

PROJETO WEB SERVICES RESTfull

Prof. Jeangrei Veiga

Sumário

REQUISITOS	2
CRIANDO O BANCO DE DADOS E O PROJETO WEB	3
1. Criar banco de dados "DB_AULAWEBSERVICES" no postgres.	3
CRIAR O PROJETO JAVA WEB	4
ESTRUTURANDO O PROJETO JAVA WEB	6
1. Criando as classes de entidade automatizadas com base no banco de dados:.....	6
2. Criando as páginas JSF de forma automatizada, utilizando o framework PrimeFaces, com base nas classes de entidade de banco de dados:.....	8
3. Testando e ajustando os formulários:	10
CRIANDO A CLASSE “NOTASSERVICE” WEB SERVICES – RESTfull	13
1. Criando a classe que vai representar WS de tbNotas.....	13
2. Configurando a classe... ..	13
3. Configurando classe WS para trabalhar com persistência de dados utilizando JPA.....	14
4. Adicionando primeiro serviço de busca (GET) RESTFull.....	14
CONFIGURANDO WEB SERVICES – RESTfull	15
PRIMEIRO TESTE DO WEB SERVICES	16
INCLUÍDO NOVOS SERVIÇOS GET	17
1. Método GET: contar a quantidade de registros que constam cadastrados na base de dados.	17
3. Método GET: passando parâmetro para buscar um registro no banco de dados, passando {id}... ..	18
ADICIONANDO UM NOVO SERVIÇO GET PARA BUSCAR UMA NOTA UTILIZANDO PARTE DO TÍTULO INFORMADO.	19
1. Criando SQL de consulta e testando no PGAdmin.	19
2. Criando a NativeQuery na de entidade.....	19
3. Na classe Facade, criar o método para consumir a NativeQuery.	20
4. Na classe de WebService, criar o novo serviço que busca um registro utilizando parte do título.....	20
a. Criar objeto EJB (Enterprise Java Bean) utilizado para consumir o serviço EJB da facade.	20
b. Criar serviço para buscar nota utilizando parte do título	21
c. Testando serviço	21
ADICIONANDO SERVIÇOS POST – PUT – DELETE	22

1. POST para inserir um registro na base de dados.	22
2. PUT para atualizar um registro na base de dados.	23
3. DELETE para deletar um registro na base de dados.	24

REQUISITOS

As ferramentas abaixo devem ser instaladas para uso do projeto.

- NetBeans 8.2
Utilizado para codificar o projeto
- ServerWeb Glassfish 4.0
Utilizado para publicar o projeto
- PostgreSQL
Utilizado para manipular os dados.
- Postman para Google Chrome
Utilizado para testar o consumo dos serviços Web Services.

CRIANDO O BANCO DE DADOS E O PROJETO WEB

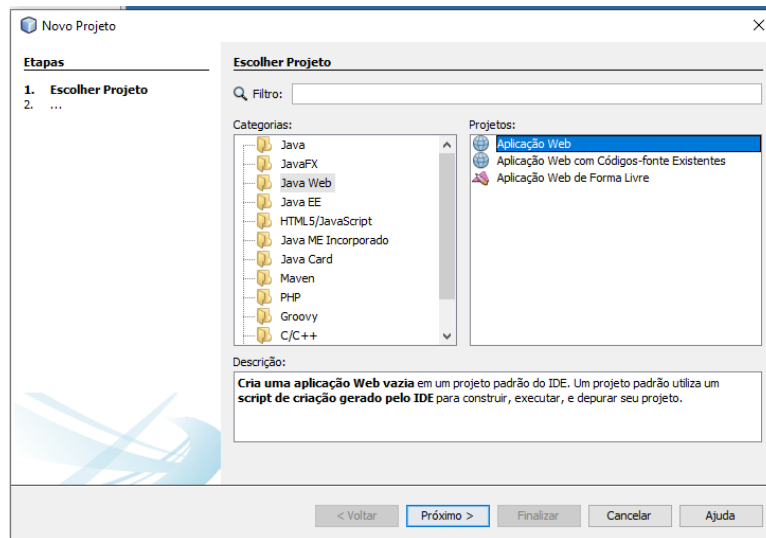
1. Criar banco de dados "DB_AULAWEBSERVICES" no postgres.

```
--Data Base Name: DB_AULAWEBSERVICES
Create table tb_nota
(
    id_note Serial NOT NULL,
    titulo Varchar(100) NOT NULL,
    descricao Varchar(250) NOT NULL,
    constraint pk_nota primary key (id_note)
);
```

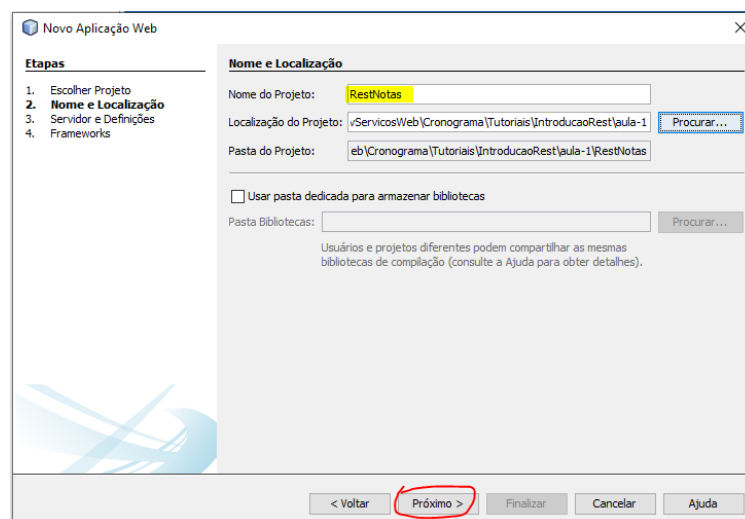
CRIAR O PROJETO JAVA WEB

Criar o projeto Java Web "RestNotas" utilizando GlassFish, e:

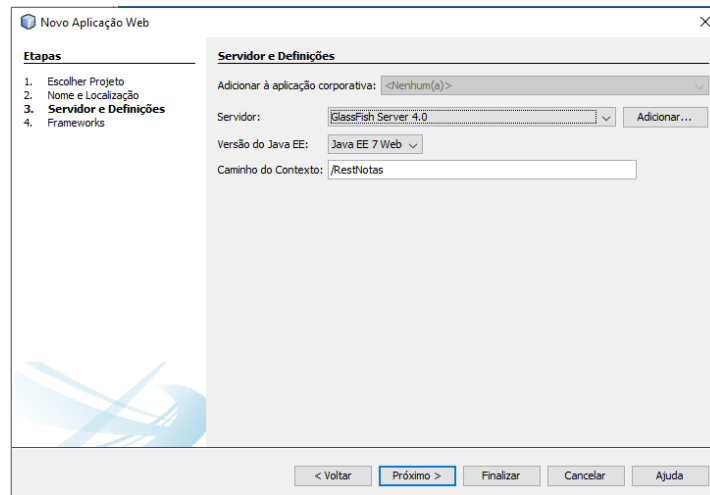
- Em framework, selecionar JavaServer Faces
- Em componentes, selecionar PrimeFaces



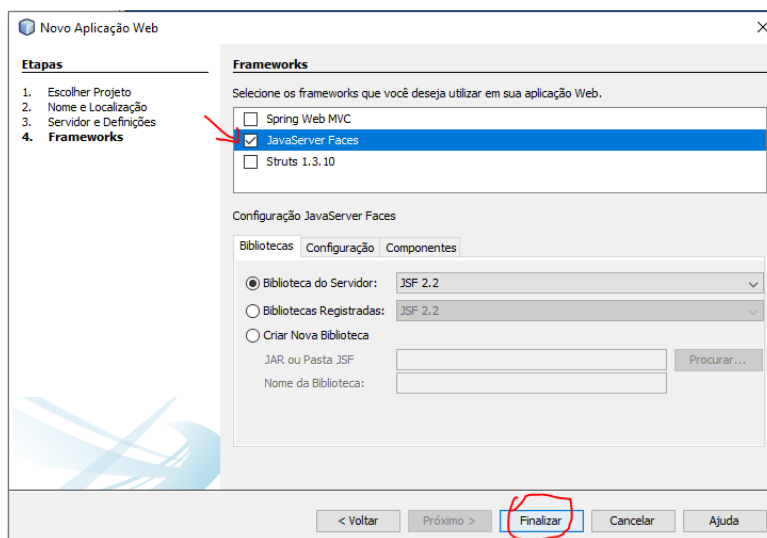
Criando o projeto Java Web



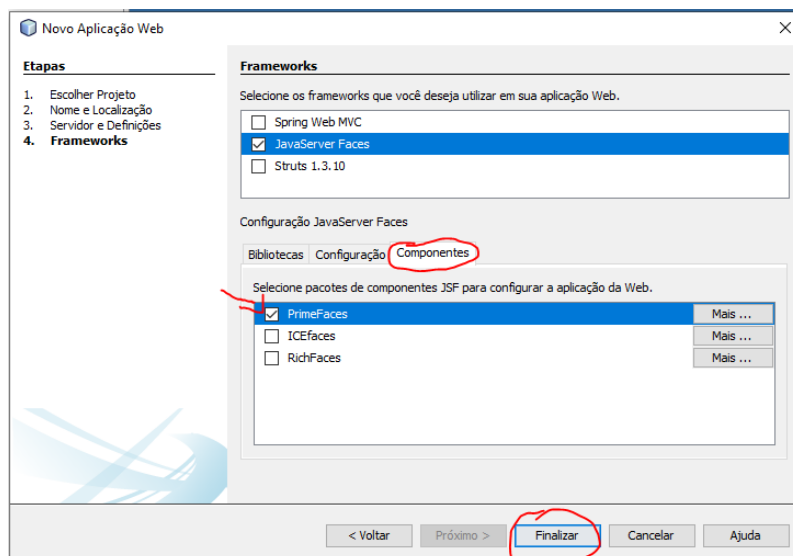
Definindo o nome da aplicação "RestNotas"



Definindo servidor "Galssfish 4.0"



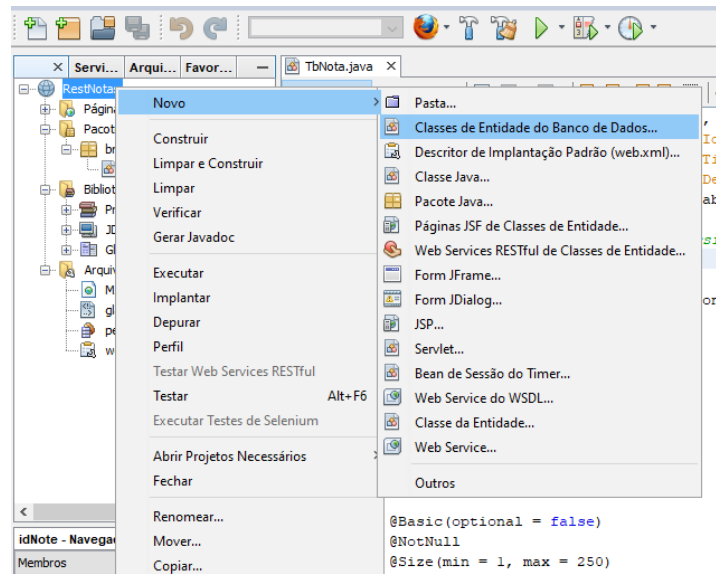
Definindo framework de desenvolvimento Java Web



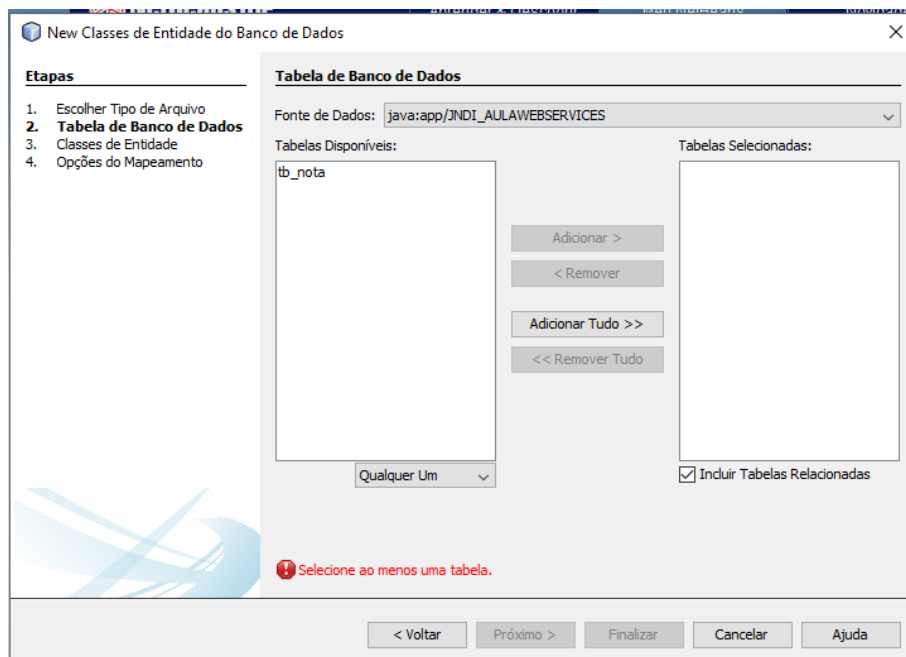
Definindo componente PrimeFaces.

ESTRUTURANDO O PROJETO JAVA WEB

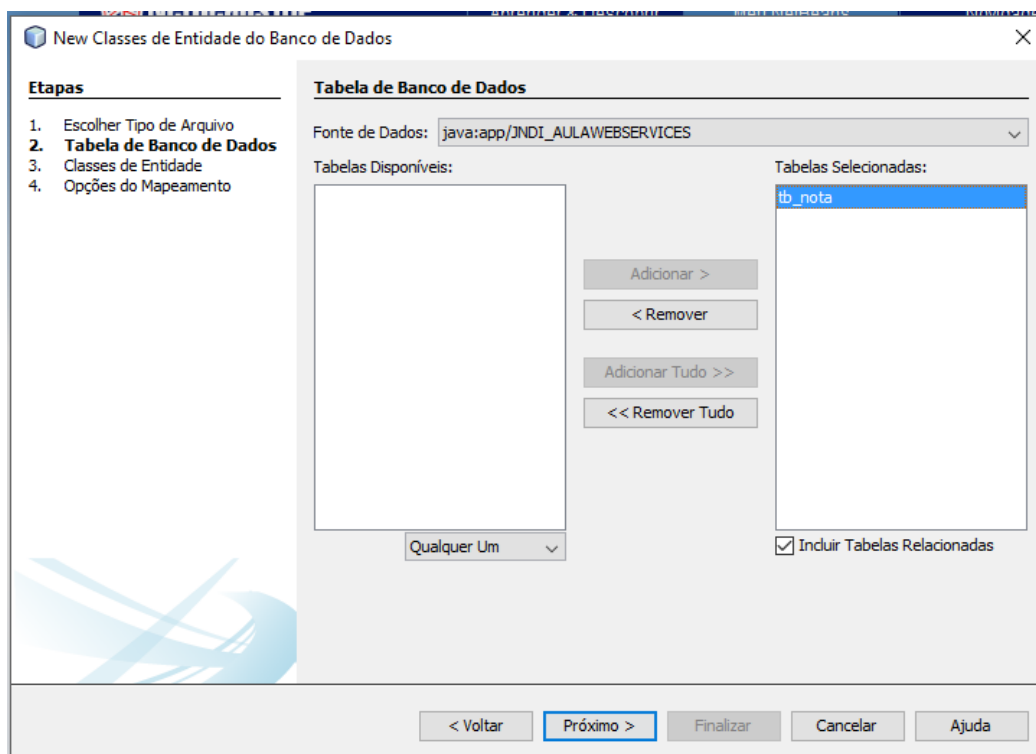
1. Criando as classes de entidade automatizadas com base no banco de dados:
 - a. Iniciando processo...



- b. Criar JNDI de conexão com banco de dados "JNDI_AULAWEBSERVICES".

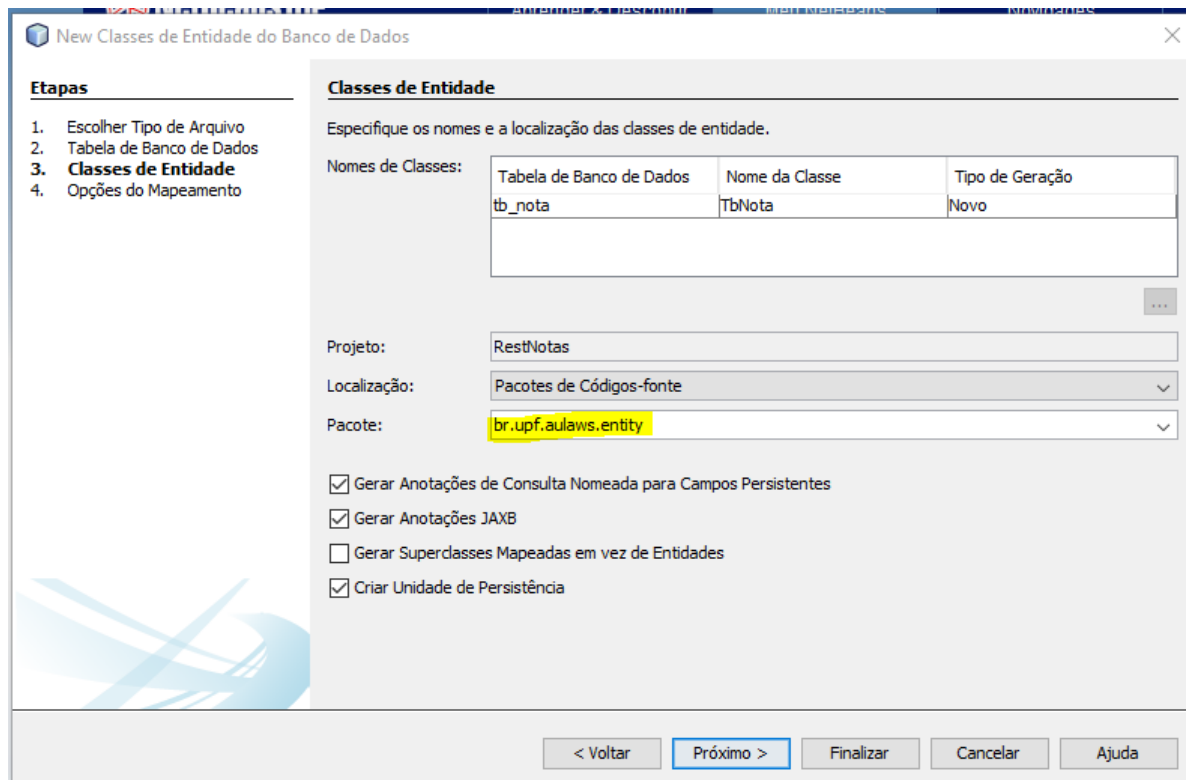


- c. Definindo as tabelas que serão utilizadas para criação das classes de entidade de banco de dado.

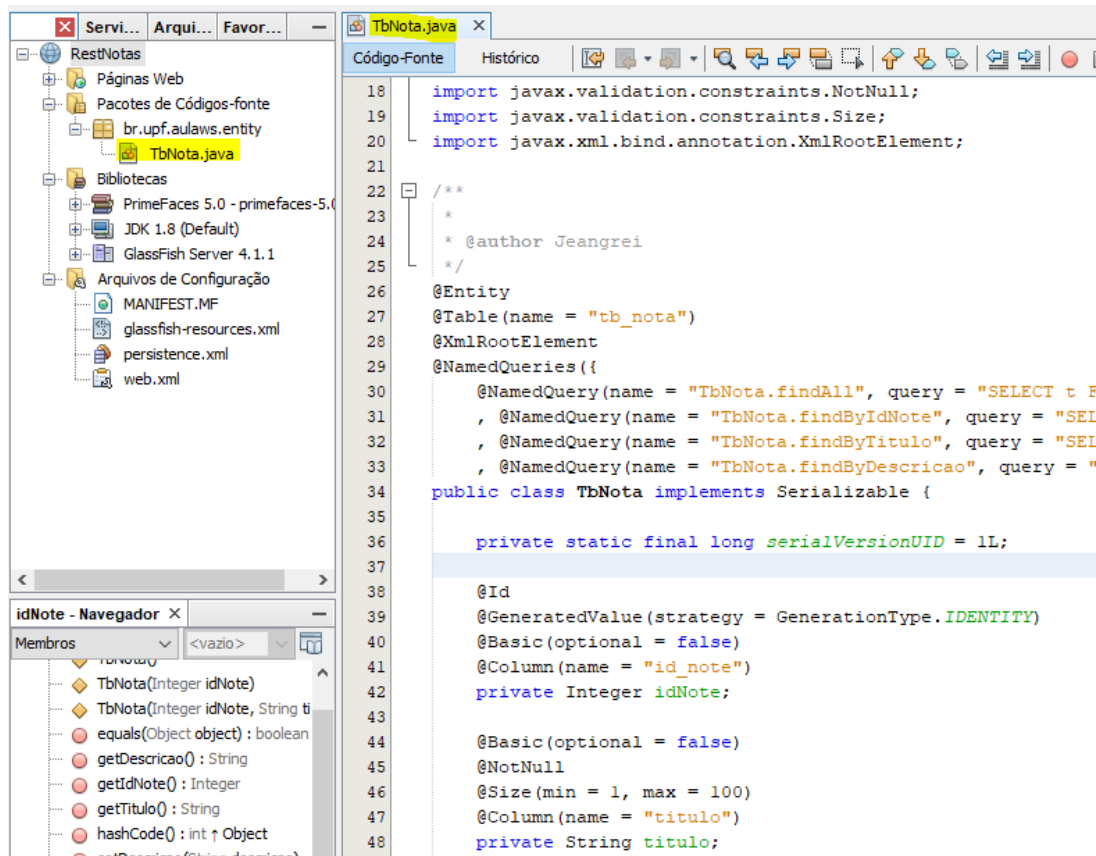


Obs.: Neste momento serão criadas as classes, onde será utilizado o framework JPA “EclipseLink” para mapear as tabelas do banco de dados.

- d. Definindo as características das classes de entidade

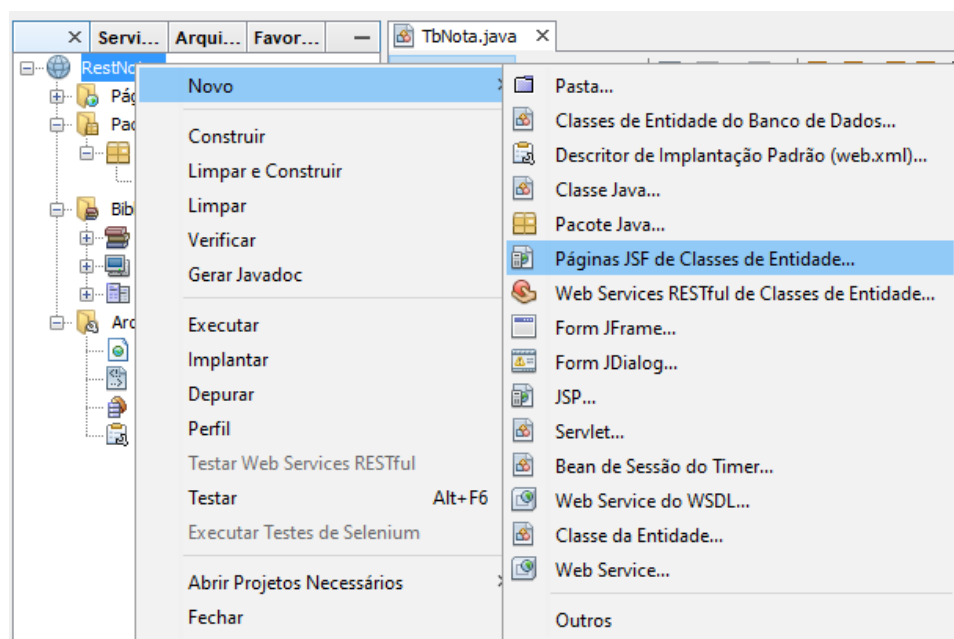


e. Classe de entidade criada:

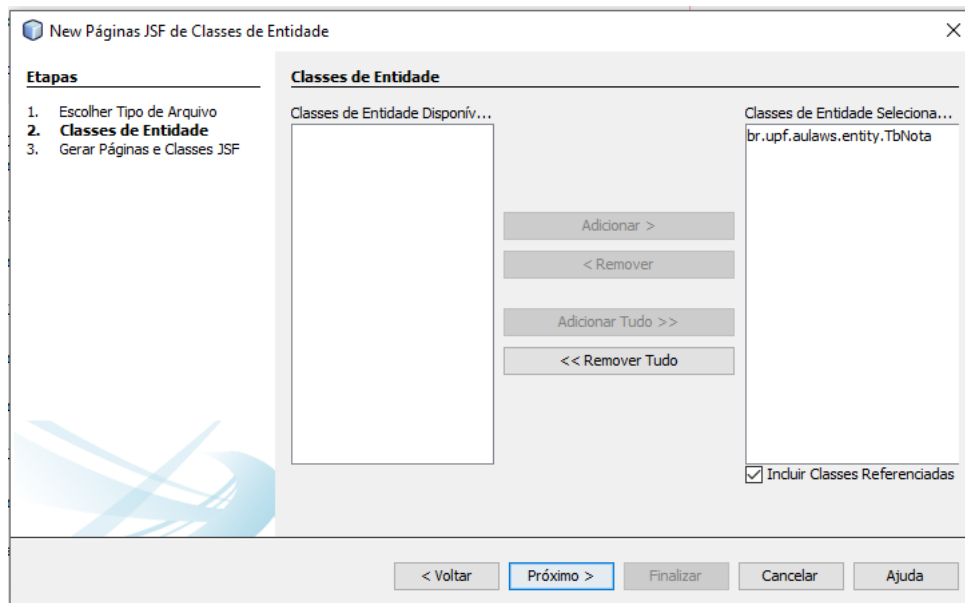


2. Criando as páginas JSF de forma automatizada, utilizando o framework PrimeFaces, com base nas classes de entidade de banco de dados:

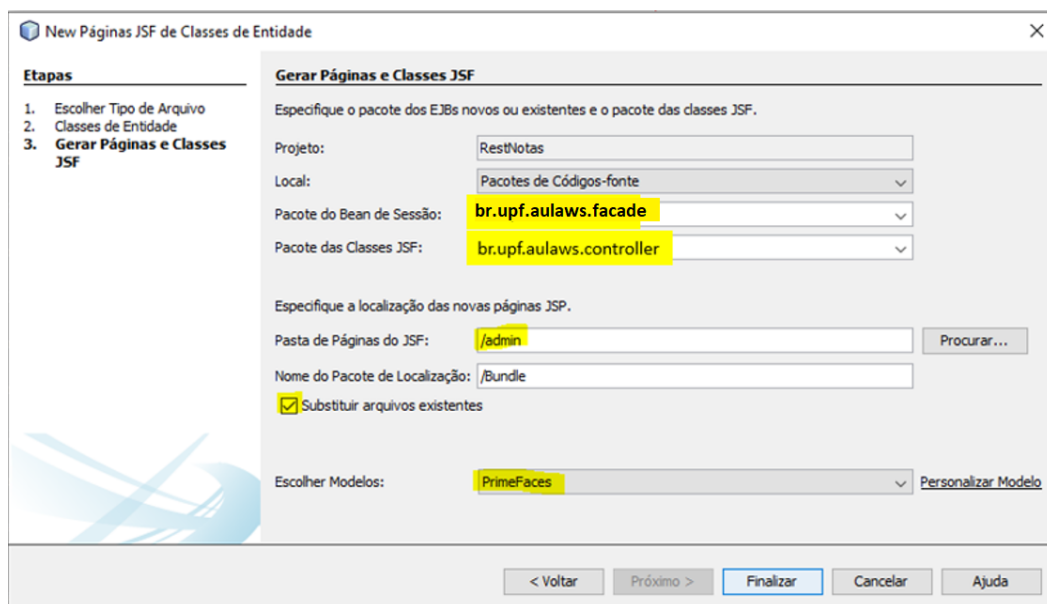
a. Iniciando processo...



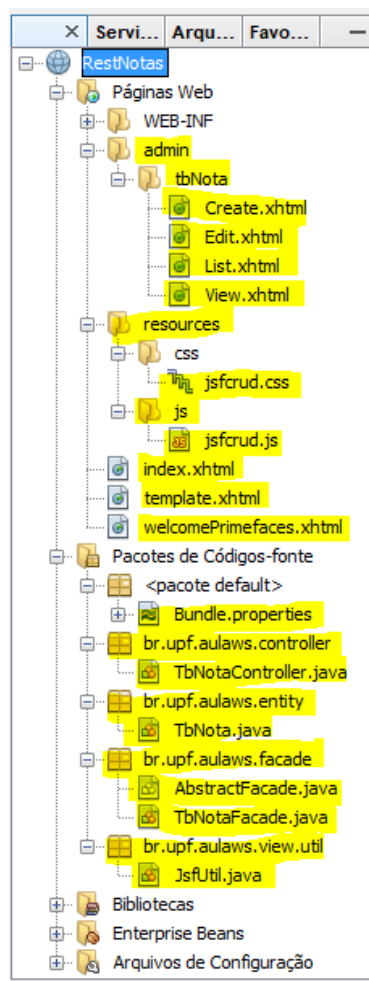
- b. Definindo a construção automatizada das páginas com base nas classes de entidade criadas no item anterior...



- c. Definindo as características para criação das páginas JSF



d. Estrutura criada de forma automática até o momento...



3. Testando e ajustando os formulários:

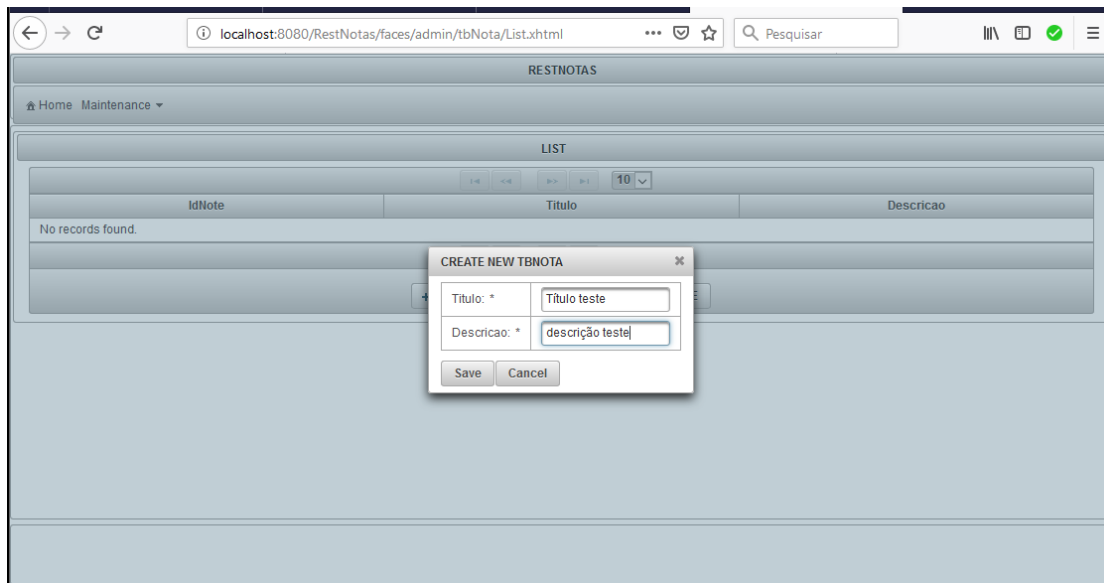
a. Removendo campo id para uso do auto incremento do banco...

```
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

<ui:composition>
  <p:dialog id="TbNotaCreateDlg" widgetVar="TbNotaCreateDialog" modal="true" resizable="false" appendTo="
  <h:form id="TbNotaCreateForm">
    <h:panelGroup id="display">
      <p:panelGrid columns="2" rendered="#{tbNotaController.selected != null}">
        <p:outputLabel value="#{bundle.CreateTbNotaLabel_titulo}" for="titulo" />
        <p:inputText id="titulo" value="#{tbNotaController.selected.titulo}"
          title="#{bundle.CreateTbNotaTitle_titulo}" required="true"
          requiredMessage="#{bundle.CreateTbNotaRequiredMessage_titulo}" />
        <p:outputLabel value="#{bundle.CreateTbNotaLabel_descricao}" for="descricao" />
        <p:inputText id="descricao" value="#{tbNotaController.selected.descricao}"
          title="#{bundle.CreateTbNotaTitle_descricao}" required="true"
          requiredMessage="#{bundle.CreateTbNotaRequiredMessage_descricao}" />
      </p:panelGrid>
      <p:commandButton actionListener="#{tbNotaController.create}"
        value="#{bundle.Save}" update="display,TbNotaListForm:datalist,:growl"
        oncomplete="handleSubmit(args,'TbNotaCreateDialog');" />
      <p:commandButton value="#{bundle.Cancel}" onclick="TbNotaCreateDialog.hide()" />
    </h:panelGroup>
  </h:form>
</p:dialog>
</ui:composition>
</html>
```

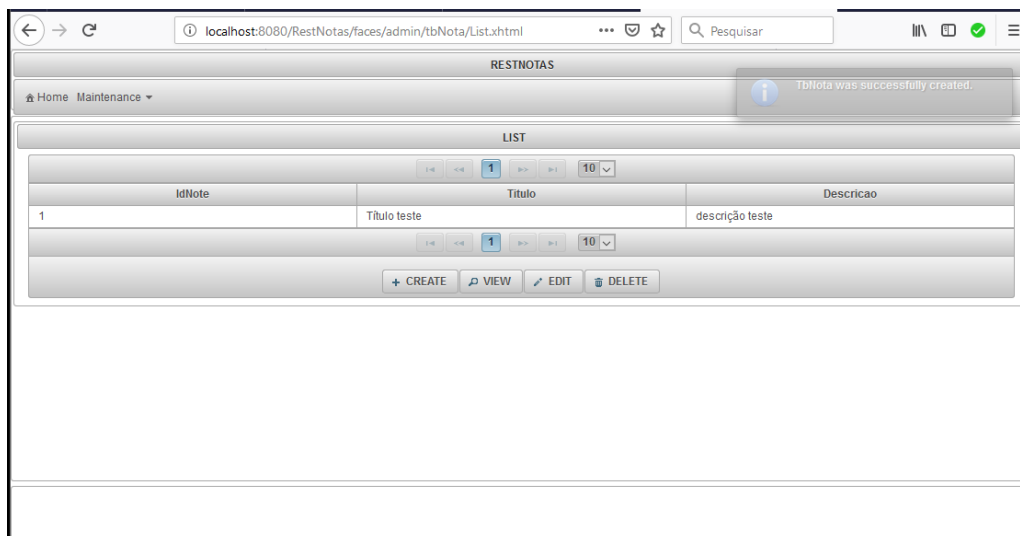
b. Testando o projeto Java Web:

- i. Executar o servidor Glassfish no Netbeans.
- ii. Criando registro...

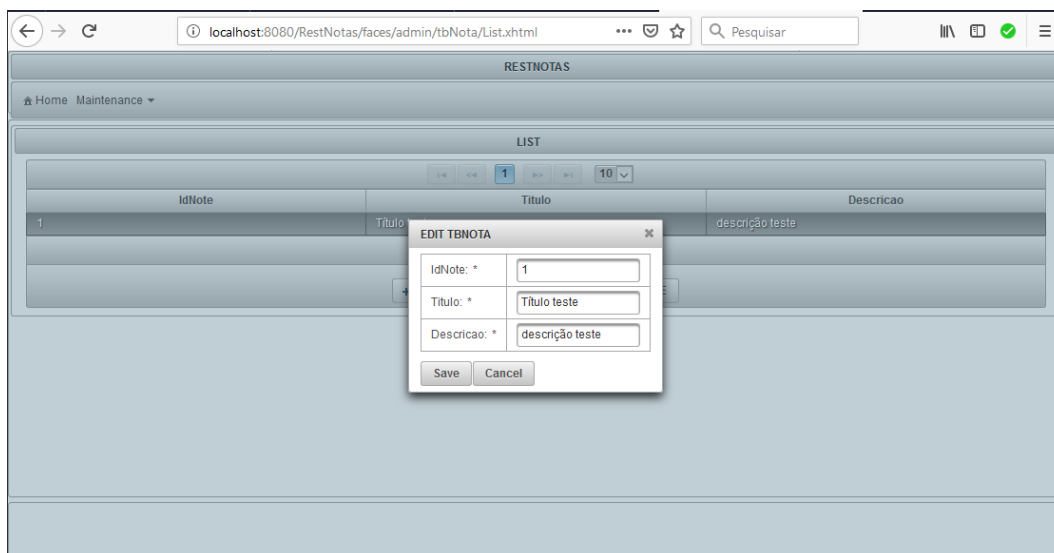


Obs.: Se tudo correr sem erro, o projeto será aberto no navegador de internet. Senão, deverá ser verificado o log do servidor no Netbeans.

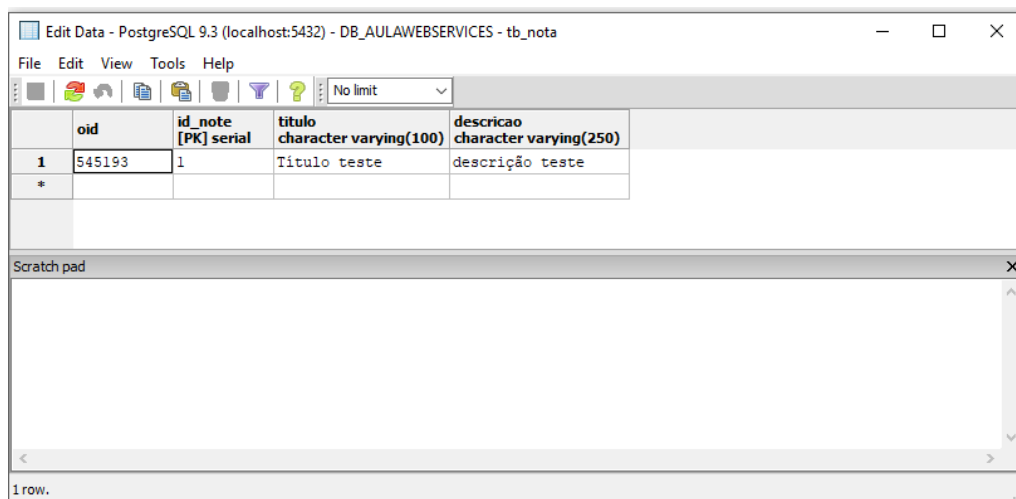
iii. Listando registro...



iv. Editando registro...



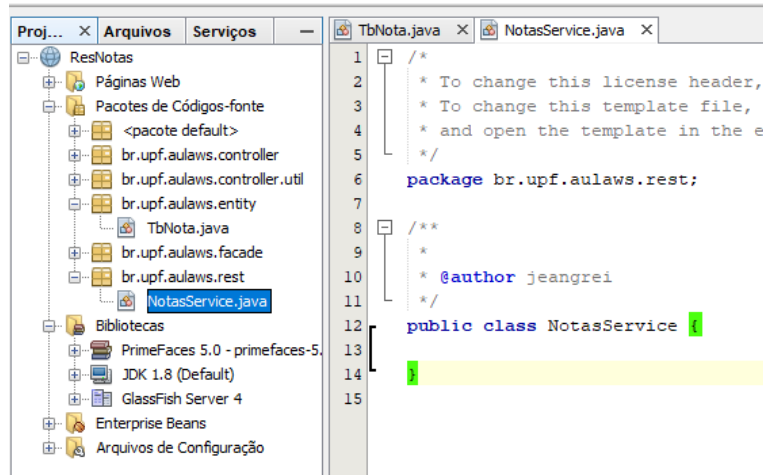
c. Confirmando registro adicionado no banco de dados...



CRIANDO A CLASSE “NOTASSERVICE” WEB SERVICES – RESTfull

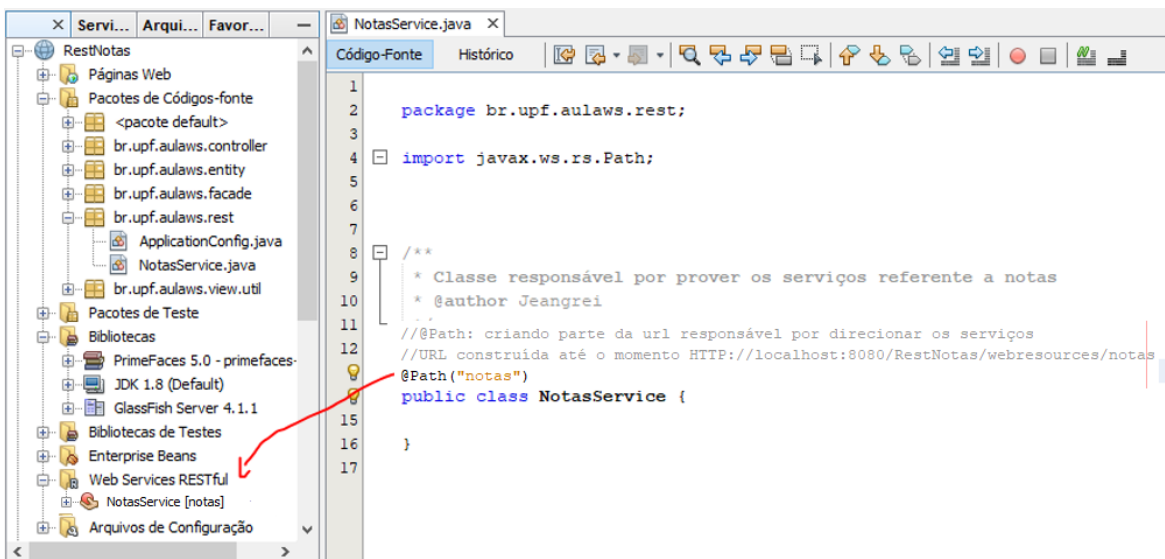
Implementando a classe que será fornecedora de serviços RESTFull:

1. Criando a classe que vai representar WS de tbNotas...



2. Configurando a classe...

Após incluir a notação “@Path(*notas*)”, o servidor GlassFish identifica a classe como sendo um WS e registra a classe e, Web Services RESTful...



3. Configurando classe WS para trabalhar com persistência de dados utilizando JPA...

The screenshot shows an IDE with the following components:

- Proj... (Project Explorer):** Shows the project structure. The 'RestNotas' package contains 'Páginas Web', 'Pacotes de Códigos-fonte', and 'br.upf.aulaws'. The 'br.upf.aulaws' package contains 'controller', 'entity', 'facade', 'rest', and 'view.util'. The 'entity' package contains 'TbNota.java'. The 'facade' package contains 'AbstractFacade.java' and 'TbNotaFacade.java'. The 'rest' package contains 'ApplicationConfig.java' and 'NotasService.java'.
- Código-Fonte (Source Editor):** Shows the source code of 'NotasService.java'. The code includes annotations for JPA and REST, and implements the 'AbstractFacade' interface. Key annotations include `@Stateless`, `@Path("notas")`, and `@PersistenceContext(unitName = "RestNotasPU")`. The class implements the `getEntityManager()` method.
- Navegador (Navigator):** Shows the 'NotasService' class and its methods: `NotasService()`, `getEntityManager() : EntityManager`, and `em : EntityManager`.

4. Adicionando primeiro serviço de busca (GET) RESTFull...

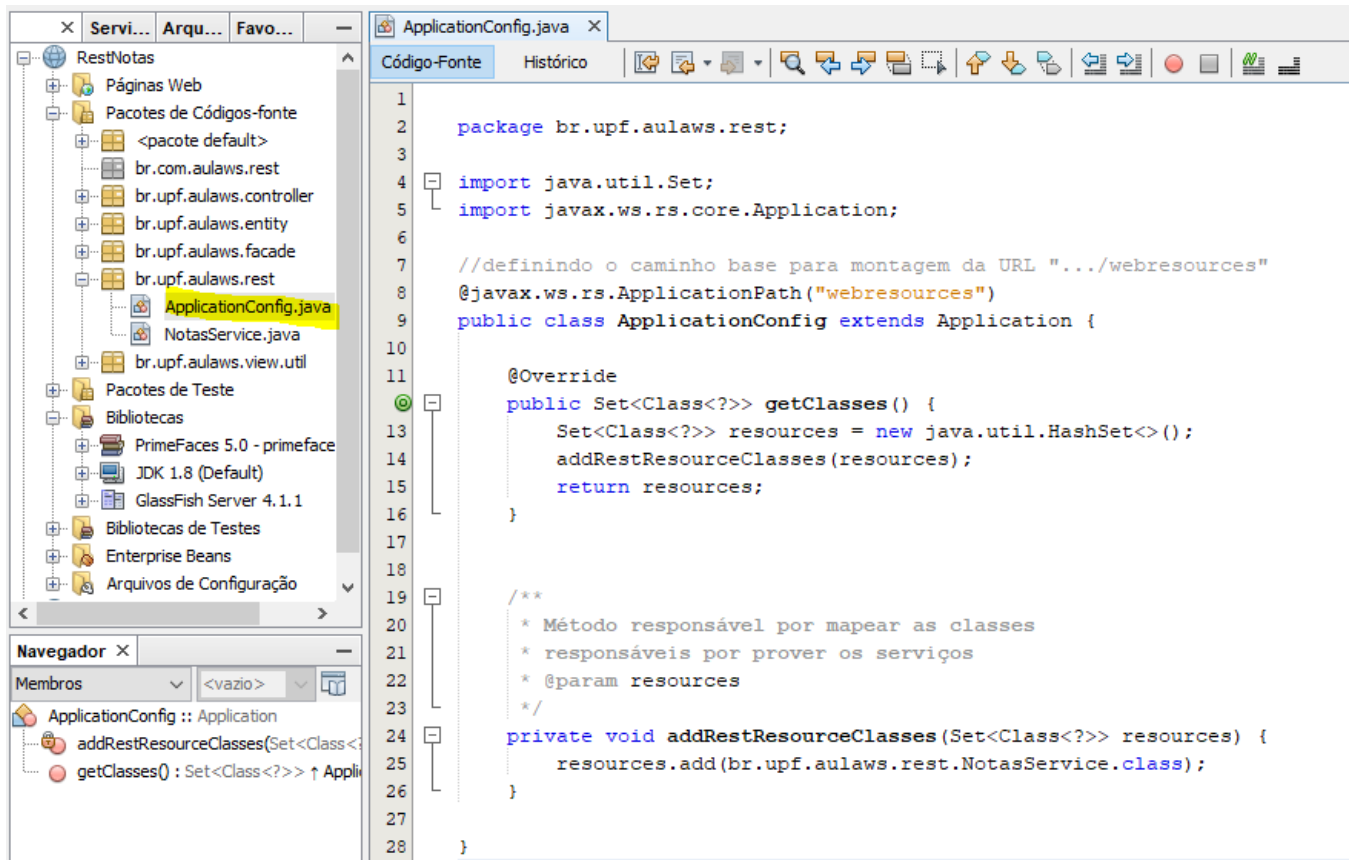
Serviço que irá buscar todos os serviços cadastrados no banco de dados...

```
50  /**
51   * serviço responsável por listar todos registros
52   * @return
53   */
54   @GET
55   @Override
56   @Path("/listAll") //adicionando caminho na URL
57   @Produces(MediaType.APPLICATION_JSON)
58   public List<TbNota> findAll() {
59       return super.findAll();
60   }
```

Obs.: É importante salientar que o serviço JPA que executa busca no banco através do método `findAll` foi construído de forma automática no projeto. Caso seja necessária uma nova forma de busca, será necessário construir a consulta.

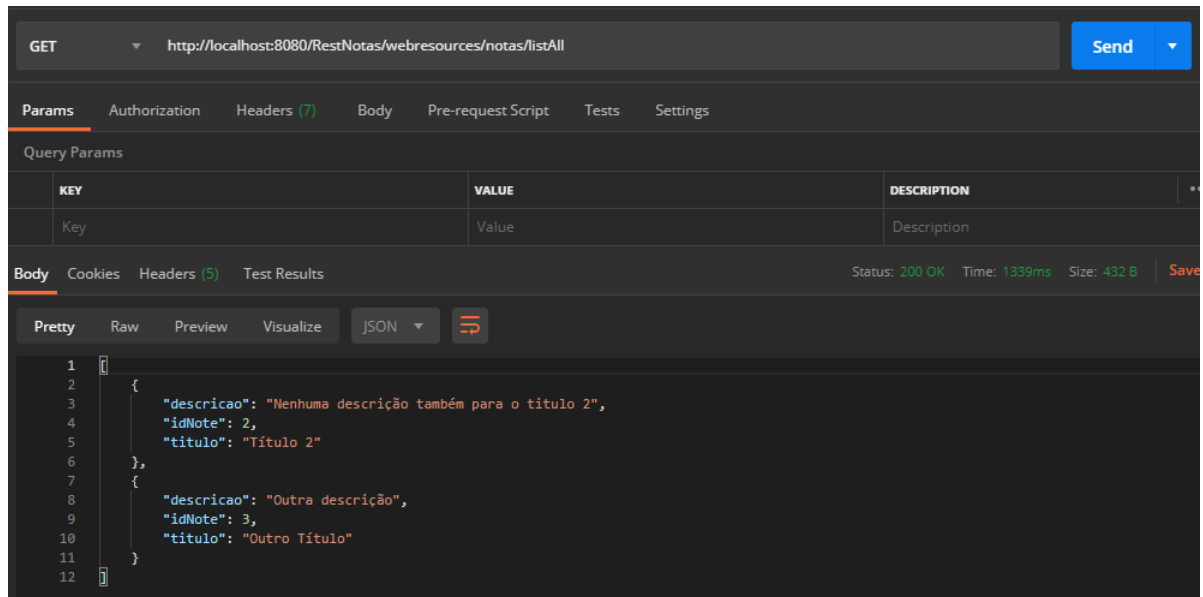
CONFIGURANDO WEB SERVICES – RESTful

Adicionando a classe de configuração de acesso aos Web Services ...



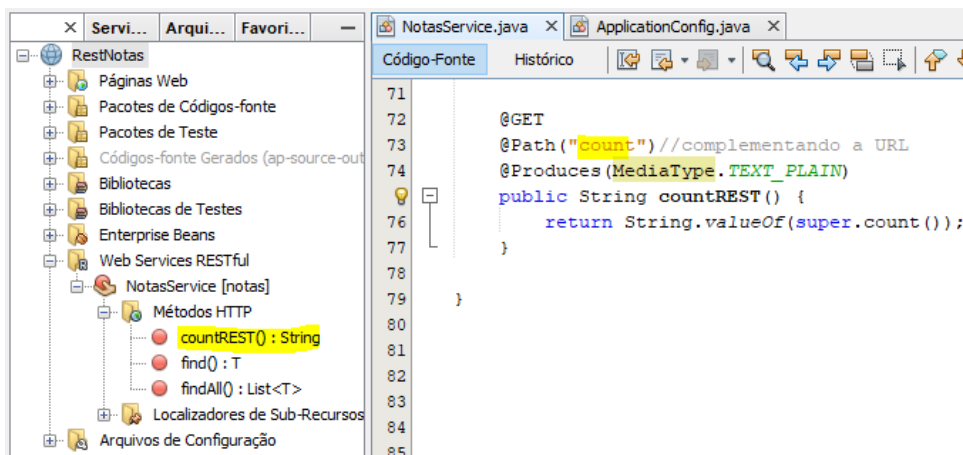
PRIMEIRO TESTE DO WEB SERVICES

Utilizando Postman, executar o teste utilizando método GET e executando a URI (endpoint: *http://localhost:8080/RestNotas/webresources/notes/listAll*).

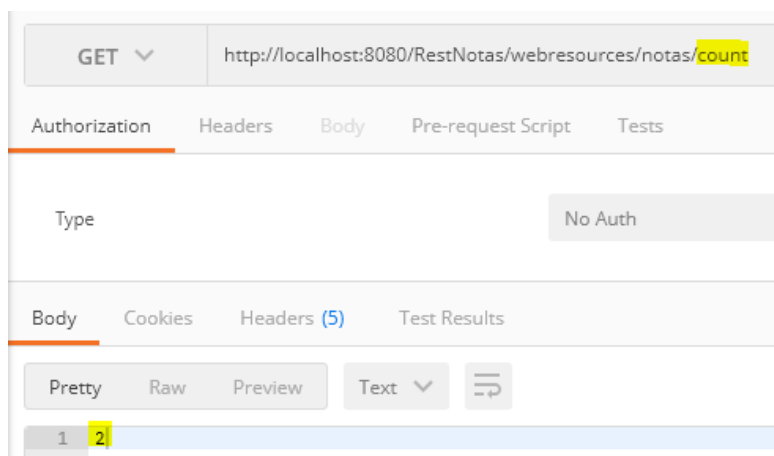


INCLUÍNDO NOVOS SERVIÇOS GET

1. Método GET: contar a quantidade de registros que constam cadastrados na base de dados.



2. Testando serviço de incremento da URL utilizando Postman...



3. Método GET: passando parâmetro para buscar um registro no banco de dados, passando {id}...

```
NotasService.java
104
105 /**
106  * Serviço responsável por buscar um registro, recebendo como parâmetro o ID
107  * do mesmo.
108  * @param id
109  * @return
110  */
111 @GET
112 @Path("/findByID/{id}") //recebendo parâmetro
113 @Produces(MediaType.APPLICATION_JSON)
114 public TbNota findById(@PathParam("id") Integer id) {
115     return super.find(id);
116 }
```

4. Testando serviço de passagem de parâmetro utilizando Postman...

GET http://localhost:8080/RestNotas/webresources/notas/findByID/3

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

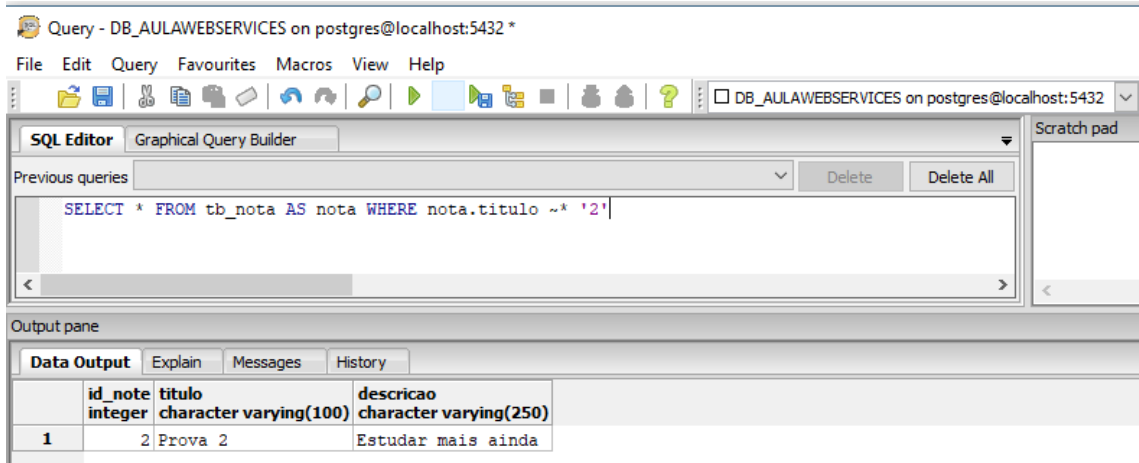
Body Cookies Headers (5) Test Results Status: 200 OK Time: 7ms Size: 352 B

Pretty Raw Preview Visualize JSON

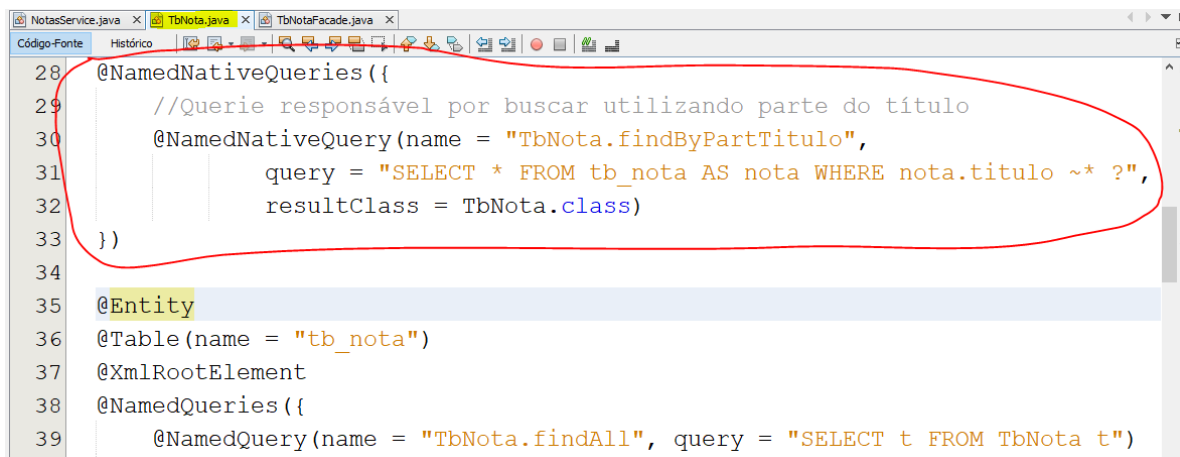
```
1 {
2   "descricao": "Nova descrição para o título 2",
3   "idNota": 3,
4   "titulo": "Título dois"
5 }
```

ADICIONANDO UM NOVO SERVIÇO GET PARA BUSCAR UMA NOTA UTILIZANDO PARTE DO TÍTULO INFORMADO.

1. Criando SQL de consulta e testando no PGAdmin.



2. Criando a NativeQuery na de entidade.



Obs: se necessário, pode ser adicionada uma JPQL na “@NamedQueries”, como a exemplo da “TbNota.findAll”.

3. Na classe Facade, criar o método para consumir a NativeQuery.

```
NotasService.java x TbNota.java x TbNotaFacade.java x
Código-Fonte Histórico
36 /**
37  * Método responsável por executar a consulta no banco buscando registro por
38  * parte do título.
39  *
40  * @param partTitulo
41  * @return
42  */
43 public List<TbNota> findByPartTitulo(String partTitulo) {
44     List<TbNota> lista = new ArrayList<>();
45     try {
46         Query query
47             = getEntityManager().createNamedQuery("TbNota.findByPartTitulo");
48         query.setParameter(1, partTitulo);
49         lista = query.getResultList();
50     } catch (Exception e) {
51         System.err.println("Error: " + e);
52     }
53     return lista;
54 }
```

4. Na classe de WebService, criar o novo serviço que busca um registro utilizando parte do título.

- a. Criar objeto EJB (Enterprise Java Bean) utilizado para consumir o serviço EJB da facade.

```
TbNota.java x TbNotaFacade.java x NotasService.java x
Código-Fonte Histórico
31 *
32 * @author Jeangrei
33 */
34 //montando o caminho base para montar a
35 //URL "http://localhost:8080/RestNotas/webresources/notas
36 @Stateless
37 @Path("notas")
38 public class NotasService extends AbstractFacade<TbNota> {
39
40     /**
41     * JPA: mapeando a unidade de persistencia...
42     */
43     @PersistenceContext(unitName = "RestNotasPU")
44     private EntityManager em;
45
46     @EJB
47     private br.upf.aulaws.facade.TbNotaFacade.ejbFacade;
48 }
```

b. Criar serviço para buscar nota utilizando parte do título

```
NotasService.java x
125
126 /**
127  * Serviço responsável por listar todos os registros que contenham parte do título.
128  *
129  * @param titulo
130  * @return
131  */
132 @GET
133 @Path("/findByPartTitulo/{partTitulo}")
134 @Produces({MediaType.APPLICATION_JSON})
135 public List<TbNota> findByIdPartTitulo(@PathParam("partTitulo") String titulo) {
136     return.ejbFacade.findByPartTitulo(titulo);
137 }
```

c. Testando serviço

GET http://localhost:8080/RestNotas/webresources/notas/findByPartTitulo/tulo Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (5) Test Results Status: 200 OK Time: 11.85s Size: 591

Pretty Raw Preview Visualize JSON

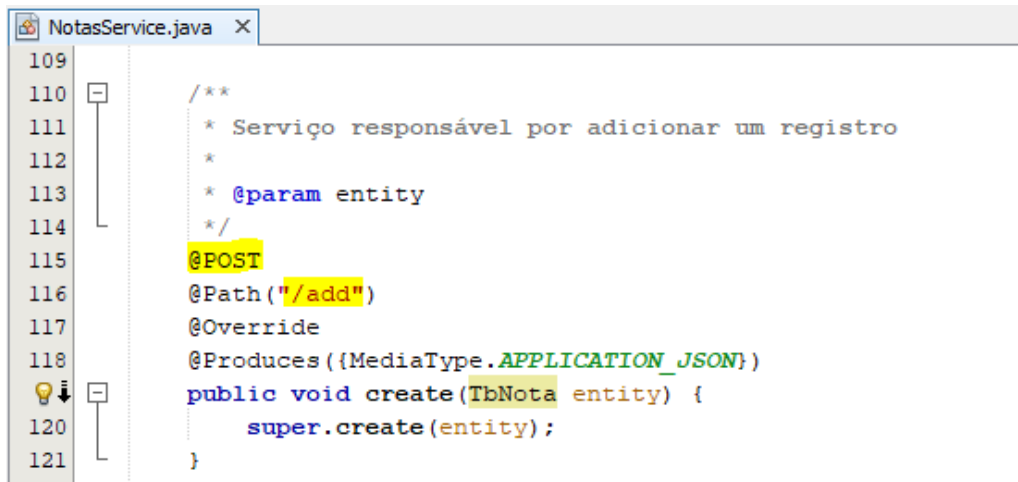
```
1 {
2   {
3     "descricao": "Nova descrição para o título 3",
4     "idNote": 4,
5     "titulo": "Título três"
6   },
7   {
8     "descricao": "Nova descrição para o título 2",
9     "idNote": 3,
10    "titulo": "Título dois"
11  },
12 }
```

ADICIONANDO SERVIÇOS POST – PUT – DELETE

A implementação dos métodos POST – PUT e DELETE são essenciais para completar as operações de inserção, atualização e remoção de registros via WebServices, como segue:

1. POST para inserir um registro na base de dados.

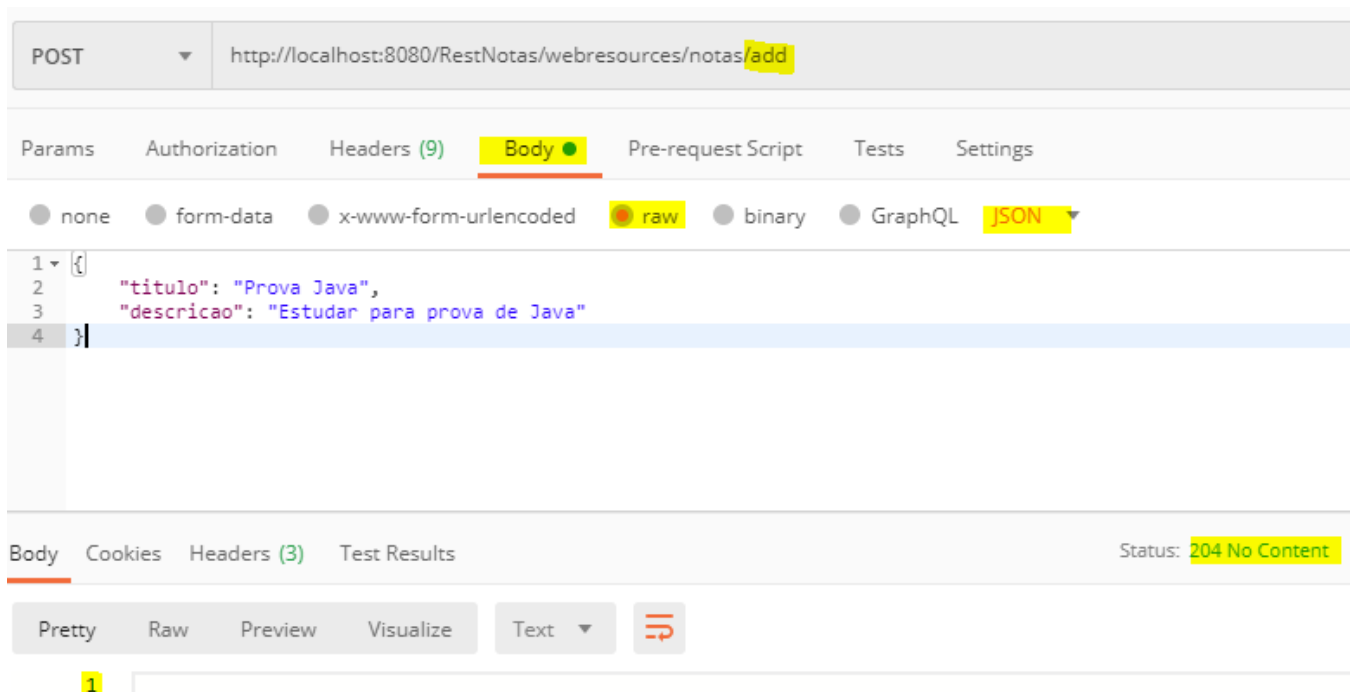
Para adicionar um serviço responsável por inserir um registro na base de dados, utilizamos o método POST conforme segue na imagem.



```

109
110  /**
111   * Serviço responsável por adicionar um registro
112   *
113   * @param entity
114   */
115   @POST
116   @Path("/add")
117   @Override
118   @Produces({MediaType.APPLICATION_JSON})
119   public void create(TbNota entity) {
120       super.create(entity);
121   }
  
```

Testando a criação de uma nota utilizando Postman, utilizando JSON como descritor.



POST http://localhost:8080/RestNotas/webresources/notas/add

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded **raw** binary GraphQL **JSON**

```

1 {
2   "titulo": "Prova Java",
3   "descricao": "Estudar para prova de Java"
4 }
  
```

Body Cookies Headers (3) Test Results Status: 204 No Content

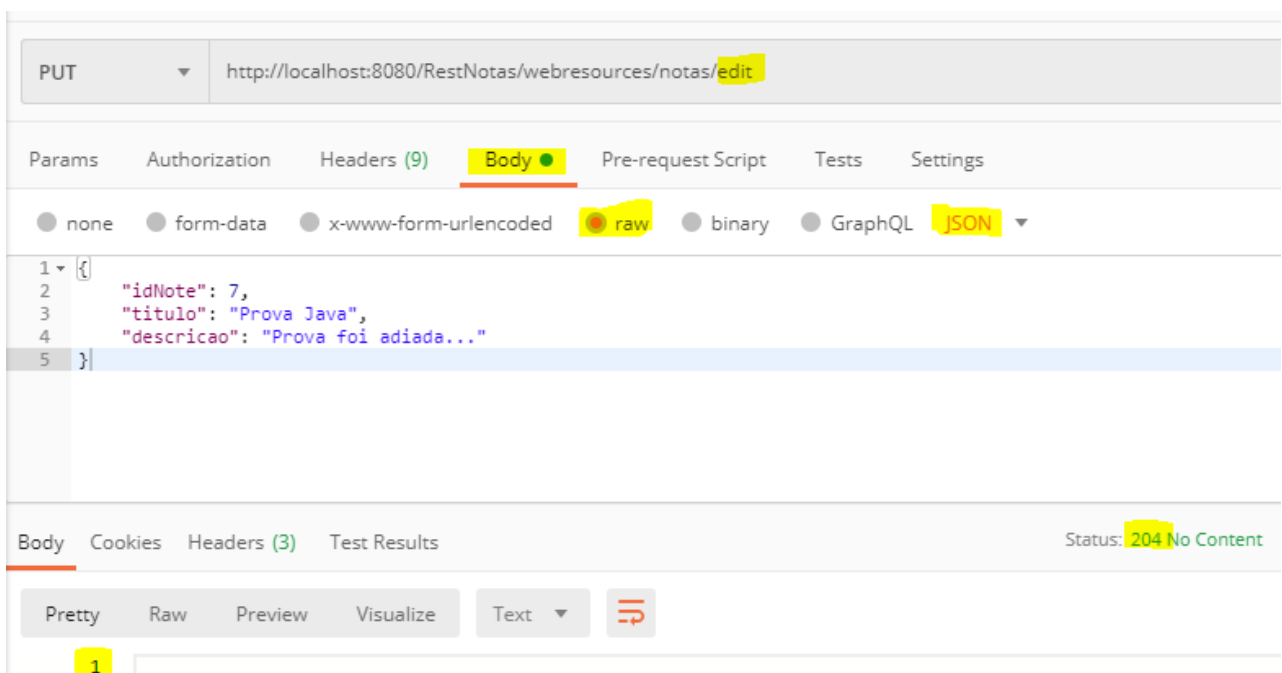
Pretty Raw Preview Visualize Text 1

2. PUT para atualizar um registro na base de dados.

Para adicionar um serviço responsável por atualizar um registro na base de dados, utilizamos o método PUT conforme segue na imagem.

```
NotasService.java X
128  /**
129   * Serviço responsável por salvar um registro editado
130   *
131   * @param entity
132   */
133  @PUT
134  @Path("/edit")
135  @Produces({MediaType.APPLICATION_JSON})
136  public void editNota(TbNota entity) {
137      super.edit(entity);
138  }
139
```

Testando a criação de uma nota utilizando Postman, utilizando JSON como descritor.



3. DELETE para deletar um registro na base de dados.

Para adicionar um serviço responsável por deletar um registro na base de dados, utilizamos o método DELETE conforme segue na imagem.

```
NotasService.java x
143
144  /**
145   * Serviço responsável por remover um registro
146   *
147   * @param id
148   */
149  @DELETE
150  @Path("/delete/{id}")
151  @Produces({MediaType.APPLICATION_JSON})
152  public void remove(@PathParam("id") Integer id) {
153      super.remove(super.find(id));
154  }
155
156
157
```

Testando a criação de uma nota utilizando Postman.

