PLATAFORMA SIGA RIO DAS VELHAS

MANUAL DO CÓDIGO FONTE

ATO CONVOCATÓRIO Nº 006/2016
CONTRATO DE GESTÃO IGAM Nº 002/IGAM/2012
09/2017

PLATAFORMA SIGA RIO DAS VELHAS

MANUAL DO CÓDIGO FONTE

ATO CONVOCATÓRIO Nº 006/2016
CONTRATO DE GESTÃO IGAM Nº 002/IGAM/2012
09/2017

1.0	01/09/2017	Versão Inicial

Revisão	Data	Descrição Breve	Ass. do Autor	Ass. do Superior	Ass. de Aprovação

PLATAFORMA SIGA RIO DAS VELHAS

MANUAL DO CÓDIGO FONTE

Elaborado por: K2 Sistemas	**Supervisionado por:**		
Aprovado por:	Revisão	Finalidade	Data
	1	3	
Legenda Finalidade [1] Para Informação [2] Para Comentário [3] Para Aprovação			

K2FS Sistemas e Projetos Ltda.

Av. Rio Branco 26, Sobreloja 20090-001, Centro

Rio de Janeiro, RJ

Telefone: 21-2239-1190 – k2@k2sistemas.com.br

Apresentação do Trabalho

O presente documento “MANUAL DO CÓDIGO FONTE” contém as instruções de uso e entendimento do código fonte da plataforma SIGA Rio das Velhas, com o intuito de facilitar a manutenção do sistema por seus administradores.

Índice Analítico

Lista de Figuras

Manual do Código Fonte

1) Introdução

O projeto SIGA Rio das Velhas – Sistema de Informações Geográficas e Ambientais da bacia do Rio das Velhas – tem como objetivo principal a construção de uma plataforma tecnológica para auxiliar no processo de gestão do conhecimento produzido sobre a bacia do Rio das Velhas, permitindo o acesso às informações de forma abrangente, interoperável e colaborativa.

As soluções desenvolvidas no projeto possibilitam o armazenamento, publicação e manutenção dos dados produzidos na elaboração do Plano Diretor de Recursos Hídricos da Bacia Hidrográfica do Rio das Velhas, dos dados de acompanhamento das outorgas de uso da água (emitidos para a bacia do Rio das Velhas) e, também, dos dados geográficos da bacia. Ainda, a solução permite gerir os conteúdos dos usuários e das ferramentas que compõem a plataforma, permitindo a difusão de informações e conteúdos por meio de uma sala de situação que será futuramente implantada pelo CBH Rio das Velhas.

Os resultados deste projeto servirão como base para a descentralização da obtenção e produção de dados, para garantir a sociedade o acesso às informações e, principalmente, para possibilitar a coordenação unificada da bacia hidrográfica.

2) Ambiente

O servidor que irá hospedar o sistema necessita ter os seguintes softwares instalados:

- Sistema Operacional: Windows ou Linux
- Servidor Apache Tomcat 8
- Banco de dados PostgresSQL

Para visualização do código fonte e do banco de dados, são necessários os softwares:

- Netbeans 8
- PgAdmin

3) Visão geral do código fonte

O código fonte foi desenvolvido no padrão **MVC** (Model – View – Controller) usando o framework **ZK Framework 7** (https://www.zkoss.org/).

As linguagens de programação utilizadas foram:

- Java
- HTML
- Javascript
- CSS
- XML

3.1 Entendendo o MVC

Model-view-controller (MVC), em português **modelo-visão-controlador**, é um padrão de arquitetura de software que separa a representação da informação da interação do usuário com ele, sendo que:

- O modelo (**model**) consiste nos dados da aplicação, regras de negócios, lógica e funções;
- Uma visão (**view**) pode ser qualquer saída de representação dos dados, como uma tabela ou um diagrama. É possível ter várias visões do mesmo dado, como um gráfico de barras para gerenciamento e uma visão tabular para contadores;
- O controlador (**controller**) faz a mediação da entrada, convertendo-a em comandos para o modelo ou visão. As ideias centrais por trás do MVC são a reusabilidade de código e separação de conceitos.

Além de dividir a aplicação em três tipos de componentes, o desenho MVC define as interações entre eles:

- Um controlador (**controller**) envia comandos para o modelo para atualizar o seu estado (por exemplo, editando um documento). O controlador também

pode enviar comandos para a visão associada para alterar a apresentação da visão do modelo (por exemplo, percorrendo um documento);

- Um modelo (**model**) armazena dados e notifica suas visões e controladores associados quando há uma mudança em seu estado. Estas notificações permitem que as visões produzam saídas atualizadas e que os controladores alterem o conjunto de comandos disponíveis. Uma implementação passiva do MVC monta estas notificações, devido a aplicação não necessitar delas ou a plataforma de software não as suportar;
- A visão (**view**) Gera uma representação (Visão) dos dados presentes no modelo solicitado.

4) Estrutura das pastas

O projeto está organizado conforme mostra a Figura 1, onde cada tópico corresponde a uma pasta:

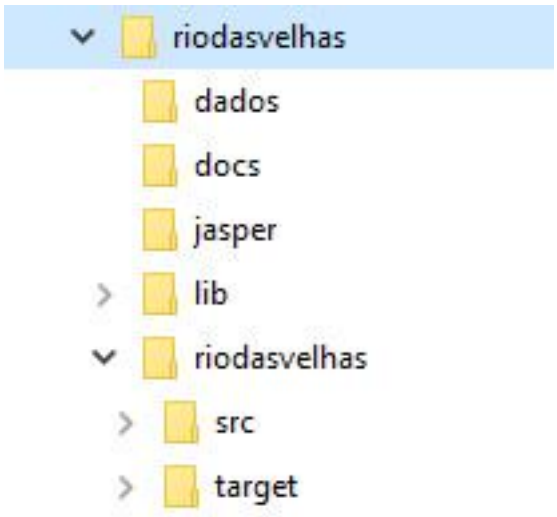

Figura 1 - Estrutura das pastas

As pastas disponíveis são:

- Riodasvelhas
  - Dados (scripts relacionados ao banco de dados);
  - Docs (documentos relacionados ao projeto);

- Jasper (arquivos relacionados aos possíveis relatórios gerados pelo sistema);
- Lib (dependências em java dos plugins externos utilizados no código fonte);
- Riodasvelhas (pasta onde ficam armazenados os arquivos do projeto)
  - Src (código-fonte);
  - Target (arquivo compilado do projeto que deve ser instalado no servidor).

5) Editando o código fonte

Para editar o código fonte, é preciso abrir o projeto utilizando o software NetBeans 8.0, conforme mostra a Figura 2.

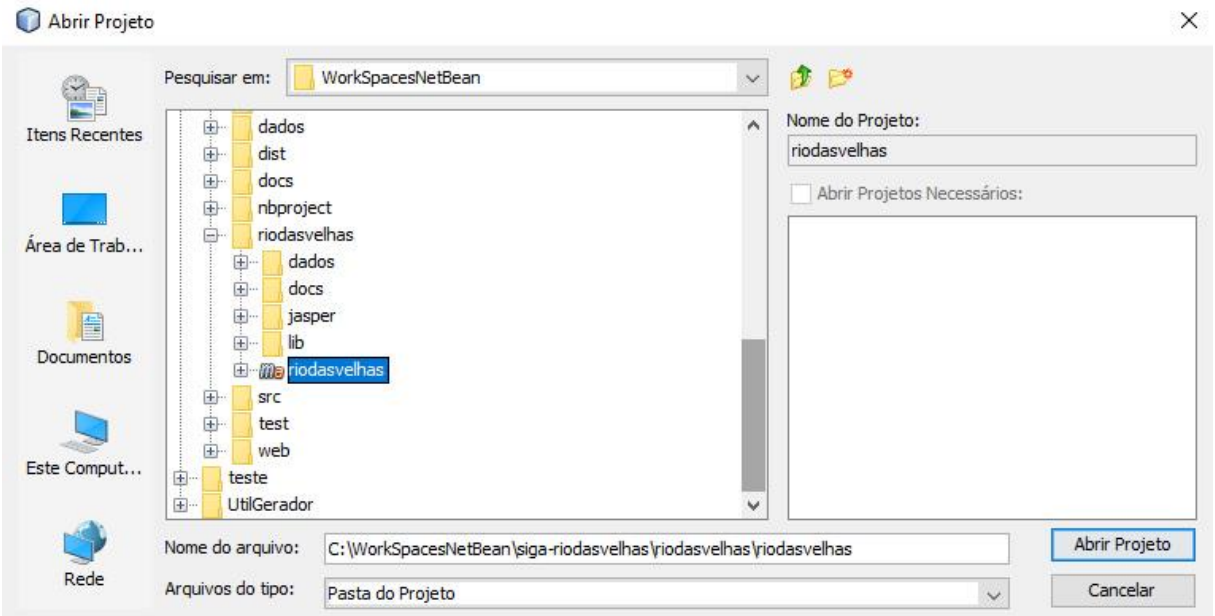

Figura 2 – Abrindo o projeto

Após abrir o projeto no NetBeans, será possível identificar a estrutura do projeto e seus arquivos fonte, conforme mostra a Figura 3.

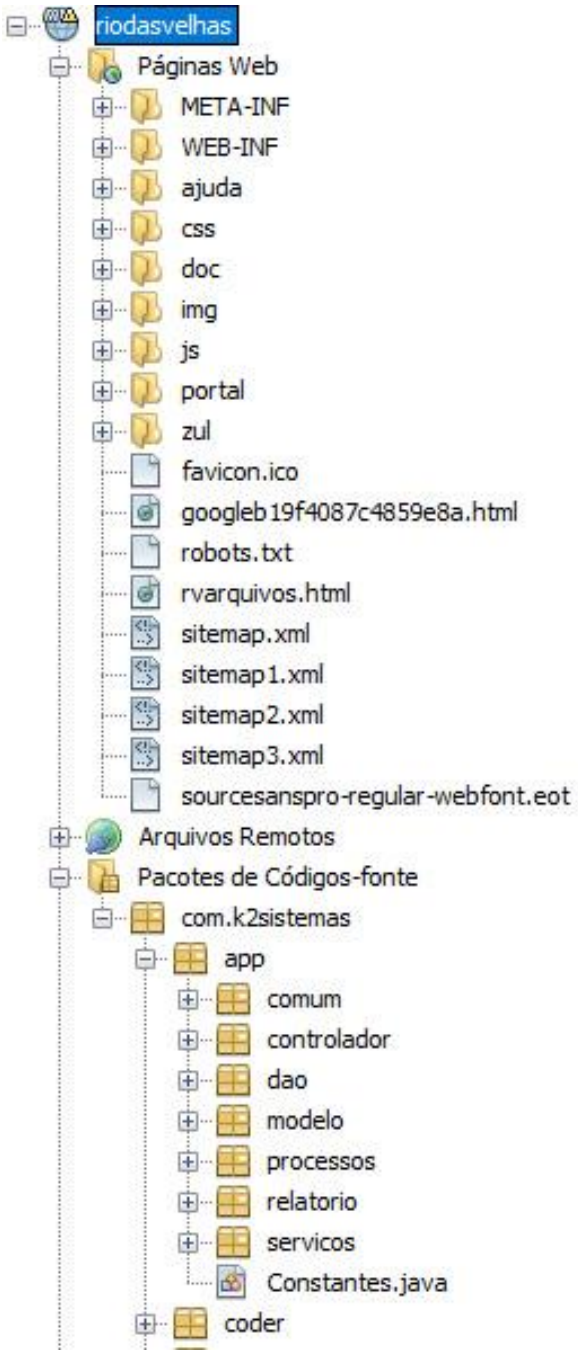

Figura 3 – Estrutura do projeto

5.1 View

Os arquivos de “View” (relacionados à aparência das páginas) podem ser encontrados dentro do diretório “Páginas Web” e são divididos em: Portal e Módulo de administração.

5.1.1 Portal

Dentro do diretório “Portal” podemos localizar os arquivos utilizados na construção do portal Rio das Velhas.

Estes arquivos estão divididos em CSS, Javascript, ZUL entre outros.

Os arquivos base para a construção das principais páginas do portal SIGA RIO DAS VELHAS são:

Página	Arquivo
Início	index.zul
Velhas Map	sigaweb.zul
Siplan	siplan.zul
[Menu lateral]	sidebar.zul

A Figura 4 apresenta a localização destes arquivos base dentro do projeto.

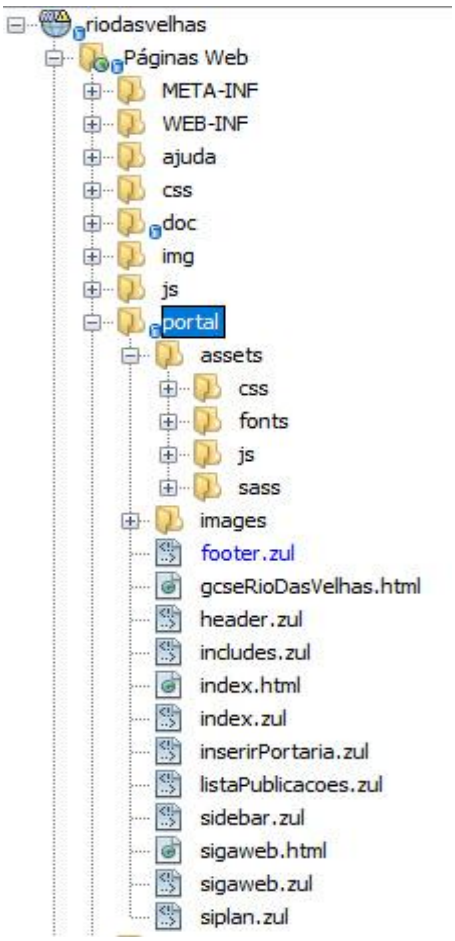

Figura 4 – View do portal

5.1.2 Módulo de administração

Dentro da pasta “zul” estão os arquivos do módulo de administração. Neste diretório, cada pasta corresponde a um item do menu do SIGA.

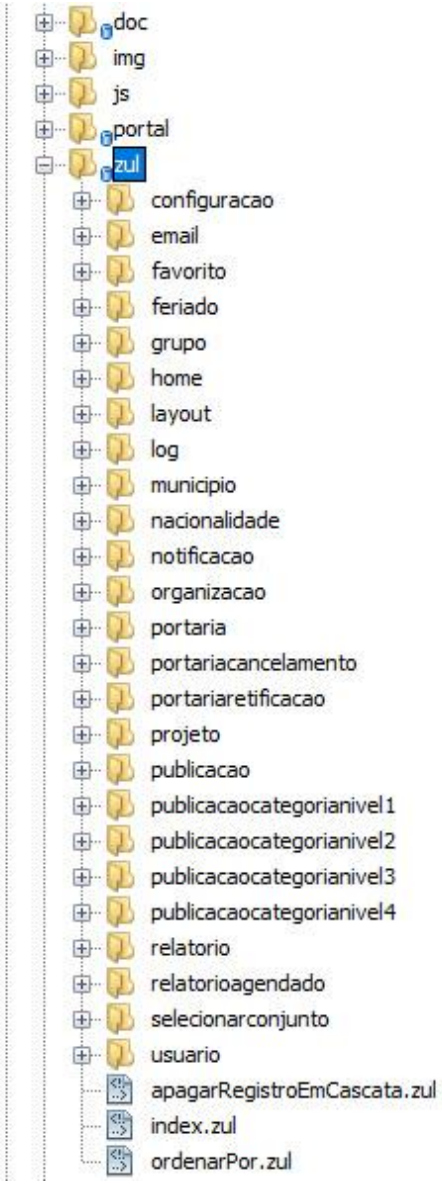

Figura 5 – View do módulo de administração

5.2 Modelos, Controladores e Dados

Os arquivos que contém toda a lógica da programação e consulta a base de dados estão localizados no diretório “Pacotes de código fonte”.

Estes arquivos estão organizados pelas respectivas pastas:

- modelo;
- controlador;
- dao (dados).

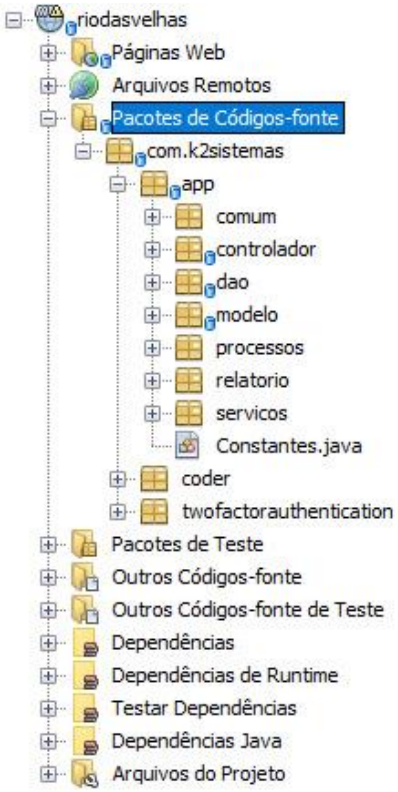

Figura 6 – Modelos, controladores e dados

5.2.1 Modelos

São as classes de objetos com seus atributos e métodos, por exemplo:

A classe “Portaria” contem os atributos (data, requerente, outorgante, entre outros...):

```
public class Portaria extends ModeloBase {
```

```
    private Integer codPortaria;

    private String portaria;

    private String outorgante;

    private String requerente;

...

}
```

E seus respectivos métodos:

```
    public Integer getCodPortaria() {

        return codPortaria;

    }

    public void setCodPortaria(Integer codPortaria) {

        this.codPortaria = codPortaria;

    }
```

5.2.2 Controladores

5.2.2.1 Interface

São os métodos abstratos que serão implementados posteriormente nos controladores, como podemos ver no arquivo "**FormPortariaI.java**":

```
public interface FormPortariaI extends FormModeloBaseI {

    public String getPortaria();

    public String getOutorgante();

    public String getRequerente();

...

}
```

5.2.2.2 Implementação

É a implementação da interface abstrata que foi definida anteriormente. Onde fica toda a lógica da comunicação entre a “view” e os métodos, cálculos e processamentos:

```
    @Override
    public void doAfterCompose(Component comp, FabricaDAO fabricaDAO) throws Exception {
        super.doAfterCompose(comp, fabricaDAO);
...
        portaria.setValue(registroSelecionado.getPortaria());
        outorgante.setValue(registroSelecionado.getOutorgante());
        requerente.setValue(registroSelecionado.getRequerente());
...
}
    @Override
    public String getOutorgante() {
        return outorgante.getValue();
    }
    @Override
    public String getRequerente() {
        return requerente.getValue();
    }
```

5.2.3 Dados

Lugar onde são feitas as consultas, inserções, edições e exclusões de registros no banco de dados:

```
    @Override
    public Portaria getPortaria(Integer codPortaria) throws Exception {
        if (codPortaria == null) {
```

```
        return null;
    }
    Portaria Portaria = null;
    NamedParameterStatement nps = null;
    ResultSet rs = null;
    try {
        Connection conn = fabricaDAO.getAppConnection();
        String sql = constroiSqlBasicoPortaria(false, false) + " AND (CodPortaria = :CodPortaria)";
        nps = new NamedParameterStatement(conn, sql);
        nps.setInt("CodPortaria", codPortaria);
        rs = nps.executeQuery();
        if (rs.next()) {
            Portaria = constroiPortaria(rs, false);
        }
    } finally {
        fabricaDAO.close(nps, rs);
    }
    return Portaria;
}
```

6) Exemplos

As figuras 7, 8, 9 e 10 apresentam, respectivamente, exemplos de arquivos que representam uma “view”, um “modelo”, um “controlador” e uma interação com a base de dados.

- Na figura 7 podemos ver a criação de alguns elementos na tela, definição de classes com estilos pré-definidos e inclusão de texto estático.

```
<?xml version="1.0" encoding="UTF-8"?>
<?page docType="html PUBLIC &quot;-//W3C//DTD XHTML 1.0 Transitional//EN&quot; &quot;http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd&quot;" title="S

<zk xmlns:w="client" xmlns:n="native">

    <!--Inclui css e favicon -->
    <div fulfill="=header.zul"></div>

    <!-- Sidebar -->
    <div fulfill="=sidebar.zul"></div>

    <!-- Wrapper -->
    <n:div id="wrapper">

        <!-- Intro -->
        <n:section id="intro" class="wrapper style1 fullscreen fade-up">

            <n:div class="inner">
                <n:h1>SIGA Rio das Velhas</n:h1>

                <n:p>O projeto SIGA VELHAS tem como objetivo principal a construção de uma plataforma tecnológica para auxiliar no processo de gestão do con

                <n:p>As soluções desenvolvidas no projeto possibilitam o armazenamento, publicação e manutenção dos dados produzidos na elaboração do Plano

                <n:p>Os resultados deste projeto servirão como base para a descentralização da obtenção e produção de dados, para garantir a sociedade o ace

                <n:p><n:strong><n:u>Os produtos desenvolvidos são:</n:u></n:strong></n:p>

                <n:p><n:strong>SIPLAN</n:strong> - Módulo de gestão e atualização de dados do Plano Diretor: objetivo de administrar toda a informação produ
```

Figura 7 – Exemplo de uma view

(/src/main/webapp/portal/índex.zul)

- Na figura 8 vemos a criação do objeto “Publicacao” com seus atributos e alguns dos seus métodos.

```
import com.k2sistemas.app.modelo.publicacaocategoriaNivel3.PublicacaoCategoriaNivel3;
import com.k2sistemas.app.modelo.publicacaocategoriaNivel4.PublicacaoCategoriaNivel4;

public class Publicacao extends ModeloBase {

    private Integer codPublicacao;
    private PublicacaoCategoriaNivel2 publicacaoCategoriaNivel2;
    private PublicacaoCategoriaNivel1 publicacaoCategoriaNivel1;
    private PublicacaoCategoriaNivel3 publicacaoCategoriaNivel3;
    private PublicacaoCategoriaNivel4 publicacaoCategoriaNivel4;
    private Integer ano;
    private String publicacao;
    private String tags;
    private Midia arquivoMidia;
    private Midia arquivoCapa;
    private Integer ordem;

    @Override
    public String getNomeUniversal() {
        return publicacao;
    }

    @Override
    public Integer getCodigoUniversal() {
        return codPublicacao;
    }

    public Integer getCodPublicacao() {
        return codPublicacao;
    }
```

Figura 8 – Exemplo de um modelo

(/src/main/Java/com/k2sistemas/app/modelo/publicacao/Publicacao.java)

- Na figura 9, temos um controlador que gerencia a tela de cadastro de uma publicação. No script podemos ver que é aplicada uma lógica condicional, a definição de valores às variáveis e o preenchimento das comboboxes com valores que são buscados no banco de dados.

```
@Override
public void doAfterCompose(Component comp, FabricaDAO fabricaDAO) throws Exception {
  super.doAfterCompose(comp, fabricaDAO);
  prepararBtnEnviarMidia(btnEnviarArquivo);
  prepararBtnEnviarMidia(btnEnviarArquivoCapa);
  codPublicacaoCategoriaNivel1Selecionado = null;
  codPublicacaoCategoriaNivel2Selecionado = null;
  codPublicacaoCategoriaNivel3Selecionado = null;
  codPublicacaoCategoriaNivel4Selecionado = null;

  if (!isAdicao()) {
    // Recarregar o registro: 1) pode ter mudado 2) a lista pode ter uma "versao simplificada"
    if (!recarregarRegistroSelecionado(fabricaDAO)) {
      abortarFormulario(ControladorPublicacaoLista.URL, Constantes.REGISTRO_NAO_DISPONIVEL);
      return;
    }
  }

  if (isAdicao()) {
    setAutocompleteCombobox(codPublicacaoCategoriaNivel2, this, codPublicacaoCategoriaNivel2Selecionado);
    setAutocompleteCombobox(codPublicacaoCategoriaNivel1, this, codPublicacaoCategoriaNivel1Selecionado);
    setAutocompleteCombobox(codPublicacaoCategoriaNivel3, this, codPublicacaoCategoriaNivel3Selecionado);
    setAutocompleteCombobox(codPublicacaoCategoriaNivel4, this, codPublicacaoCategoriaNivel4Selecionado);
    lblInformacaoArquivoNovo.setVisible(false);
    return;
  }
  publicacao.setValue(registroSelecionado.getPublicacao());
  tags.setValue(registroSelecionado.getTags());
```

Figura 9 – Exemplo de um controlador

(/src/main/Java/com/k2sistemas/app/controlador/zk/publicacao/ControladorPublicacao.java)

- Na figura 10, podemos ver um exemplo de atualização de registro na base de dados, onde são definidos os valores resgatados do formulário e inseridos nas colunas do registro em questão no banco de dados.

```
    sql = "UPDATE " + fabricaDAO.getAppTabsOwner() + "Publicacao SET"
            + " Publicacao = :Publicacao"
            + ", CodPublicacaoCategoriaNivel2 = :CodPublicacaoCategoriaNivel2"
            + ", CodPublicacaoCategoriaNivel1 = :CodPublicacaoCategoriaNivel1"
            + ", CodPublicacaoCategoriaNivel3 = :CodPublicacaoCategoriaNivel3"
            + ", CodPublicacaoCategoriaNivel4 = :CodPublicacaoCategoriaNivel4"
            + ", Ano = :Ano"
            + ", Ordem = :Ordem"
            + ", Tags = :Tags"
            + " WHERE CodPublicacao = :CodPublicacao";
  }
  boolean salvo = false;
  Connection conn = fabricaDAO.getAppConnection();
  boolean auto = conn.getAutoCommit();
  NamedParameterStatement nps = null;
  conn.setAutoCommit(false);
  try {
    nps = new NamedParameterStatement(conn, sql);
    nps.setString("Publicacao", publicacao.getPublicacao());
    nps.setInt("CodPublicacaoCategoriaNivel2", publicacao.getPublicacaoCategoriaNivel2() != null ? publicacao.getPublicacaoCategoriaNivel2().getCodPublica
    nps.setInt("CodPublicacaoCategoriaNivel1", publicacao.getPublicacaoCategoriaNivel1() != null ? publicacao.getPublicacaoCategoriaNivel1().getCodPublica
    nps.setInt("CodPublicacaoCategoriaNivel3", publicacao.getPublicacaoCategoriaNivel3() != null ? publicacao.getPublicacaoCategoriaNivel3().getCodPublica
    nps.setInt("CodPublicacaoCategoriaNivel4", publicacao.getPublicacaoCategoriaNivel4() != null ? publicacao.getPublicacaoCategoriaNivel4().getCodPublica
    nps.setInt("Ano", publicacao.getAno());
    nps.setInt("Ordem", publicacao.getOrdem());
    nps.setString("Tags", publicacao.getTags());
    if (!insercao) {
      nps.setInt("CodPublicacao", publicacao.getCodPublicacao());
    }
```

Figura 10 – Exemplo de uma interação com a base de dados

(/src/main/Java/com/k2sistemas/app/dao/jdbc/postgres/publicacao/PublicacaoDAO.java)