

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

APLICATIVO PARA CONSULTA DE NOTAS

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

2015

SUMÁRIO

1	QUADRO METODOLÓGICO	2
1.1	Tipo de pesquisa	2
1.2	Contexto de pesquisa	2
1.3	Instrumentos	3
1.4	Procedimentos e Resultados	4
1.4.1	Modelagem	4
1.4.2	Google Cloud Messaging	4
1.4.3	Aplicativo	5
1.4.4	Web service	5
REFERÊNCIAS.....		24

1 QUADRO METODOLÓGICO

Neste 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

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

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

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

1.2 Contexto de pesquisa

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

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

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

1.3 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 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 afirmaram 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.

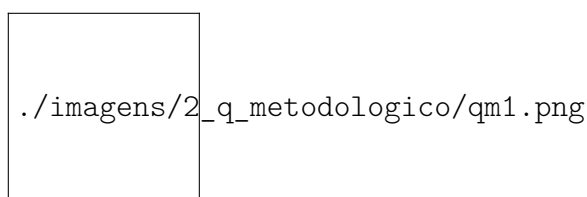


Figura 1 – Questionário Aplicado. **Fonte:**Elaborado pelos autores.

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

1.4 Procedimentos e Resultados

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

1.4.1 Modelagem

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

1.4.2 Google Cloud Messaging

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

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

Com a criação do projeto, a *Google* oferece um número que identificará o *software*, também chamado de *Sender ID*, conforme mostra a Figura 2.

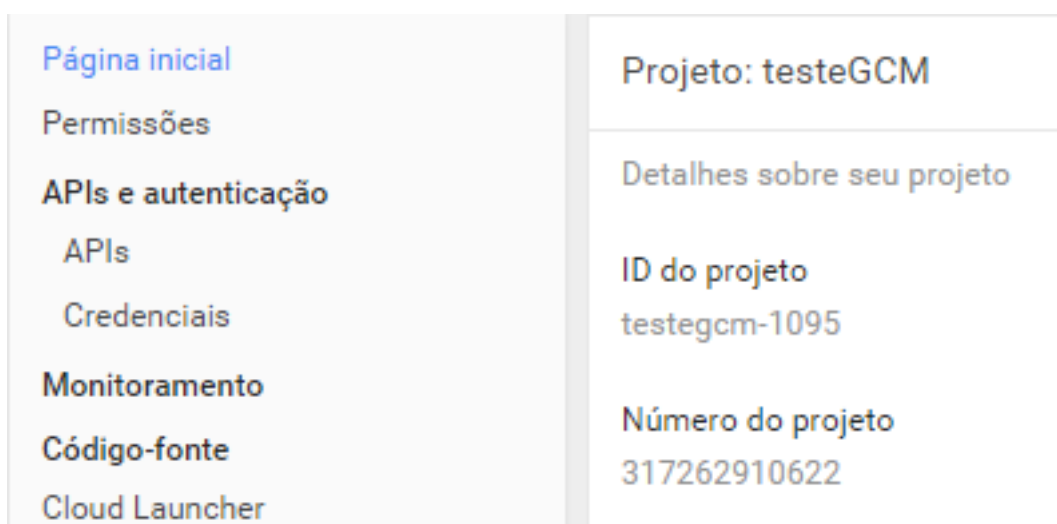


Figura 2 – *Sender ID* do GCM. **Fonte:**Elaborado pelos autores.

Por fim, acessou-se a aba Credenciais para indicar o IP do servidor. Ao informa-lo, o serviço gerou uma chave pública a qual foi inserida no *web service*. Na Figura 3, é possível ver o código de acesso criado.

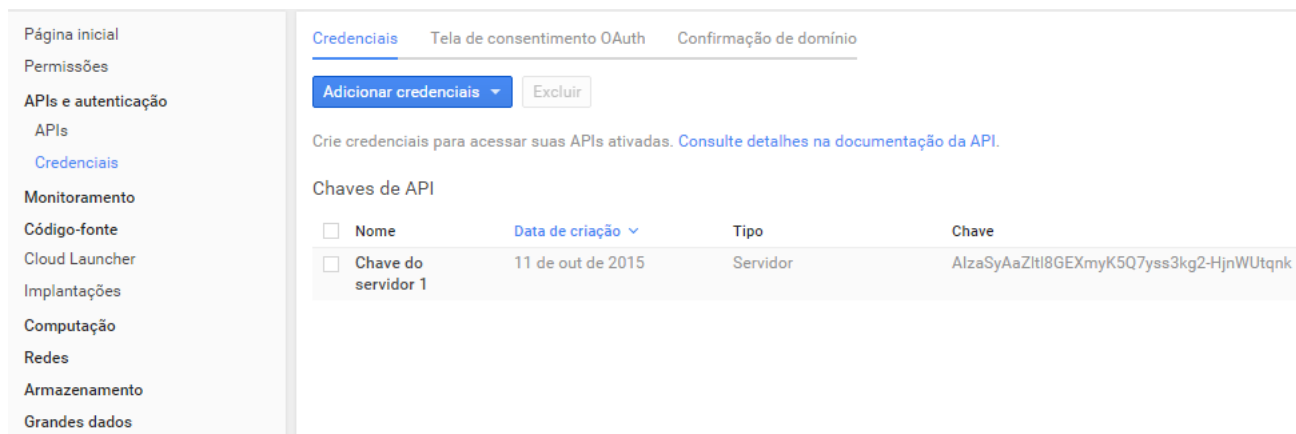


Figura 3 – Geração da credencial do GCM. **Fonte:**Elaborado pelos autores.

1.4.3 Aplicativo

1.4.4 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 configuração necessárias para a montagem do ambiente de desenvolvimento e sua posterior implantação.

1.4.4.1 Montagem do Ambiente de Desenvolvimento

No que diz respeito à contrução 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. 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 *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 do mesmo. A instalação consiste apenas em extrair os dados do arquivo em uma pasta da preferência do desenvolvedor. Esta abordagem permitiu a integração do Apache Tomcat com o 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 preterido 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.

1.4.4.2 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 gráfica de apoio para o PotgreSql. Para criar era necessário já estar com o PgAdmin aberto e conectado a um servidor de banco de dados que neste caso era em servidor local como pode ser visto na Figura 4.

¹ HTTP - Hypertext Transfer Protocol

² IDE - Integrated Development Environment

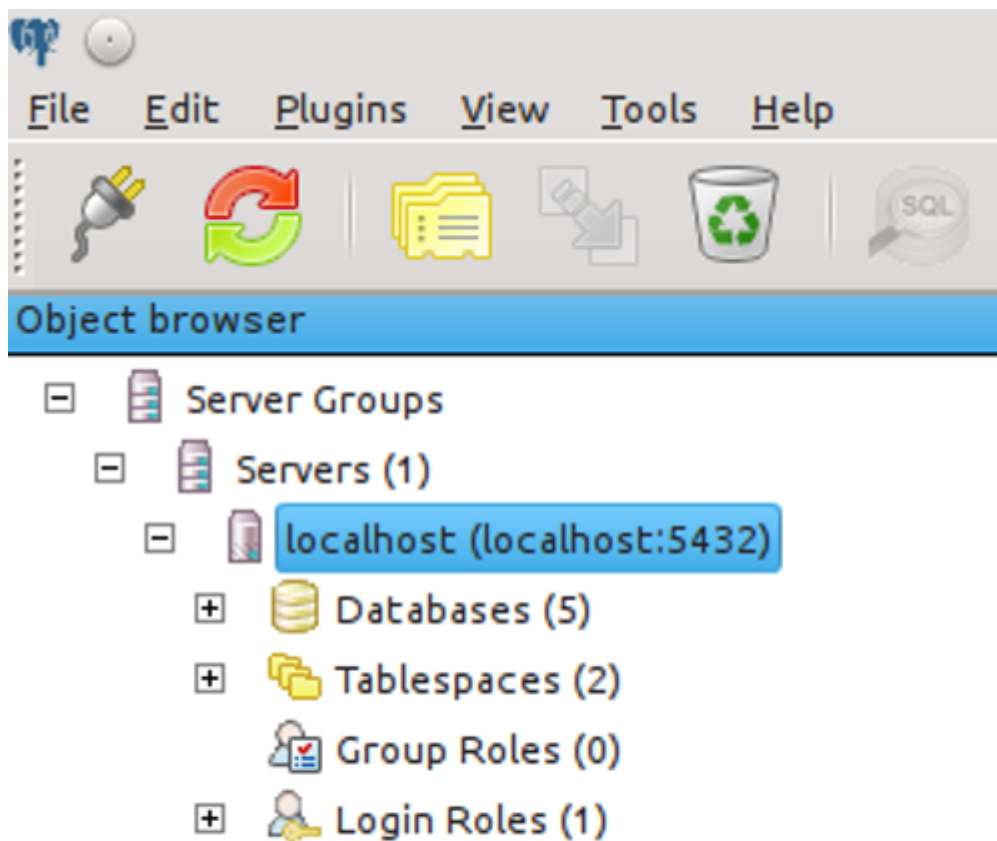


Figura 4 – Servidor de banco de dados local no PgAdmin. **Fonte:**Elaborado pelos autores.

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 5.

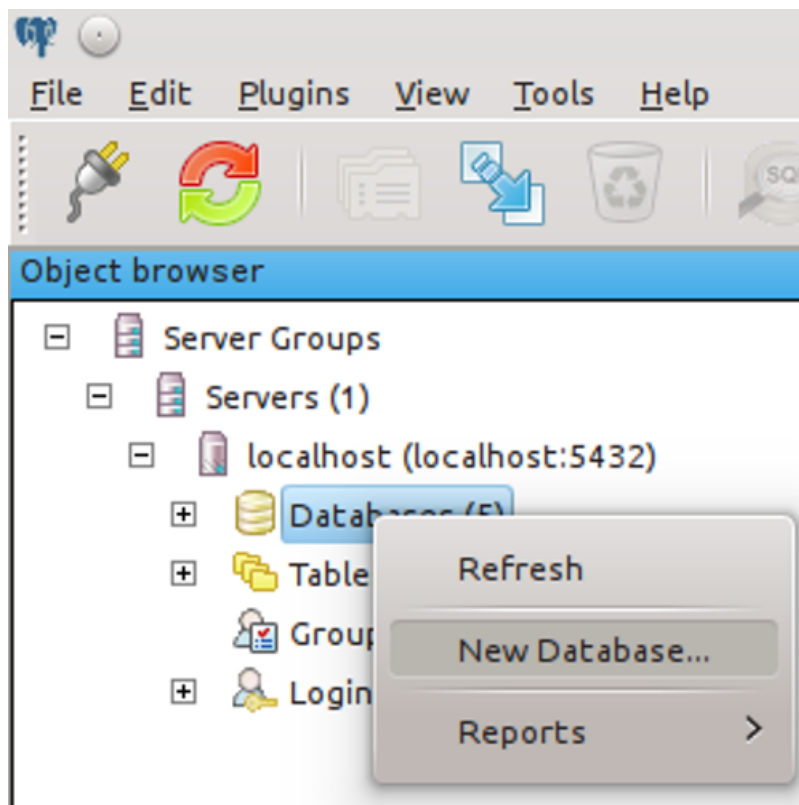


Figura 5 – Opção *New Database...* . **Fonte:**Elaborado pelos autores.

Em seguida foi necessário preencher o dados da janela apresentada, como está apresentado na Figura 6.

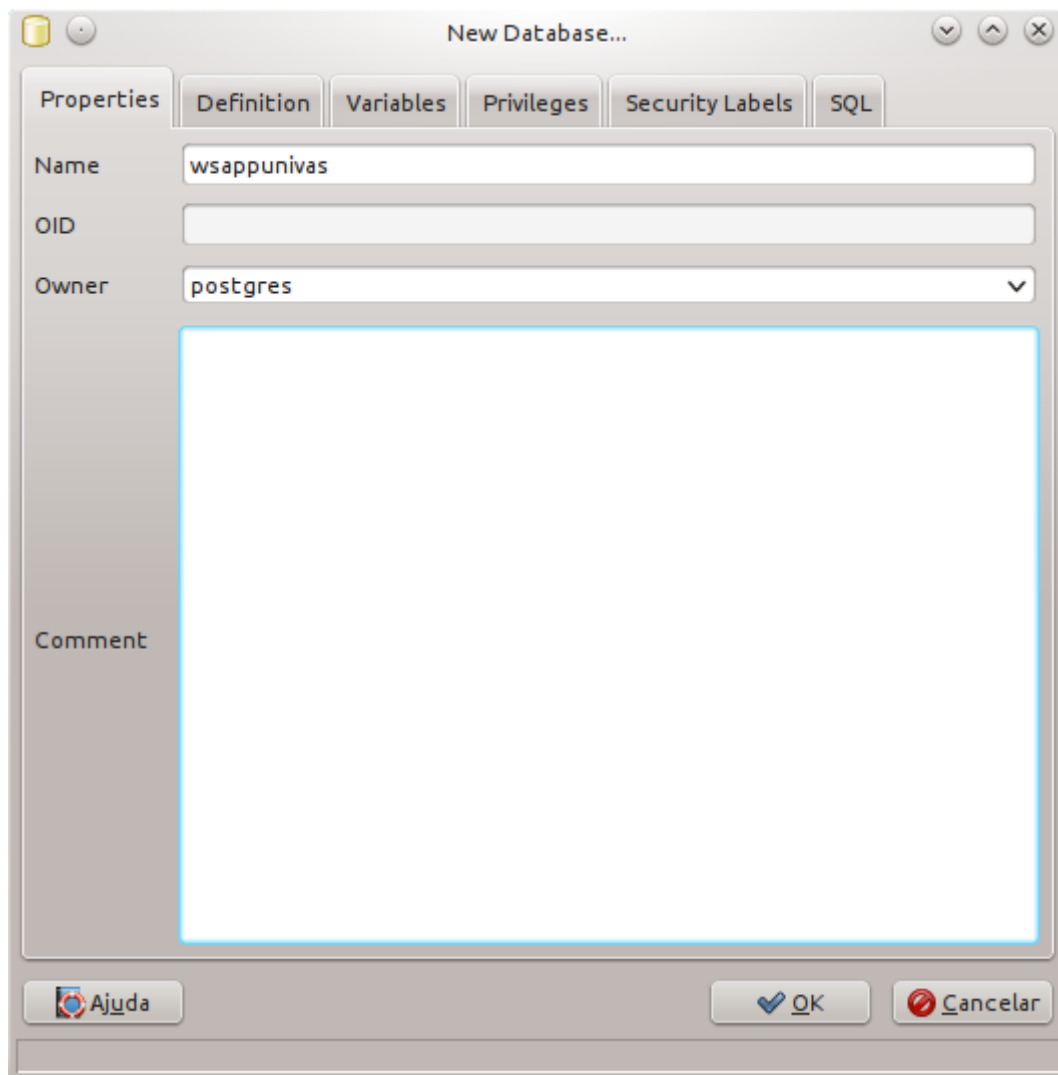


Figura 6 – Tela *New Database...* . . . **Fonte:**Elaborado pelos autores.

Como pode ser visto foram preenchidos os campos nome e usuário . O campo nome se refere ao nome do banco de dados que foi definido com *wsappunivas*, e usuário, o responsável pelo banco de dados, que para este caso foi usuário padrão do SGDB, que é o *postgres*. Além destas configurações mais nenhuma foi necessária. O banco de dados foi criado, porém sua estrutura não foi definida, pois como será visto mais adiante o Hibernate, possui um mecanismo, que com algumas configurações, permite a estruturação do banco de dados, de acordo com o mapeamento objeto-relacional 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.

Em seguida foi criado um projeto do tipo Dynamic Web Project no Eclipse. Para proceder com a criação de um novo projeto deste tipo no Eclipse, é necessário acessar na IDE, a opção **File -> New-> Dynamic Web Project** como pode ser visto na figura 7.

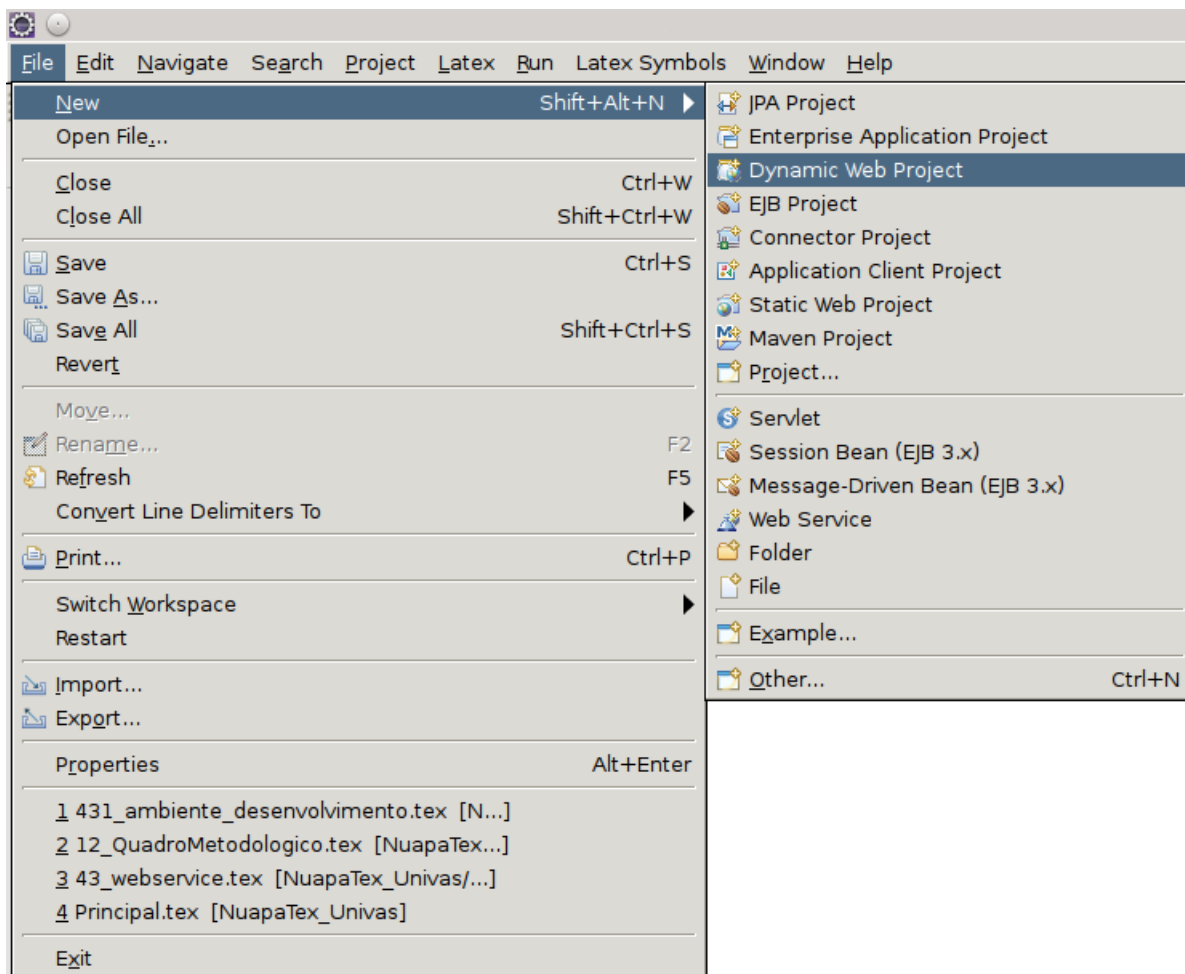


Figura 7 – Tela *New Database...* **Fonte:**Elaborado pelos autores.

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 8.

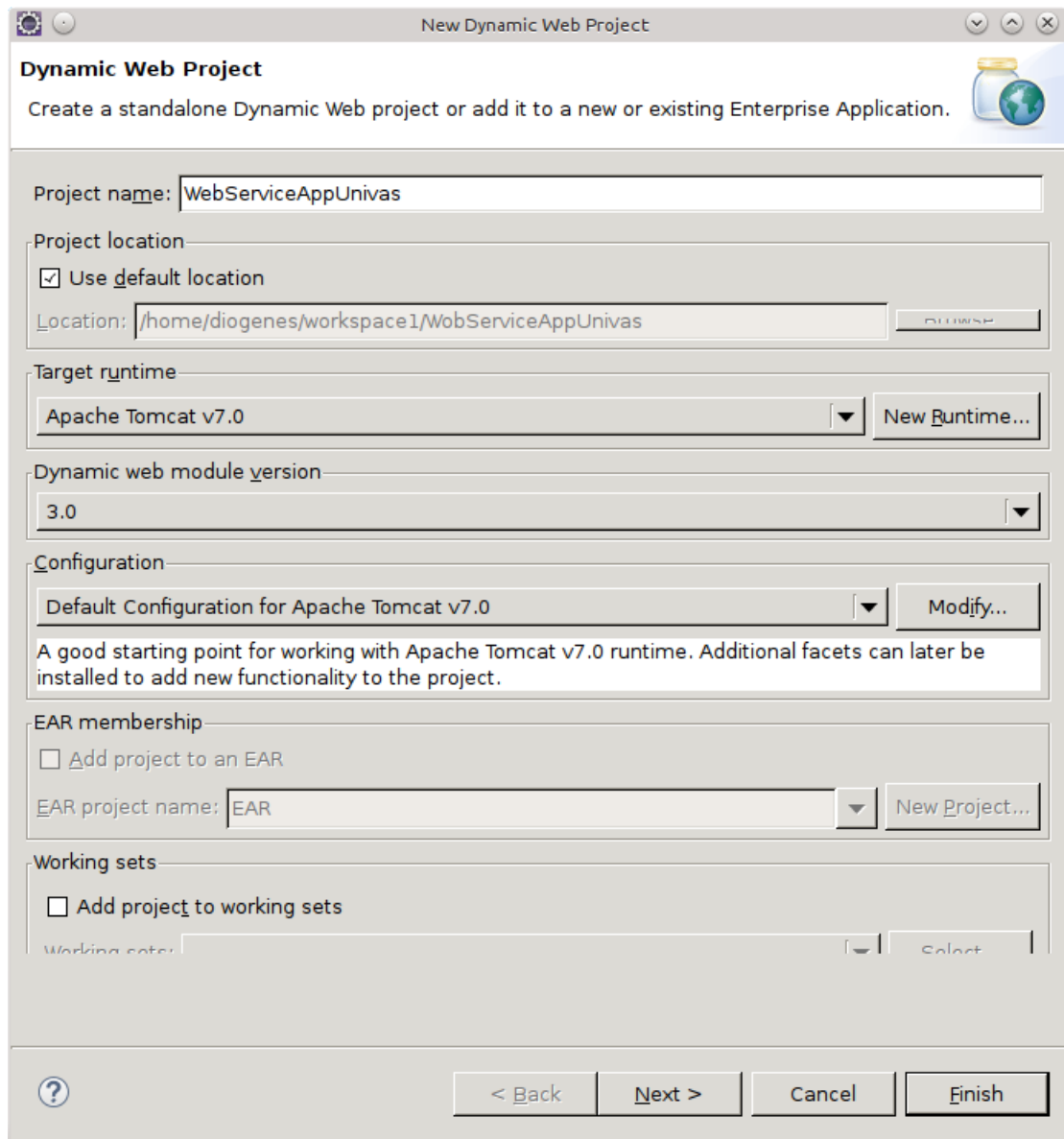


Figura 8 – Tela para criação de um novo projeto no Eclipse. **Fonte:**Elaborado pelos autores.

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 9.

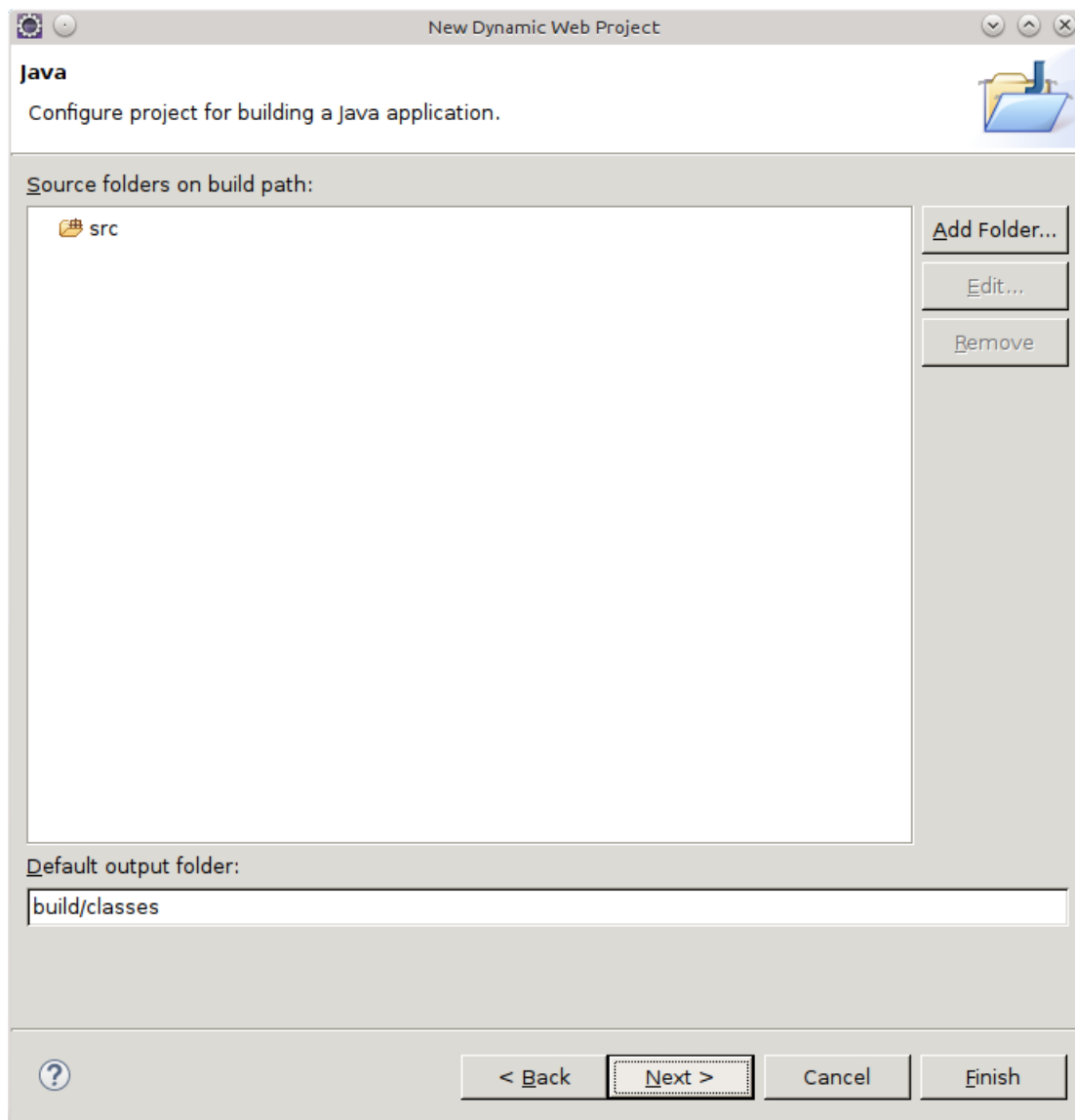


Figura 9 – Tela para criação de um novo projeto no Eclipse. **Fonte:**Elaborado pelos autores.

Na sequencia, 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 esta apresentada na Figura 10.

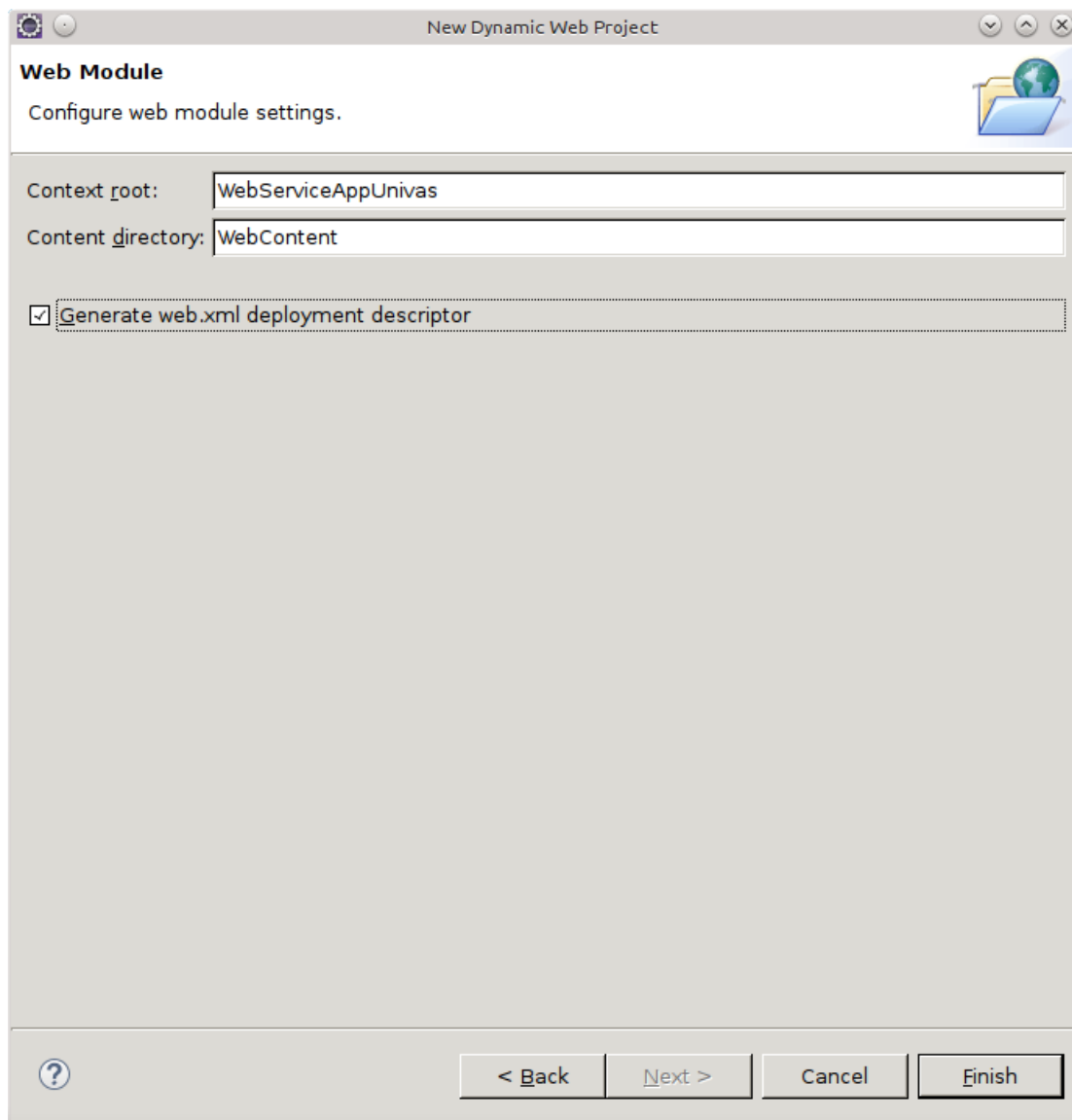


Figura 10 – Tela para criação de um novo projeto no Eclipse. **Fonte:**Elaborado pelos autores.

Após este passo foi concluído a criação do projeto, e já era possível iniciar os trabalhos com a camada de persistência de dados do projeto. Para este propósito, primeiramente foi criado um pacote, onde ficaram contidas as classes que representam as entidades do ORM. Para a criação do pacote foi necessário clicar com o botão direito do mouse sobre o projeto e acessar a opção **New -> Package**, como pode ser visto na Figura 11.

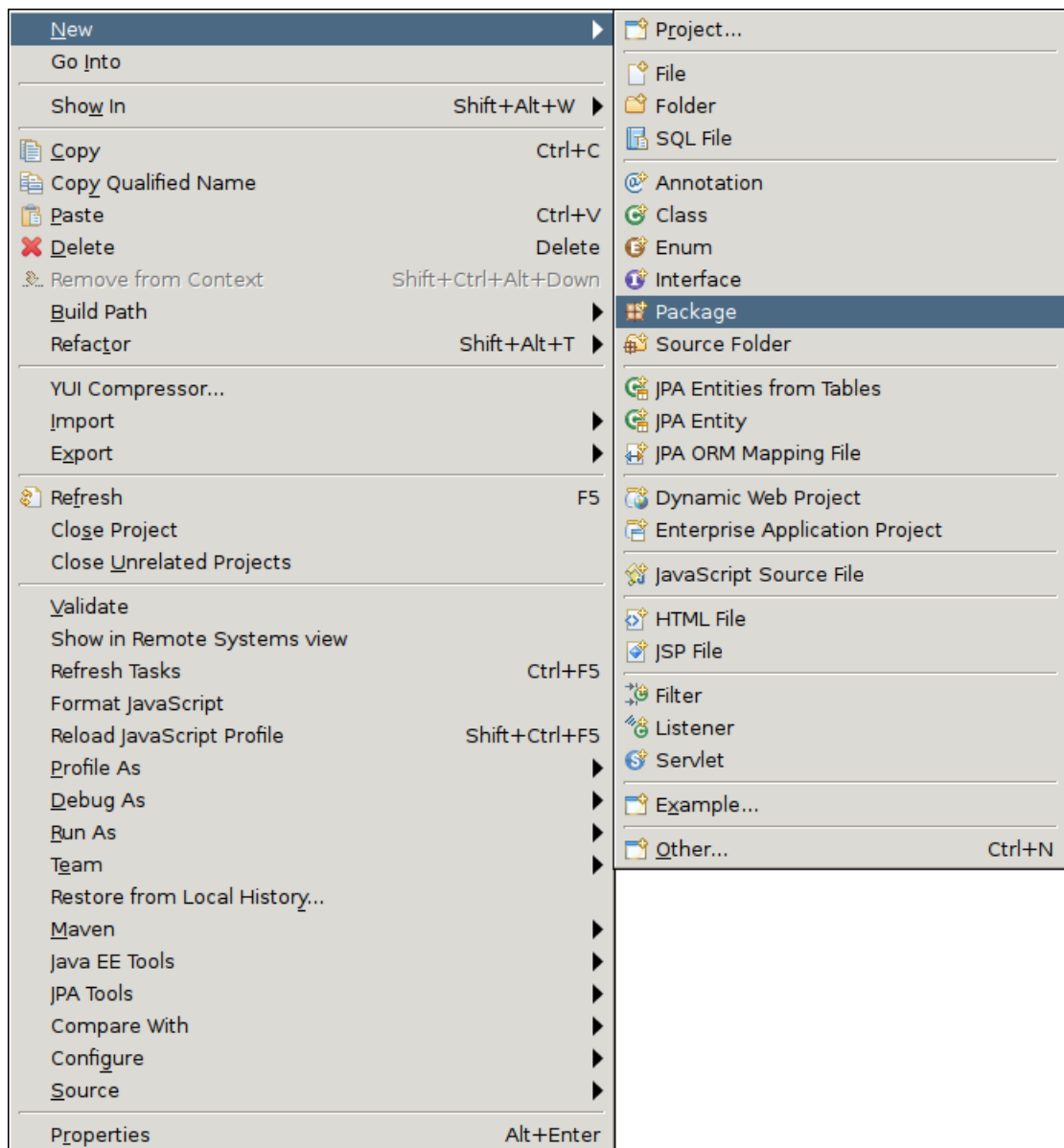


Figura 11 – Tela para criação de um novo projeto no Eclipse. **Fonte:**Elaborado pelos autores.

Em seguida foi apresentada a janela New Java Package, para a criação de um novo pacote mostrada na Figura 12. O pacote recebeu o nome de "br.edu.univas.restapiappunivas.model", pois nele estão contidas as classes que fazem parte do modelo de negócios da aplicação. Este pacote foi criado visando a divisão das responsabilidades internas no projeto, além de contribuir positivamente com a organização do mesmo.

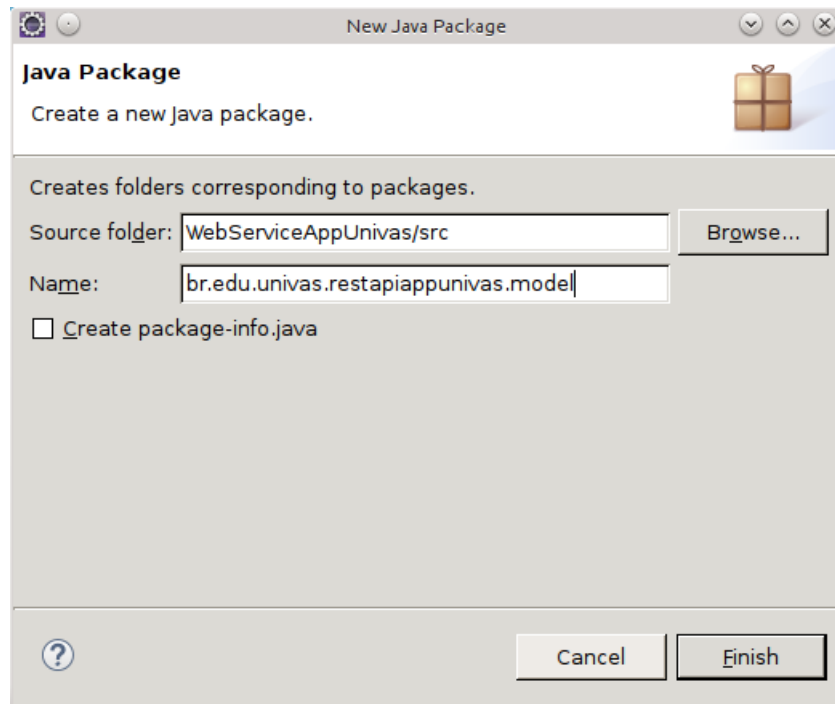


Figura 12 – Tela para criação de um novo projeto no Eclipse. **Fonte:**Elaborado pelos autores.

Com este pacote criado, já era possível criar as classes do ORM. Foi criada primeiramente a classe `Student.java`. Para a criação desta classe foi necessário clicar com o botão direito do *mouse* sobre o projeto e navegar até a opção **New -> Class** como pode ser visto na Figura 13.

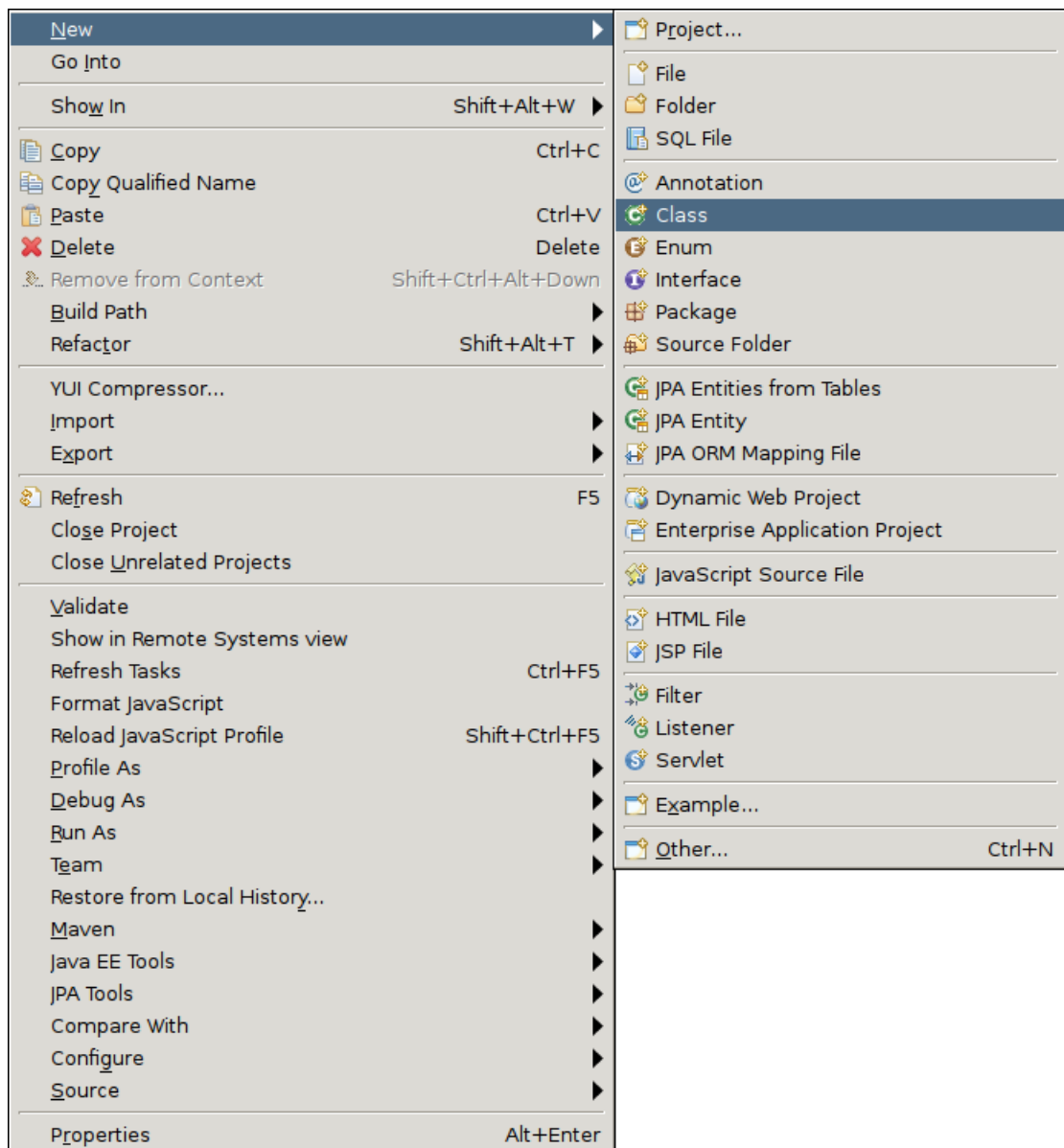


Figura 13 – Sem legenda. **Fonte:**Elaborado pelos autores.

Em seguida foi apresentada uma janela chamada New Java Class. Nesta janela somente foi necessário preencher o campo **Name:** que representa o nome da classe que está sendo criada.

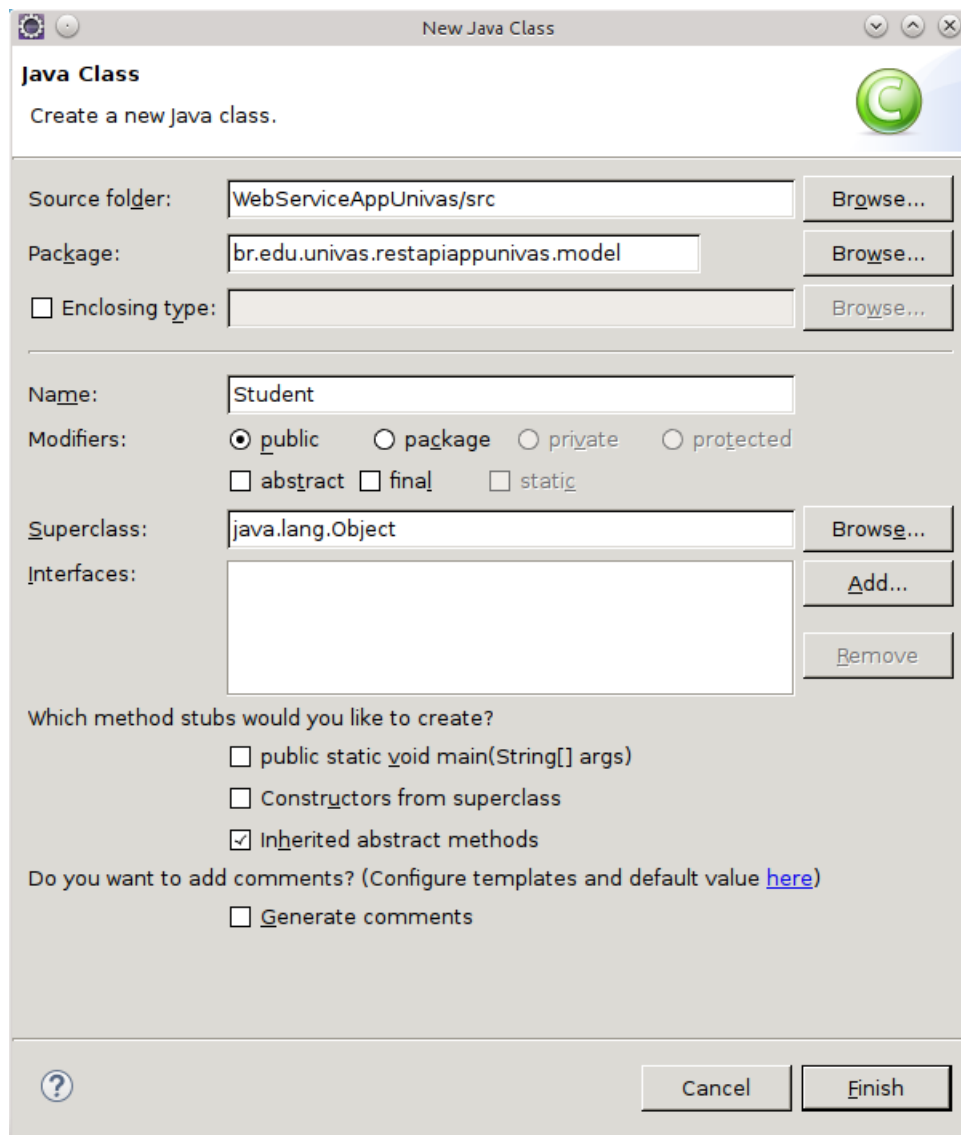


Figura 14 – Sem legenda. **Fonte:**Elaborado pelos autores.

Esta classe foi definida para representar as informações referente aos alunos. o código da classe pode ser visto na Figura 15.

```

1  package br.edu.univas.restapiapp.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 15 – Sem legenda. **Fonte:**Elaborado pelos autores.

É válido lembrar 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 específica. A seguir estão listadas todas as anotações da

classe `Student.java` e qual sua funcionalidade:

- `@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`: por padrão quando
- `@Id`
- `@SequenceGenerator`
- `@Column`
- `@OneToMany`
- `OneToMany`

Com isso esta classe já poderia ser entendida como uma entidade e poderia ser persistida no banco de dados, porém além dessa anotação outras foram usadas para que a persistência pudesse ocorrer de forma consistente.

Estas 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 pojos 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.

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

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

³ JPA - Java Persistence API

Em seguida à criação das entidades, foi necessário configurar o arquivo `persistence.xml` que fica dentro do *classpath* do projeto Java ou seja, dentro da mesma pasta onde estão contidos pacotes do projeto. Este arquivo é extremamente importante, pois é nele que estão todas as configurações relativas à conexão com o banco de dados, configurações referentes ao Dialeto SQL que vai ser usado para as consultas e configurações referentes ao *persistence unit* que é o conjunto de classes mapeadas para o banco de dados. O arquivo `persistence.xml` está exposto na Figura 16.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="2.1"
3   xmlns="http://xmlns.jcp.org/xml/ns/persistence"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
6     http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
7   <persistence-unit name="WsAppUnivas" transaction-type="
8     RESOURCE_LOCAL">
9     <provider>
10      org.hibernate.jpa.HibernatePersistenceProvider
11    </provider>
12    <properties>
13      <property name="javax.persistence.jdbc.url"
14        value="jdbc:postgresql://localhost:5432/wsappunivas"
15      />
16      <property name="javax.persistence.jdbc.user"
17        value="postgres" />
18      <property name="javax.persistence.jdbc.password"
19        value="omitido" />
20      <property name="javax.persistence.jdbc.driver"
21        value="org.postgresql.Driver" />
22      <property name="hibernate.dialect"
23        value="org.hibernate.dialect.PostgreSQLDialect" />
24      <property name="hibernate.format_sql"
25        value="true" />
26      <property name="hibernate.temp.
27        use_jdbc_metadata_defaults"
28        value="false" />
29      <property name="hibernate.show_sql"
30        value="true" />
31      <property name="hibernate.hbm2ddl.auto"
32        value="create" />
33    </properties>
34  </persistence-unit>
35 </persistence>
```

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

Em seguida à confecção do `persistence.xml` foi criada a classe `JpaUtil` que está representada na Figura 17. Esta classe é responsável por criar uma `EntityManagerFactory` que é uma fábrica de instâncias de `EntityManager` que nada mais é que um *persistence unit* ou unidade de persistência. Essa classe tem a responsabilidade de prover um modo de comunicação entre a aplicação e o banco de dados. No entanto a classe `JpaUtil` cria uma única instância

de `EntityManagerFactory`, que é responsável por disponibilizar e gerenciar as instâncias de `EntityManager` de acordo com a necessidade da aplicação.

```

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

```

Figura 17 – Classe JpaUtil.java. **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 *web service* para o aplicativo Android. Essa classe e seus respectivos métodos estão representada na Figura .

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 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 *web service*. Com isso concluiu-se o desenvolvimento do *web service* 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*.

⁴ JSON - Javascript Object Notation

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.

REFERÊNCIAS

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

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.

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.