

UNIVERSIDADE LUTERANA DO BRASIL – ULBRA CAMPUS CANOAS

Av. Farroupilha, 8.001 – Bairro São José – CEP 92425- 90 – Canoas - RS

**GESTÃO DE TROCAS DE PRODUTOS E SERVIÇOS  
TRABALHO FINAL DESENVOLVIMENTO DE APLICAÇÕES  
ORIENTADA A COMPONENTES**

ALUNOS: ANTONIO CARLOS GUIMARÃES ROLIM

ALEX SANDRO CÓRDOVA DE SOUZA

CHARLES PINTO DE LEMOS

GIOVANI LIMA

DISCIPLINA: DESENVOLVIMENTO DE APLICAÇÕES ORIENTADA A COMPONENTES

PROFESSORA: FERNANDA DE NARDIN WALKER

Canoas, 03 de dezembro de 2020.

## **1. INTRODUÇÃO**

Esta é a documentação do trabalho final do projeto da cadeira de Desenvolvimento de Aplicações Orientadas a Objetos e leva o nome fantasia de “BRIQUE DA GURIZADA”. A aplicação tem como objetivo facilitar a maneira que as pessoas efetuam uma negociação, seja ela de relação serviço-serviço, serviço-produto ou produto-produto, sem a necessidade de envolver moeda corrente visto que durante o período de pandemia as pessoas encontram-se com menor poder aquisitivo.

Para a realização do mesmo, foram estipulados requisitos mínimos pela professora, como a utilização do Paradigma de Orientação a Objetos, linguagem de programação Java, interface gráfica, utilização de Threads e persistência dos dados em um BD ou arquivos. Também foi autorizada a utilização de recursos como frameworks, IDE's, entre outros, desde que tivessem sido informados anteriormente.

O trabalho divide-se em quatro etapas, sendo elas:

- Etapa 1: Definição do projeto e integrantes do grupo de trabalho – 22/10/2020.
- Etapa 2: Apresentação da modelagem (diagrama de classes) – 31/05/2020.
- Etapa 3: Entrega do projeto (código fonte) – 03/12/2020.
- Etapa 4: Entrega da documentação do projeto – 03/12/2020.

Por fim, no dia 03/11/2020 será realizado a apresentação dos projetos, considerando que cada participante terá o tempo de 5 minutos. A seguir apresenta-se a estrutura do projeto.

## **2. ESTRUTURA DO PROJETO**

O Backend do projeto foi criado em Java em conjunto com o Framework Spark. Para persistência dos dados foi utilizado o banco de dados Oracle XE e o driver Oracle JDBC.

O Frontend, como a aplicação é server-side, foi utilizado o Thymeleaf para realizar a renderização. Os mesmos foram desenvolvidos com HTML, CSS e utilizado a lib Bootstrap.

Por fim, o versionamento de código foi feito com o Git e hospedado no Github. O gerenciamento de atividades da equipe foi na ferramenta no Trello.

## **3. DIAGRAMA DE CLASSES – PROJETO**

O projeto consiste em um sistema de trocas de produtos e/ou serviços entre usuários, desde que essas trocas não envolvam moeda corrente de nenhuma espécie também não envolvam nada de cunho ilícito (armas, drogas, etc.) bem como de cunho sexual, político e religioso. O sistema oferecerá a seus usuários a possibilidade de cadastrar produtos e serviços que tenha interesse de divulgar e busca por produtos que necessite. O sistema proporcionará ao usuário fazer proposta para outro usuário, podendo inclusive alterar a proposta durante a negociação caso a outra parte não queira seu produto ou serviço oferecidos.

Com base nas descrições acima, o projeto foi pensado na seguinte diagramação:

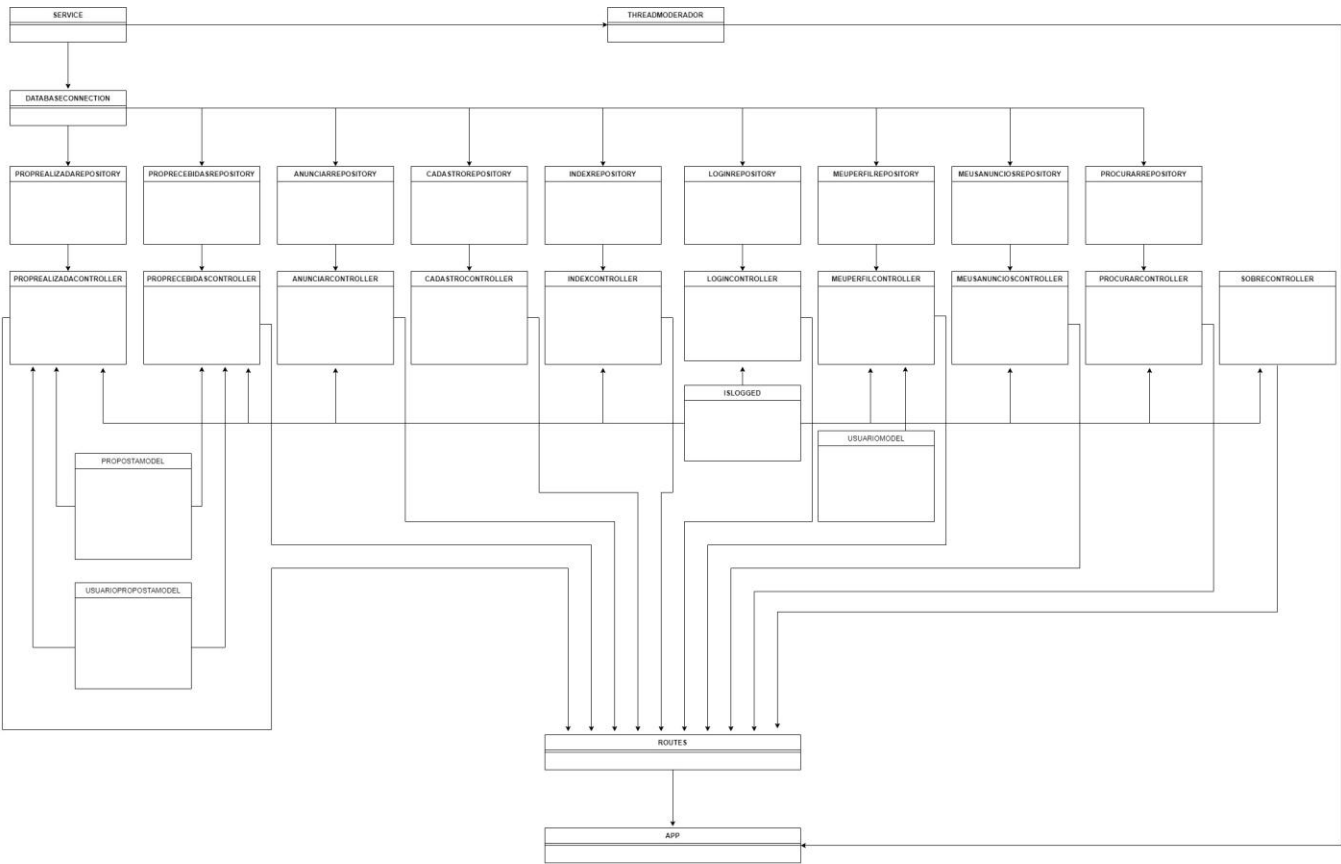


Figura 1: Diagramação do projeto.

## 4. CLASSES DO PROJETO

O projeto tem a finalidade única e exclusivamente de proporcionar a seus usuários a possibilidade de realizar a troca de produtos e/ou serviços, através de aplicação web. Para isto foram criadas 29 classes, sendo uma delas a App principal de aplicação.

Abaixo, a lista das classes do projeto:

- class App
- class UsuárioModel
- class PropostaModel
- class PropostaxUsuárioModel
- class NegociacaoModel
- class AnunciarController
- class CadastroController
- class IndexController
- class LoginController
- class MeuPerfilController
- class MeusAnunciosController
- class ProcurarController
- class PropRealizadasController
- class PropRecebidasController
- class SobreController
- class AnunciarRepository
- class CadastroRepository
- class IndexRepository
- class LoginRepository
- class MeuPerfilRepository
- class MeusAnunciosRepository
- class ProcurarRepository
- class PropRealizadasRepository
- class PropRecebidasRepository
- class ServiceRepository
- class Router
- class DataBaseConection
- class IsLogged
- class ThreadModerador

## 4.1 Class App

A classe App utiliza a classe Router e a classe ThreadModerador. A Router serve para ficar “escutando” a rota que o usuário está solicitando e assim disponibiliza-la quando solicitada e a ThreadModerador instancia uma thread que executa de tempos em tempos, removendo palavras proibidas dos anúncios cadastrados, inativando os mesmos, agindo como uma espécie de moderador do sistema.

```
package trabalhos;  
  
import routes.Router;  
import service.ThreadModerador;  
  
public class App {  
    Run | Debug  
    public static void main(String[] args) {  
  
        new Router().getAllRoutes();  
        ThreadModerador tm = new ThreadModerador();  
        Thread t1 = new Thread(tm);  
        t1.start();  
  
        //http://localhost:4567  
    }  
}
```

Figura 2: Código da Classe App

## 4.2 Class UsuarioModel

A classe UsuarioModel possui oito atributos e seus métodos getters e setters e está contido dentro do package **MODEL**.

```
1  package model;  
2  
3  public class UsuarioModel {  
4  
5      private int pkcodusuario;  
6      private String nome;  
7      private String cpf;  
8      private String telefone;  
9      private String login;  
10     private String senha;  
11     private String id;  
12     private String foto;  
13 }
```

Figura 3: código da classe UsuárioModel

### 4.3 Class PropostaModel

A classe **PropostaModel** possui oito atributos e seus métodos getters e setters e está contido dentro do package **MODEL**.

```
1 package model;
2
3 import java.sql.Timestamp;
4
5 public class PropostaModel {
6
7     private int pkcodproposta;
8     private String titulo;
9     private String descricao;
10    private String tipo;
11    private int ativo;
12    private int oferta;
13    private String foto1;
14    private String foto2;
15    private String foto3;
16    private Timestamp dataHora;
17    private int fkcodusuario;
18 }
```

Figura 4: código classe PropostaModel

### 4.4 Class UsuarioxPropostaModel

A classe **UsuarioxPropostaModel** possui oito atributos e seus métodos getters e setters e está contido dentro do package **MODEL**.

```
1 package model;
2
3 public class UsuarioxPropostaModel extends UsuarioModel{
4
5     private int pkcodproposta;
6     private String titulo;
7     private String descricao;
8     private String tipo;
9     private String dataHora;
10    private String ativo;
11 }
```

Figura 5: código da classe UsuarioxPropostaModel

## 4.5 Class AnunciarController

Esta classe está contida dentro do package **CONTROLLER** e tem por objetivo primeiramente de ver se a pessoa está logada no sistema, caso não esteja, envia o usuário para a página de login, caso contrário, redireciona para página de cadastro de anúncio. Também é responsável por criar um anúncio.

```
package controller;

import static spark.Spark.*;
import java.util.HashMap;
import java.util.Map;
import model.PropostaModel;
import repository.AnunciarRepository;
import service.IsLogged;
import spark.Request;
import spark.Response;
import spark.template.thymeleaf.ThymeleafTemplateEngine;

public class AnunciarController {

    public static String getAnunciarPage(Request req, Response res) { ...

    public static String createAnunciar(Request req, Response res){ ...
}
```

Figura 6: código classe AnunciarController

## 4.6 Class CadastroController

Esta classe está contida dentro do package **CONTROLLER** e tem por objetivo enviar o usuário para a página de cadastro de usuário. Também é responsável por criar um usuário.

```
package controller;

import static spark.Spark.*;
import java.util.HashMap;
import java.util.Map;
import model.UsuarioModel;
import spark.Request;
import spark.Response;
import spark.template.thymeleaf.ThymeleafTemplateEngine;
import repository.CadastroRepository;

public class CadastroController {

    public static String getCadastroPage(Request req, Response res) { ...

    public static String createCadastro(Request req, Response res){ ...
}
```

Figura 7: código classe CadastroController

## 4.7 Class IndexController

Esta classe verifica se usuário está logado, se estiver, envia para a página principal, caso não esteja, redireciona para página de login. Como os anteriores, está contido no package **CONTROLLER**.

```
package controller;

import static spark.Spark.*;
import java.util.HashMap;
import java.util.Map;
import service.IsLogged;
import spark.Request;
import spark.Response;
import spark.template.thymeleaf.ThymeleafTemplateEngine;

public class IndexController {

    public static String getIndexPage(Request req, Response res) { ...
}
```

Figura 8: código da classe IndexController

## 4.8 Class LoginController

Esta classe verifica se usuário já está logado, se não estiver redireciona o usuário para a página de login. Também é responsável por efetuar o logout do sistema. Está inserida no package **CONTROLLER**.

```
package controller;

import static spark.Spark.*;
import java.io.IOException;
import java.sql.SQLException;
import java.util.HashMap;
import java.util.Map;
import java.util.UUID;
import model.UsuarioModel;
import spark.Request;
import spark.Response;
import spark.template.thymeleaf.ThymeleafTemplateEngine;
import repository.LoginRepository;
import service.IsLogged;

public class LoginController {

    public static String getLoginPage(Request req, Response res) { ...

    public static String loginPage(Request req, Response res) throws SQLException, IOException { ...

    public static String logoutPage(Request req, Response res) throws SQLException, IOException { ...
}
```

Figura 9: código da classe LoginController



## 4.9 Class MeusAnunciosController

Esta classe verifica se o usuário está logado, se sim redireciona para a página meus anúncios, onde poderá visualizar/editar todos os anúncios feitos pelo usuário. Está no package **CONTROLLER**.

```
package controller;

import static spark.Spark.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import model.PropostaModel;
import model.UsuarioPropostaModel;
import repository.MeusAnunciosRepository;
import service.IsLogged;
import spark.Request;
import spark.Response;
import spark.template.thymeleaf.ThymeleafTemplateEngine;

public class MeusAnunciosController {

    public static String getMeusAnunciosPage(Request req, Response res) { ...

    public static String getEditarMeusAnunciosPage(Request req, Response res){ ...

    public static String editAnuncio(Request req, Response res){ ...

}
```

Figura 10: código da classe MeusAnunciosController

## 4.10 Class MeuPerfilController

Esta classe verifica se o usuário está logado e se estiver direciona para a página do usuário onde ele poderá editar seu perfil. Está inserido no package **CONTROLLER**.

```
package controller;

import static spark.Spark.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import model.UsuarioModel;
import service.IsLogged;
import spark.Request;
import spark.Response;
import spark.template.thymeleaf.ThymeleafTemplateEngine;
import repository.MeuPerfilRepository;

public class MeuPerfilController {

    public static String getMeuPerfilPage(Request req, Response res) {

    public static String editPerfil(Request req, Response res){ ...

}
```

Figura 11: código classe MeuPerfilController

## 4.11 Class ProcurarController

Esta classe verifica se o usuário está logado e se estiver, direciona para a página de procurar anúncios, onde estão os produtos/serviços ativos já anunciados no sistema. Estes anúncios podem ser visualizados e caso o usuário deseje, pode ser feita uma oferta para o mesmo. Está inserido no package **CONTROLLER**.

```
package controller;

import static spark.Spark.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import model.PropostaModel;
import model.UsuarioPropostaModel;
import repository.ProcurarRepository;
import service.IsLogged;
import spark.Request;
import spark.Response;
import spark.template.thymeleaf.ThymeleafTemplateEngine;

public class ProcurarController {

    public static String getProcurarAnunciosPage(Request req, Response res) { ...
    public static String selectAnuncios(Request req, Response res) { ...
    public static String getAnuncioPage(Request req, Response res) { ...
    public static String getPropostaAnuncioPage(Request req, Response res) { ...
    public static String createPropostaAnuncioPage(Request req, Response res) { ...
}
```

Figura12: código da classe ProcurarController

## 4.12 Class PropRealizadas

Esta classe verifica se usuário está logado, se sim redireciona para a página das propostas realizadas, podendo editar propostas já existentes. Está contido no package **CONTROLLER**.

```
package controller;

import static spark.Spark.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import model.PropostaModel;
import model.UsuarioPropostaModel;
import repository.PropRealizadasRepository;
import service.IsLogged;
import spark.Request;
import spark.Response;
import spark.template.thymeleaf.ThymeleafTemplateEngine;

public class PropRealizadasController {

    public static String getPropRealizadasPage(Request req, Response res) { ...

    public static String getEditarPropRealizadasPage(Request req, Response res) { ...

    public static String editProposta(Request req, Response res){ ...
}
```

Figura 13: código PropRealizadasController

## 4.13 Class PropRecebidasController

Esta classe Contida no package **CONTROLLER** e tem por finalidade verificar se o usuário está logado, então redireciona para a página de propostas recebidas, onde se pode ver, aceitar ou recusar as propostas.

```
package controller;

import static spark.Spark.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import model.UsuarioPropostaModel;
import repository.PropRecebidasRepository;
import service.IsLogged;
import spark.Request;
import spark.Response;
import spark.template.thymeleaf.ThymeleafTemplateEngine;

public class PropRecebidasController {

    public static String getPropRecebidasPage(Request req, Response res) { ...

    public static String getVisualizarPropRecebidasPage(Request req, Response res) { ...

    public static String acceptPropRecebidas(Request req, Response res){ ...
}
```

Figura 14: código da classe PropRecebidasController

#### 4.14 Class SobreController

Esta classe verifica se usuário está logado e redireciona para a página sobre, que contém as principais informações sobre o projeto. Contida no package **CONTROLLER**.

```
package controller;

import static spark.Spark.*;
import java.util.HashMap;
import service.Islogged;
import spark.Request;
import spark.Response;
import spark.template.thymeleaf.ThymeleafTemplateEngine;

public class SobreController {

    public static String getSobrePage(Request req, Response res) {...
```

Figura 15: Código classe SobreController

#### 4.15 Class AnunciarRepository

Esta classe está contida no package **REPOSITORY** e tem por finalidade a conexão com o banco de dados. Ela possui dois métodos, que é o de inserir um anúncio no banco e o de buscar o fkcodusuario que vai ser inserido junto no cadastro do anúncio.

```
package repository;

import java.io.IOException;
import java.sql.*;
import model.PropostaModel;
import service.DatabaseConnection;

public class AnunciarRepository {

    public boolean insertAnuncio(PropostaModel anuncioInfo) throws SQLException, IOException {...

    public int selectFkCodUsuario(String id) throws SQLException, IOException {...

}
```

Figura 16: código da classe AnunciarRepository

#### 4.16 Class CadastroRepository

Esta classe conecta com o banco de dados e tem a função de cadastrar usuários no BD. Pertence ao package **REPOSITORY**.

```
package repository;

import java.io.IOException;
import java.sql.*;
import model.UsuarioModel;
import service.DatabaseConnection;

public class CadastroRepository {

    public boolean insertCadastro(UsuarioModel cadastroInfo) throws SQLException, IOException { ...
    }
}
```

Figura 17: código da Classe CadastroRepository

#### 4.17 Class LoginRepository

Classe que pertence ao package **REPOSITORY** e tem a função de conectar ao banco de dados para verificar um login e senha e dar ou não acesso ao sistema.

```
package repository;

import java.io.IOException;
import java.sql.*;
import model.UsuarioModel;
import service.DatabaseConnection;

public class LoginRepository {

    public int doLogin(UsuarioModel loginInfo) throws SQLException, IOException { ...
    }
}
```

Figura 18: código da classe LoginRepository



#### 4.18 Class MeusAnunciosRepository

Classe responsável por conectar ao banco de dados e trazer uma lista contendo os anúncios existentes do usuário e alterar os mesmos individualmente caso necessário. Pertence ao package **REPOSITORY**.

```
package repository;

import java.io.IOException;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import model.PropostaModel;
import service.DatabaseConnection;

public class MeusAnunciosRepository {

    public ArrayList selectPropostas(String id) throws SQLException, IOException { ...

    public ArrayList selectProposta(String pkcodproposta, String id) throws SQLException, IOException { ...

    public boolean updateProposta(PropostaModel propostaInfo) throws SQLException, IOException { ...
}
```

Figura 19: código classe MeusAnunciosRepository

#### 4.19 Class MeuPerfilRepository

Essa classe tem a função de conectar o banco de dados e trazer as informações do perfil, bem com alterá-lo quando necessário. Pertence ao package **REPOSITORY**.

```
package repository;

import java.io.IOException;
import java.sql.*;
import java.util.ArrayList;
import model.UsuarioModel;
import service.DatabaseConnection;

public class MeuPerfilRepository {

    public ArrayList selectInfosPerfil(String id) throws SQLException, IOException { ...

    public boolean updatePerfil(UsuarioModel perfilInfo) throws SQLException, IOException { ...
}
```

Figura 20: código da classe MeuPerfilRepository

## 4.20 Class ProcurarRepository

Esta classe tem a finalidade de conectar ao banco de dados e buscar as informações de produtos e serviços cadastrados, fazer uma proposta para um produto/serviço cadastrado e de buscar o fkcodusuario que vai ser inserido junto no cadastro da proposta. Está dentro do package **REPOSITORY**.

```
package repository;

import java.io.IOException;
import java.sql.*;
import java.util.ArrayList;
import model.PropostaModel;
import service.DatabaseConnection;

public class ProcurarRepository {

    public ArrayList selectPropostas(String procurar, String id) throws SQLException, IOException { ...

    public ArrayList selectProposta(String pkcodproposta, String id) throws SQLException, IOException { ...

    public boolean insertProposta(PropostaModel anuncioInfo, String fkcodproposta1) throws SQLException, IOException { ...

    public int selectFkCodUsuario(String id) throws SQLException, IOException { ...

}
```

Figura 21: código da classe ProcurarRepository

## 4.21 Class PropRealizadasRepository

Esta classe, pertencente ao package **REPOSITORY**, tem a função de conectar ao banco de dados e buscar as propostas realizadas ou editar existentes.

```
package repository;

import java.io.IOException;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import model.PropostaModel;
import service.DatabaseConnection;

public class PropRealizadasRepository {

    public ArrayList selectPropostas(String id) throws SQLException, IOException { ...

    public ArrayList selectProposta(String pkcodproposta, String id) throws SQLException, IOException { ...

    public boolean updateProposta(PropostaModel propostaInfo) throws SQLException, IOException { ...

}
```

Figura 22: código da classe PropRealizadasRepository

## 4.22 Class PropRecebidasRepository

Esta classe tem por finalidade conectar ao banco de dados e fazer a busca de todas as propostas recebidas dos anúncios e verificar se aceita ou não a determinada proposta. Pertence ao package **REPOSITORY**.

```
package repository;

import java.io.IOException;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import service.DatabaseConnection;

public class PropRecebidasRepository {

    public ArrayList selectPropostas(String id) throws SQLException, IOException { ...

    public ArrayList selectProposta(String pkcodproposta) throws SQLException, IOException { ...

    public boolean updateProposta(String pkcodproposta, int aceite) throws SQLException, IOException { ...
}
```

Figura 23: código da classe PropRecebidasRepository

## 4.23 Class ServiceRepository

Esta classe do package Repository tem por finalidade fazer a conexão com o banco de dados e fornecer a função que verifica se um usuário já está logado no sistema. Também fornece a função que é utilizada na classe ThreadModerador, que verifica por anúncios com palavras proibidas.

```
package repository;

import java.io.IOException;
import java.sql.*;
import service.DatabaseConnection;

public class ServiceRepository {

    public Boolean verifyLogin(String id) throws SQLException, IOException { ...

    public void verifyPalavra() throws SQLException, IOException { ...
}
```

Figura 24: código da classe ServiceRepository



#### 4.24 Class IndexRepository

Esta classe pertence ao package Repository e como não faz acesso ao banco de dados, não possui métodos.

```
1 package repository;
2
3 public class IndexRepository {
4
```

Figura 25: código da classe IndexRepository

#### 4.25 Class DatabaseConnection

Classe pertencente ao package SERVICE e tem por finalidade fornecer a conexão com o banco de dados. Dentro desta classe também existe uma função que captura as informações da conexão (login, senha, etc.) de um arquivo chamado Database.properties.

```
package service;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;
import java.sql.*;

public class DatabaseConnection {

    private Properties getProps() throws IOException { ...

    public Statement Executor() throws IOException, SQLException { ...
}
}
```

Figura 26: código da classe DatabaseConection

#### 4.26 Class IsLogged

Classe do package SERVICE responsável pelo serviço de verificação de existência do usuário e se o mesmo está logado.

```
package service;

import java.io.IOException;
import java.sql.SQLException;
import repository.ServiceRepository;

public class IsLogged {

    public Boolean isLogged(String id) throws SQLException, IOException {

        ServiceRepository sr = new ServiceRepository();
        return sr.verifyLogin(id);
    }
}
```

Figura 27: código da classe IsLogged

## 4.27 Class Router

Esta classe é a única pertencente ao package ROUTER e tem um método que contém todas as rotas da aplicação. Ela realiza o gerenciamento de todas essas rotas, sendo uma espécie de maestro, orquestrando o que cada rota deve fazer baseado no que lhe foi solicitado.

```
package routes;

import static spark.Spark.*;
import controller.LoginController;
import controller.MeuPerfilController;
import controller.MeusAnunciosController;
import controller.ProcurarController;
import controller.PropRealizadasController;
import controller.PropRecebidasController;
import controller.SobreController;
import controller.AnunciarController;
import controller.CadastroController;
import controller.IndexController;

public class Router {

    public void getAllRoutes(){...}
}
```

Figura 28: código da classe Router

## 4.28 Class ThreadModerador

Esta classe implementa Runnable e atua como uma Thread que é executada quando a aplicação inicializa e dura enquanto a aplicação está rodando. Ela realiza uma varredura de tempos em tempos em todos os anúncios do sistema, buscando por palavras proibidas, palavras estas que são cadastradas em uma tabela no banco de dados. Caso encontre uma palavra proibida no anúncio, ela inativa o mesmo.

```
package service;

import java.io.IOException;
import java.sql.SQLException;
import repository.ServiceRepository;

public class ThreadModerador implements Runnable {

    @Override
    public void run() {...}
}
```

Figura 29: código da classe Router

## **5. PACKAGES**

A aplicação foi separada em camadas, utilizando os princípios do MVC com mais algumas abstrações como: Repository, Service e Router, logo, podemos dizer que a aplicação tem 6 camadas.

### **5.1 MODEL**

Este package contém classes única e exclusivamente com atributos e os seus métodos getters e setters, representando a ideia da aplicação.

### **5.2 CONTROLLER**

Este package contém todos os controllers da aplicação, que fazem a união entre repository e models, repassando as informações para os templates.

### **5.3 ROUTER**

Responsável por distribuir todas as requisições recebidas para os devidos controllers

### **5.4 REPOSITORY**

Neste package as suas classes tem a função de manipulação de dados do banco de dados.

### **5.5 SERVICE**

Este package é responsável por tudo que não faz parte diretamente da aplicação, como por exemplo, fornecer as informações e a conexão do banco de dados, o serviço da ThreadModerador, etc.

## 5.6 TEMPLATES

Segue a sequência de todas as interfaces desenvolvidas em HTML, do projeto.

### 5.6.1 INDEX

Tela inicial do projeto



Figura 31: tela inicial do projeto

### 5.6.2 MEU PERFIL

Tela de cadastro e edição de perfil

A imagem mostra a interface de usuário para o cadastro de um novo usuário. No topo, há uma barra de boas-vindas com o logo de uma mão fazendo o sinal de vitória à esquerda e o texto 'BEM VINDO AO BRIQUE DA GURIZADA!'. Abaixo, há uma série de campos de entrada para: 'Nome' (com o placeholder 'Informe seu nome completo'), 'CPF' (com o placeholder 'Informe seu CPF (Somente números)'), 'Telefone' (com o placeholder 'Informe seu telefone (Somente números)'), 'Login' (com o placeholder 'Cadastre um novo login') e 'Senha' (com o placeholder 'Cadastre uma senha'). Um botão azul 'Confirmar cadastro' está posicionado abaixo dos campos. Na base, há uma mensagem de aviso: 'Certifique-se que os dados inseridos estão corretos antes de confirmar o cadastro!'.

Figura 32: Tela de cadastro de usuário

### 5.6.3 LOGIN

Tela de entrada no sistema

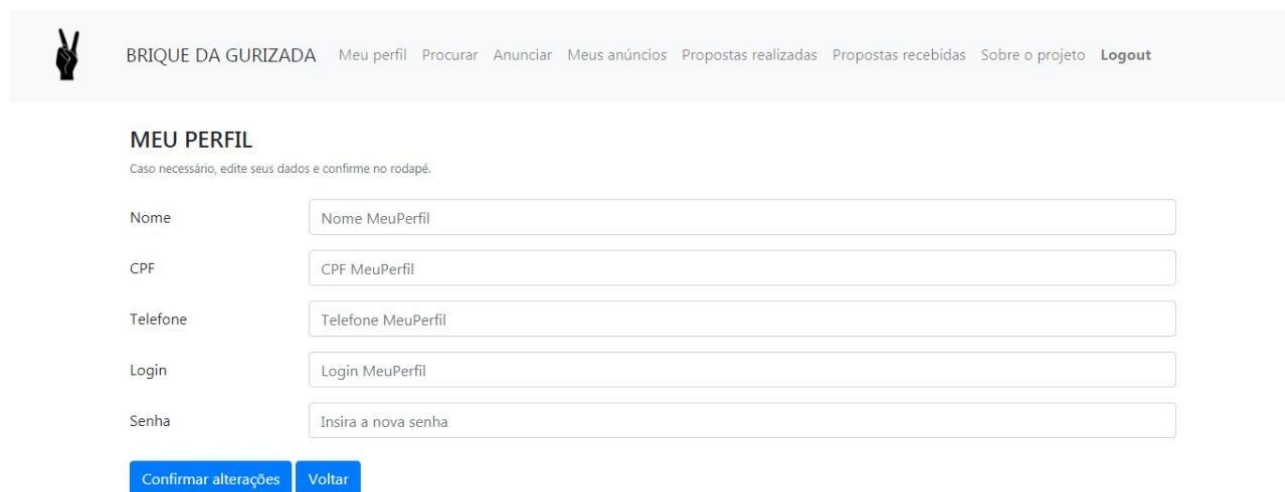


The login screen features a light gray header with a hand icon making a peace sign and the text "BEM VINDO AO BRIQUE DA GURIZADA!". Below the header, the word "Login" is displayed. There are two input fields: "Insira seu login" and "Insira sua senha". A blue button labeled "LOGIN" is positioned below the password field. Underneath the button, there is a link "Ainda não tem cadastro? Clique AQUI" and a warning "Não compartilhe seus dados com outras pessoas!".

Figura 33: tela de login do sistema

### 5.6.4 CADASTRO USUÁRIO

Tela para cadastrar as informações de usuários ou alterar os dados inseridos anteriormente.

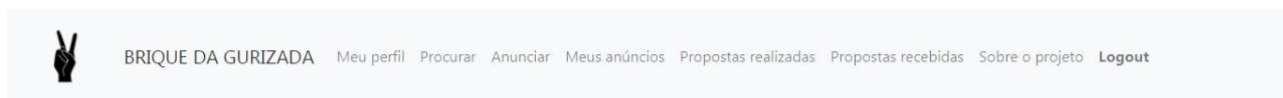


The user registration/edit profile screen has a light gray header with a hand icon making a peace sign and the text "BRIQUE DA GURIZADA". To the right of the header are navigation links: "Meu perfil", "Procurar", "Anunciar", "Meus anúncios", "Propostas realizadas", "Propostas recebidas", "Sobre o projeto", and "Logout". Below the header, the section "MEU PERFIL" is shown with a subtext "Caso necessário, edite seus dados e confirme no rodapé.". There are five input fields: "Nome" (containing "Nome MeuPerfil"), "CPF" (containing "CPF MeuPerfil"), "Telefone" (containing "Telefone MeuPerfil"), "Login" (containing "Login MeuPerfil"), and "Senha" (containing "Insira a nova senha"). At the bottom, there are two blue buttons: "Confirmar alterações" and "Voltar".

Figura 34: Tela cadastro usuário

## 5.6.5 ANUNCIAR

### Tela de cadastro de anúncios



#### NOVO ANÚNCIO

Informe os dados para seu novo anúncio.

Título

Tipo de anúncio

Serviço

Descrição do anúncio

Detalhe as condições do anúncio

Confirmar anuncio

Certifique-se que os dados inseridos estão corretos antes de confirmar o anúncio!

Figura 35: tela cadastro de anúncio

## 5.6.6 EDITAR ANÚNCIO

### Tela para modificar anúncio existente



#### EDITAR ANÚNCIO

Caso necessário, edite seu anúncio e confirme no rodapé.

Título

título Anuncio

Tipo de anúncio

Escolha um valor...

Descrição do anúncio

Datalhe as condições do anúncio


Confirmar alterações

Voltar

Figura 36 Tela de edição de anúncio

## 5.6.7 PROCURAR ANÚNCIO

Tela para buscar algum produto ou serviço que esteja precisando

 BRIQUE DA GURIZADA [Meu perfil](#) [Procurar](#) [Anunciar](#) [Meus anúncios](#) [Propostas realizadas](#) [Propostas recebidas](#) [Sobre o projeto](#) [Logout](#)

### PROCURAR ANÚNCIOS

Pesquise um anúncio inserindo uma palavra chave específica.

[Procurar](#)

Código	Anunciante	Tipo	Título do anúncio	Detalhamento do anúncio	Opção
pkcodanuncio	nome	Objeto/Serviço	titulo	descricao	<a href="#">Visualizar anúncio</a>

[Voltar](#)

Figura 37: Tela procurar anúncios

## 5.6.8 PROPOSTAS RECEBIDAS

Tela para visualizar a relação de propostas recebidas

 BRIQUE DA GURIZADA [Meu perfil](#) [Procurar](#) [Anunciar](#) [Meus anúncios](#) [Propostas realizadas](#) [Propostas recebidas](#) [Sobre o projeto](#) [Logout](#)

### PROPOSTAS RECEBIDAS

Listagem atualizada.

Código	Status	Proponente	Tipo	Título	Descrição	Anúncio	Código	Opção
pkcodanuncio	status	Proponente	Tipo	titulo	descricao	anuncio	codigo	<a href="#">Visualizar</a>

[Voltar](#)

Figura 38: tela propostas recebidas

## 5.6.9 VISUALIZAR PROPOSTAS RECEBIDAS

Tela que detalha uma proposta em específico

A interface apresenta o cabeçalho 'BRIQUE DA GURIZADA' com links para 'Meu perfil', 'Procurar', 'Anunciar', 'Meus anúncios', 'Propostas realizadas', 'Propostas recebidas', 'Sobre o projeto' e 'Logout'. O título principal é 'VISUALIZAR PROPOSTA RECEBIDA' com o subtítulo 'Você pode aceitar, recusar ou decidir mais tarde.'.

Os campos de formulário são:

- Anunciante:** Campo de texto com o placeholder 'Carrega aqui nome do anunciante'.
- Telefone:** Campo de texto com o placeholder 'Carrega aqui telefone do anunciante'.
- Título do anúncio:** Campo de texto com o placeholder 'Carrega aqui título'.
- Tipo de anúncio:** Campo de texto com o placeholder 'Carrega aqui tipo de anuncio (Serviço/Objeto)'.
- Descrição do anúncio:** Área de texto grande com o placeholder 'Datalhe das condições do anúncio ofertado'.

Na base, há um seletor 'Aceitar proposta' com a opção 'Selecione' e um botão 'Confirmar' em azul. Um botão 'Voltar' em azul está localizado abaixo do seletor.

Figura 39: tela Visualizar Proposta Recebida

## 5.6.10 EDITAR PROPOSTA

Tela com função de modificar uma proposta existente

A interface apresenta o cabeçalho 'BRIQUE DA GURIZADA' com links para 'Meu perfil', 'Procurar', 'Anunciar', 'Meus anúncios', 'Propostas realizadas', 'Propostas recebidas', 'Sobre o projeto' e 'Logout'. O título principal é 'EDITAR PROPOSTA' com o subtítulo 'Caso necessário, edite sua proposta e confirme no rodapé.'.

Os campos de formulário são:

- Título:** Campo de texto com o valor 'título Anuncio'.
- Tipo de anúncio:** Campo de texto com o placeholder 'Escolha um valor...' e uma seta para baixo.
- Descrição da proposta:** Área de texto grande com o placeholder 'Datalhe as condições do anúncio'.


Na base, há dois botões em azul: 'Confirmar alterações' e 'Voltar'.

Figura 40: tela editar proposta



## 5.6.11 PROPOSTAS REALIZADAS

Tela para visualizar as propostas que o próprio usuário realizou.



**PROPOSTAS REALIZADAS**  
Listagem atualizada.

Código	Status	Proponente	Tipo	Título	Descrição	Anúncio	Código	Opção
pkcodanuncio	status	nome	Objeto/Serviço	titulo	descricao	anuncio	anuncio	<a href="#">Editar</a>

[Voltar](#)

Figura 41: Tela propostas realizadas

## 5.6.12 SOBRE

Tela informativa sobre os dados do projeto



**SOBRE O PROJETO**

**RECURSOS**  
Framework Spark, Oracle XE, Github, Trello

**GRUPO DESENVOLVEDOR**  
Charles Lemos, Giovani Lima, Antonio Rolim, Alex Córdova

**DISCIPLINA ACADEMICA**  
Programação orientada a objetos

**INSTITUIÇÃO DE ENSINO**  
Universidade Lutera do Brasil ULBRA

**ENTREGA DO PROJETO**  
03 de Dezembro de 2020

Figura 42: Tela Sobre

## 6. DADOS DE CONEXÃO DO DADOS DO BANCO

Esta configuração é pessoal e cada pessoa tem a sua, mas segue um exemplo na imagem abaixo.

```
prop.server = ip
prop.port = 1521
prop.database = xe
prop.user = user
prop.password = senha
```

Figura 43: dados do SGDB

## 7. GITHUB

Este projeto está armazenado no GitHub(<https://github.com/everthefullstack/trabalho-as>), e disponível para quem quiser utilizar, incluindo um README.ME, onde explica como baixar e instalar para que funcione perfeitamente.

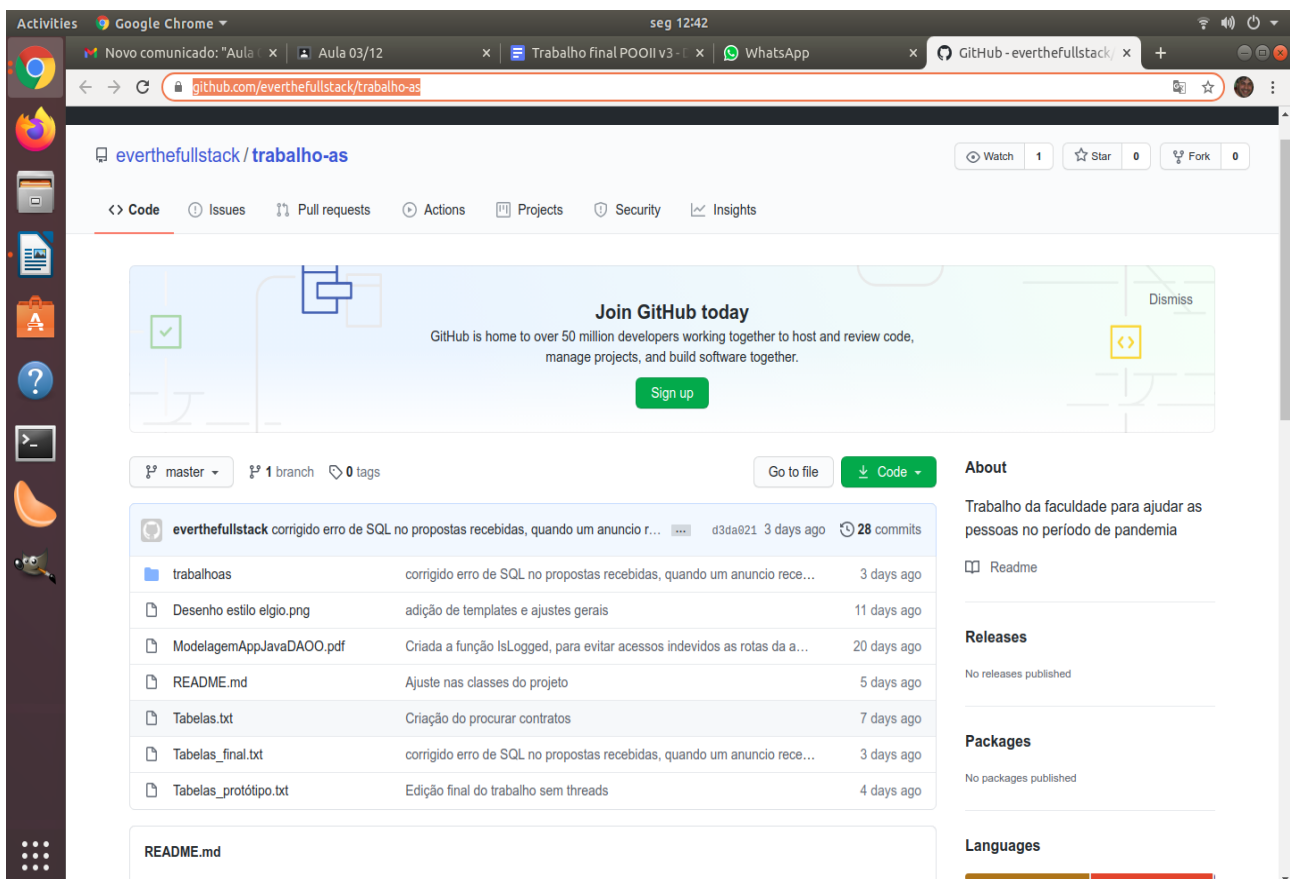


Figura 44 - Tela local de armazenamento no GITHUB

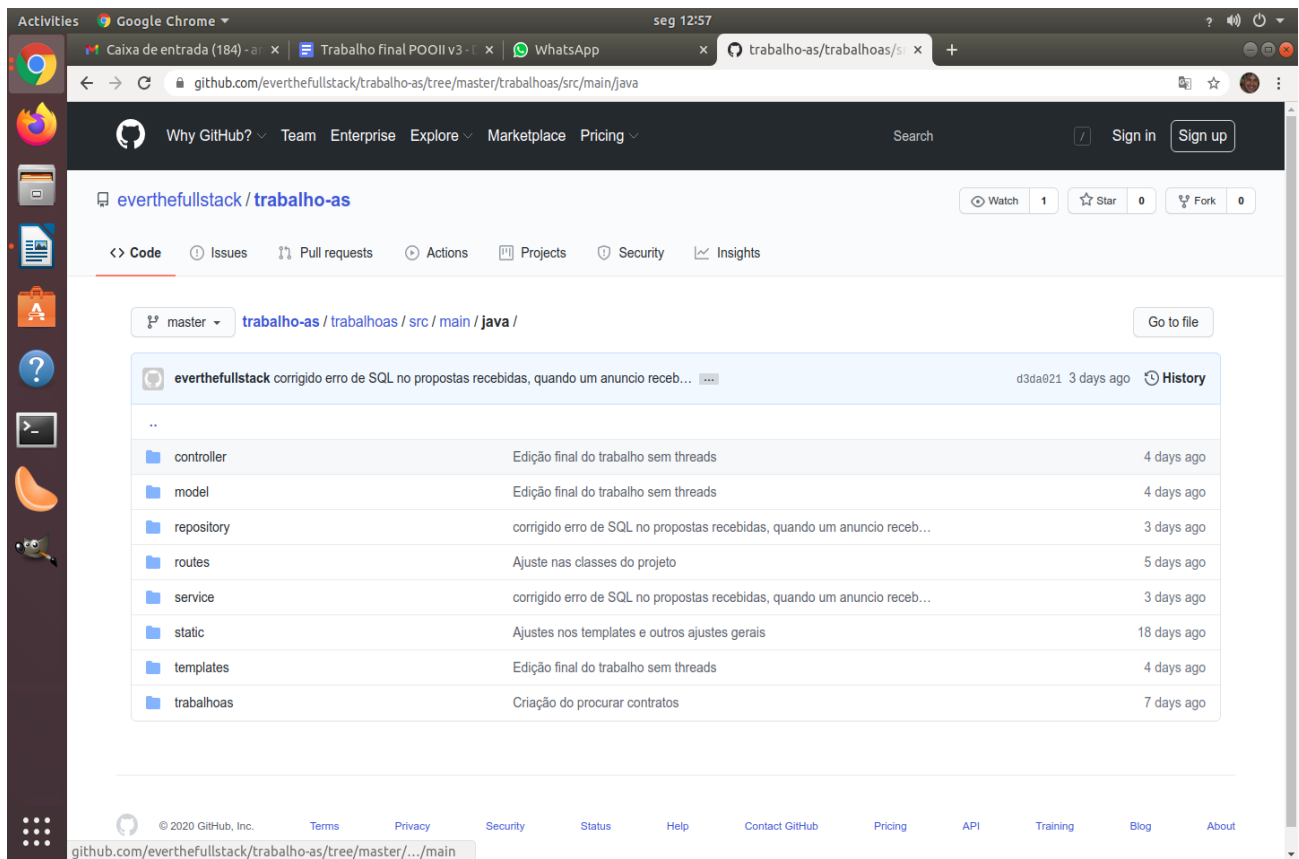


Figura 45 - Local de armazenamento dos packages

## **8. CONSIDERAÇÕES FINAIS**

O sistema realizado mostrou todo o conjunto de conhecimentos adquiridos durante a cadeira de POO II, e agregando os obtidos previamente em POO I. Mostrou ainda as vantagens de trabalho em equipe, onde podemos explorar as potencialidades de cada integrante onde possuía maior conhecimento para que pudessem auxiliar os demais onde possuíam dificuldades, traduzindo tudo isso num processo agradável, criativo e produtivo. O sistema foi realizado através de muitas conversas via WhatsApp e quando necessário videoconferências através do Google Meet, onde visualizamos o andamento do mesmo, bem como tiramos dúvidas e sugerimos mudanças conforme as necessidades se apresentavam.

Neste projeto priorizamos colocar o máximo dos conceitos aprendidos, desde os mais simples como utilização de padrões de arquitetura, passando por threads, heranças e tratamento de exceções até chegarmos em interfaces gráficas(frontend) e integração com o Banco de dados.

Concluimos que este trabalho foi de grande valia para o aprendizado do desenvolvimento de projetos orientados a componentes, com ênfase em Java, bem como a integração com outras tecnologias.