de requisitos foram utilizados os diagramas de UML para nortear o desenvolvimento. Para se ter uma visão das funcionalidades do sistema por parte do usuário foi criado o diagrama de casos de uso do aplicativo, como demonstra na Figura 2

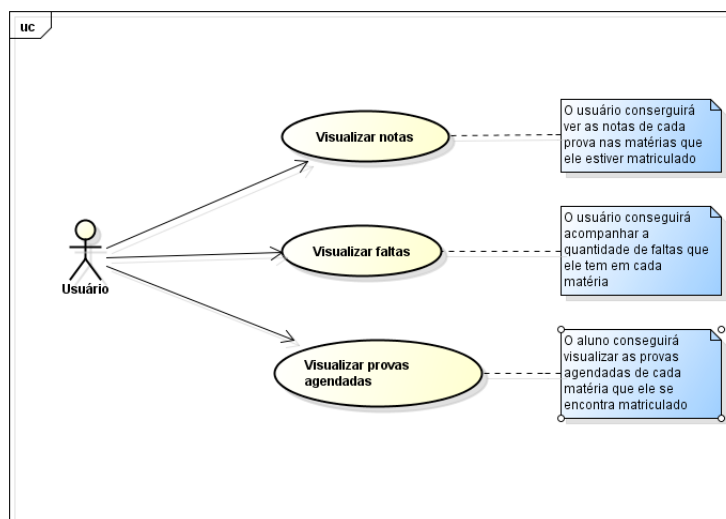


Figura 2 – Diagrama de Casos de Uso. **Fonte:**Elaborado pelos autores.

Logo após, foi projetado o diagrama de atividades para se ter uma visão da sequência do fluxo do sistema, conforme mostra a Figura 3.

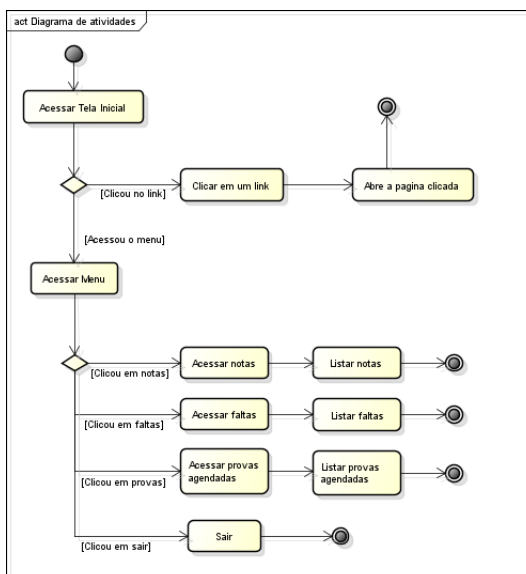
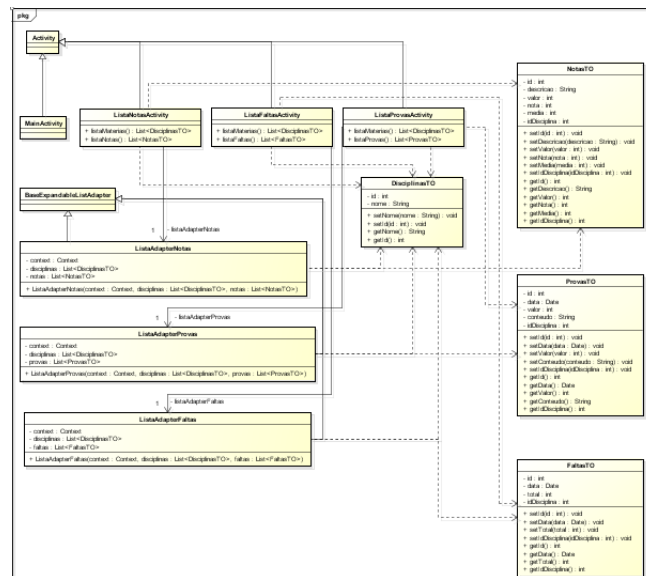


Figura 3 – Diagrama de atividades. **Fonte:**Elaborado pelos autores.

A próxima etapa realizada foi o desenvolvimento do diagrama de classes do aplicativo

Android, o qual tem por objetivo mostrar todas as classes que o sistema necessita, como pode-se observar na Figura 4.



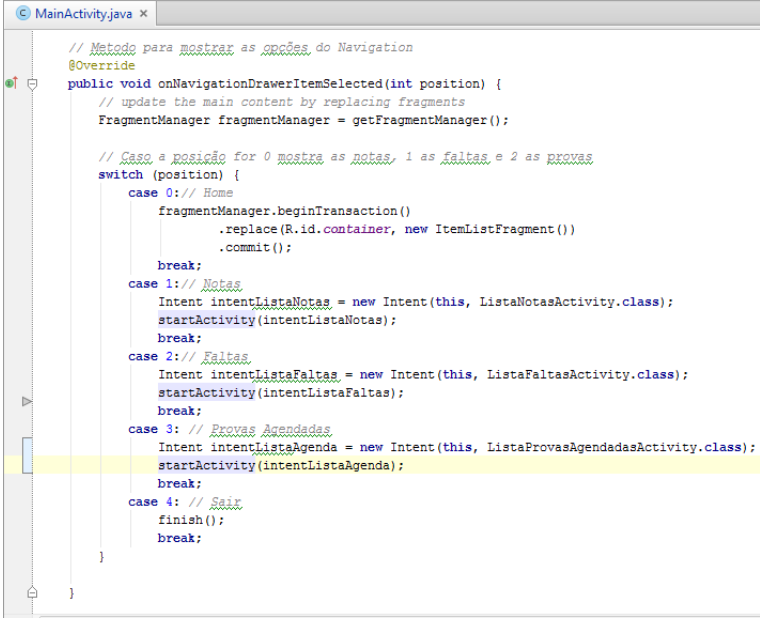
Através de um levantamento de requisitos parciais e das reuniões entre os participantes foi possível construir um Diagrama de Entidade e Relacionamento, no qual ficou definida a estrutura do banco de dados da aplicação. A Figura 5 mostra o Diagrama de Entidade e Relacionamento concebido para esta pesquisa.

Fazendo uso desse diagrama foi possível criar todas as classes *Java* que representam as entidades do mapeamento objeto-relacional.

1.4.2 Aplicativo

Para iniciar o desenvolvimento do aplicativo, primeiramente fez-se necessária a instalação e configuração da plataforma *Android Studio* versão 1.1.0 e *Android SDK* versão 24.0.2.

Logo após a configuração dos ambientes de desenvolvimento, o primeiro passo foi a criação de uma *activity main*, denominada *MainActivity*, a qual será executada quando a aplicação for iniciada. O tipo de *activity* escolhida é do tipo *Navigation Drawer Layout*. Na Figura 6, pode-se ver o método `onNavigationDrawerItemSelected()`, que tem por finalidade mostrar as opções de navegação e o que deverá acontecer quando uma delas for clicada. Um exemplo, caso o usuário escolha o item *Notas*, o software executará o *case* um, o qual criará uma *intent* e chamará a classe *ListaNotasActivity*.



```
// Metodo para mostrar as opções do Navigation
@Override
public void onNavigationDrawerItemSelected(int position) {
    // update the main content by replacing fragments
    FragmentManager fragmentManager = getFragmentManager();

    // Caso a posição for 0 mostra as notas, 1 as faltas e 2 as provas
    switch (position) {
        case 0: // Home
            fragmentManager.beginTransaction()
                .replace(R.id.container, new ItemListFragment())
                .commit();
            break;
        case 1: // Notas
            Intent intentListaNotas = new Intent(this, ListaNotasActivity.class);
            startActivity(intentListaNotas);
            break;
        case 2: // Faltas
            Intent intentListaFaltas = new Intent(this, ListaFaltasActivity.class);
            startActivity(intentListaFaltas);
            break;
        case 3: // Provas Agendadas
            Intent intentListaAgenda = new Intent(this, ListaProvasAgendadasActivity.class);
            startActivity(intentListaAgenda);
            break;
        case 4: // Sair
            finish();
            break;
    }
}
```

Figura 6 – MainActivity. Fonte:Elaborado pelos autores.

O próximo passo, foi criar a *activity Home*, o qual trará uma lista de *sites* uteis, com as opções Univás, MEC, Fies, Prouni e *Google* acadêmico. Para que essa lista aparecesse foi utilizada uma *activity* do tipo *Master/Deital Flow* que já traz consigo o *widget* de *ListView*. Na Figura 7, pode-se ver a classe *DummyContent*, local onde é declarado a lista de sites.

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

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

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

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

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

Figura 7 – Classe *DummyContent*. **Fonte:**Elaborado pelos autores.

Quando clicado em algum item, é executada a classe *ItemDetailActivity*, que chamará o *ItemDetailFragment*, responsável por carregar o *site* em questão no *layout* do aplicativo. Abaixo, na Figura 8, vê-se o método responsável por passar para o *layout* o endereço do *site*.

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

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

    return rootView;
}
```

Figura 8 – Método para carregar o *layout* com *web view*. **Fonte:**Elaborado pelos autores.

Para que as informações possam aparecer, foram criadas *activities* do tipo *Blank Activity*. No *layout* dessas *activities* foram inseridas o *widget ExpandableListView*, como é mostrado na Figura 9.

```

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

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

</RelativeLayout>

```

Figura 9 – Layout com *expandable*. Fonte:Elaborado pelos autores.

Foi criada uma classe chamada *BuscaDados*, que tem por finalidade receber os dados do *webservice*, salvá-los no banco de dados local do aplicativo e entregá-los para as classes que implementam o *Adapter* para listar as informações na tela do dispositivo.

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

```

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

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

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

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

```

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

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

1.4.3 *Webservice*

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

1.4.3.1 Montagem do Ambiente de Desenvolvimento

No que diz respeito à contrução do *webservice*, foi necessária a instalação e configuração de um ambiente de desenvolvimento compatível com as necessidades apresentadas pelo *software*.

A princípio foi instalado o *Servlet Container Apache Tomcat* em sua versão de número 7. Esse *Servlet Container* foi instalado pois implementa a API da especificação *Servlets 3.0* do *Java*. Isso era necessário pelo fato que o *framework Jersey* usa *servlets* para disponibilizar serviços REST. Além disso o *Apache Tomcat* foi escolhido, para que o *WebService* pudesse fornecer os serviços necessários para o consumo, na arquitetura REST, que sugere o uso do protocolo HTTP¹ para troca de mensagens, pois além da funcionalidade com *Servlets*, o *Apache Tomcat* também é um servidor HTTP.

O *Apache Tomcat* foi instalado no ambiente de desenvolvimento, por meio do *download* de um pacote .zip no site oficial do mesmo. Esta abordagem permitiu a integração do *Apache Tomcat* com o IDE² *Eclipse*.

No ambiente de produção como está sendo usado um sistema operacional baseado em GNU/Linux, o mesmo foi instalado através do gerenciador de pacotes da distribuição. Em ambos os casos não foram necessárias configurações além do trivial de cada plataforma.

¹ HTTP - Hypertext Transfer Protocol

² IDE - *Integrated Development Environment*

Para armazenar os dados gerados e/ou recebidos, foi necessário fazer a instalação do Sistema Gerenciador de Banco de Dados(SGBD) *PostGreSql* na sua versão de número 9.2.

Eclipse ide

Maven plugin

1.4.3.2 Desenvolvimento

Essas classes foram criadas fazendo uso de anotações próprias do *Hibernate*, que é um *framework* que implementa a especificação JPA³. Essas classes fazem parte dos mecanismos de persistência de dados e são simplesmente t ou seja, objetos simples que contêm somente atributos privados e os métodos *getters* e *setters* que servem apenas para encapsular estes atributos. Uma das classes criadas, foi a classe `Aluno.java` que representa a tabela `alunos` no banco de dados e está representada na Figura 11.

³ JPA - *Java Persistence API*

```

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

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

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

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

    @Column(nullable = false)
    private Integer periodo;

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

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

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

    /**
     *
     * GETTERS E SETTERS
     *
     */

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

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

```

Figura 11 – Classe Aluno. **Fonte:**Elaborado pelos autores.

Foram criadas outras classes *Java* com a mesma finalidade da anterior, porém com pequenas diferenças no que diz respeito à atributos, metodos e anotações. Estas classes representam, de maneira individual, as tabelas no banco de dados. Certos atributos dessas classes têm por finalidade representar as colunas de cada tabela. Já os atributos que armazenam instâncias de outras classes ou até mesmo conjuntos (coleções) de instâncias representam os relaciona-

mentos entre as tabelas. E por fim, para cada classe que representa uma entidade, foi necessário implementar os métodos `hashCode` e `equals`, para que estas pudessem facilmente ser comparadas e diferenciadas em relação aos seus valores, haja visto que cada instância destas classes representa um registro no banco de dados.

Em seguida à criação das entidades, foi necessário configurar o arquivo `persistence.xml` que fica dentro do *classpath* do projeto *Java* ou seja, dentro da mesma pasta onde estão contidos pacotes do projeto. Este arquivo é extremamente importante, pois é nele que estão todas as configurações relativas à conexão com o banco de dados, configurações referentes ao Dialeto SQL que vai ser usado para as consultas e configurações referentes ao *persistence unit* que é o conjunto de classes mapeadas para o banco de dados. O arquivo `persistence.xml` está exposto no código 12.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
  xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="WsAppUnivas" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:postgresql://localhost:5432/wsappunivas" />
      <property name="javax.persistence.jdbc.user" value="postgres" />
      <property name="javax.persistence.jdbc.password" value="password" />

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

Figura 12 – Arquivo `persistence.xml`. **Fonte:**Elaborado pelos autores.

Em seguida à confecção do `persistence.xml` foi criada a classe `JpaUtil` que está representada na Figura 13. Esta classe é responsável por criar uma `EntityManagerFactory` que é uma fábrica de instâncias de `EntityManager` que nada mais é que um *persistence unit* ou unidade de persistência. Essa classe tem a responsabilidade de prover um modo de comunicação entre a aplicação e o banco de dados. No entanto a classe `JpaUtil` cria uma única instância de `EntityManagerFactory`, que é responsável por disponibilizar e gerenciar as instâncias de `EntityManager` de acordo com a necessidade da aplicação.


```

public class JpaUtil {
    private static EntityManagerFactory factory;

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

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

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

```

Figura 13 – Classe JpaUtil. **Fonte:**Elaborado pelos autores.

Em seguida à construção das classes que fazem a parte da persistência de dados, foi desenvolvido a parte de disponibilização de serviços *RESTful*, fazendo uso do *framework Jersey*. Com isso pode-se construir a classe que representa o primeiro serviço do *webservice*, que é a classe *Alunos*. Essa classe representa um contexto REST, e portanto, dispõe de alguns recursos. Esses recursos fazem a recuperação e a transmissão dos dados do *webservice* para o aplicativo *Android*. Essa classe e seus respectivos métodos estão representada na Figura 14.

```

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

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

        return alunos;
    }

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

```

Figura 14 – Classe AlunosService. **Fonte:**Elaborado pelos autores.

O *webservice* pode fazer a busca de alunos pelo id passado ou retornar uma coleção de eventos vinculados a um alunos, dependendo do recurso acessado. Os tipos de dados que o *webservice* consome e retorna é o JSON⁴. Não foi necessário fazer nenhuma implementação

⁴ JSON - Javascript Object Notation

adicional relativa a este formato, pois o próprio *framework Jersey* faz o tratamento e a conversão dos tipos de entrada e saída de dados. No caso do saída de dados, faz a conversão de objetos *Java* para JSON. E no caso de entrada tranforma um JSON em objeto *Java* já conhecido pelo *webservice*. Com isso concluiu-se o desenvolvimento do *webservice* que fornece os dados para o aplicativo.

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

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

1.4.3.3 Implantação

REFERÊNCIAS

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

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.

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