Relatório do Trabalho de Computação III - 2019.1

Caio César Alves Sampaio – 2016780061 - caio.cesarsampaio@hotmail.com
Ingrid Cardin da Costa - 2016780231 - i.cardin.c@gmail.com
Fábio Júnior Miranda - 2014780178 - fjumisan@gmail.com

Índice:

1.Padrões do grupo

Os padrões utilizados foram o Módulo de Tabela para a camada de domínio e o Mapeador de Dados para a camada de dados.

2.Instruções

- Atenção: É preciso modificar na classe DataMapper.java o endereço do banco de dados de sua máquina para usar o sistema.
- Atenção 2: Atente-se ao caminho dos JARs do TomCat e do Junit no BuildPath do eclipse.

Primeiro é necessário executar o projeto no eclipse com o Tomcat configurado e sendo executado.

A primeira página a ser acessada para se utilizar o sistema é a index.jsp, que é selecionada automaticamente quando é acessado o endereço http://localhost:8080/Comput3/ (em que Comput3 é o nome do projeto)*. Também é possível usar o sistema ao executar a página pelo eclipse, que abre o seu navegador interno próprio para visualização.

*Requer que o projeto esteja sendo executado pelo eclipse.

No primeiro acesso, recomenda-se seguir os seguintes passos:

1. Selecionar o ícone de engrenagem (configurações) que é a última opção selecionável na index.jsp;
2. Um formulário aparecerá. Escrever a senha de super usuário “admin” e selecionar “Entrar”;
3. Selecionar a opção “Excluir tabelas”;
4. Haverá um redirecionamento para a primeira página;
5. Repetir os passos 1 e 2;
6. Selecionar a opção “Criar tabelas”;
7. Haverá um redirecionamento para a primeira página;

Esses passos do primeiro acessos são recomendados para que não haja nenhuma incompatibilidade com os dados das tabelas anteriores que possam existir em seu banco de dados.

➢ Para cadastrar um usuário:

1. Selecione “Cadastre-se”;
2. Preencha os campos Nome, CPF(válido) e Senha(Alfanumérica de 6 dígitos);
3. Selecione “Cadastrar”.

➢ Para cadastrar um Administrador:

1. Selecione “Cadastrar Administrador”;
2. Preencha os campos Código de super usuário(com o código ”admin”), Nome, CPF(válido) e Senha(Alfanumérica de 6 dígitos);
3. Selecione “Cadastrar”.

➢ Para criar uma solicitação de novo Museu:

1. Selecione “Criar solicitação de novo museu”;
2. Preencha todos os campos. Nome, Cidade e Estado só podem possuir letras. Data de criação deve ser no formato dd/mm/aaaa. CPF deve ser válido, além da senha ser alfanumérica de 6 dígitos;
3. Selecione “Solicitar”.

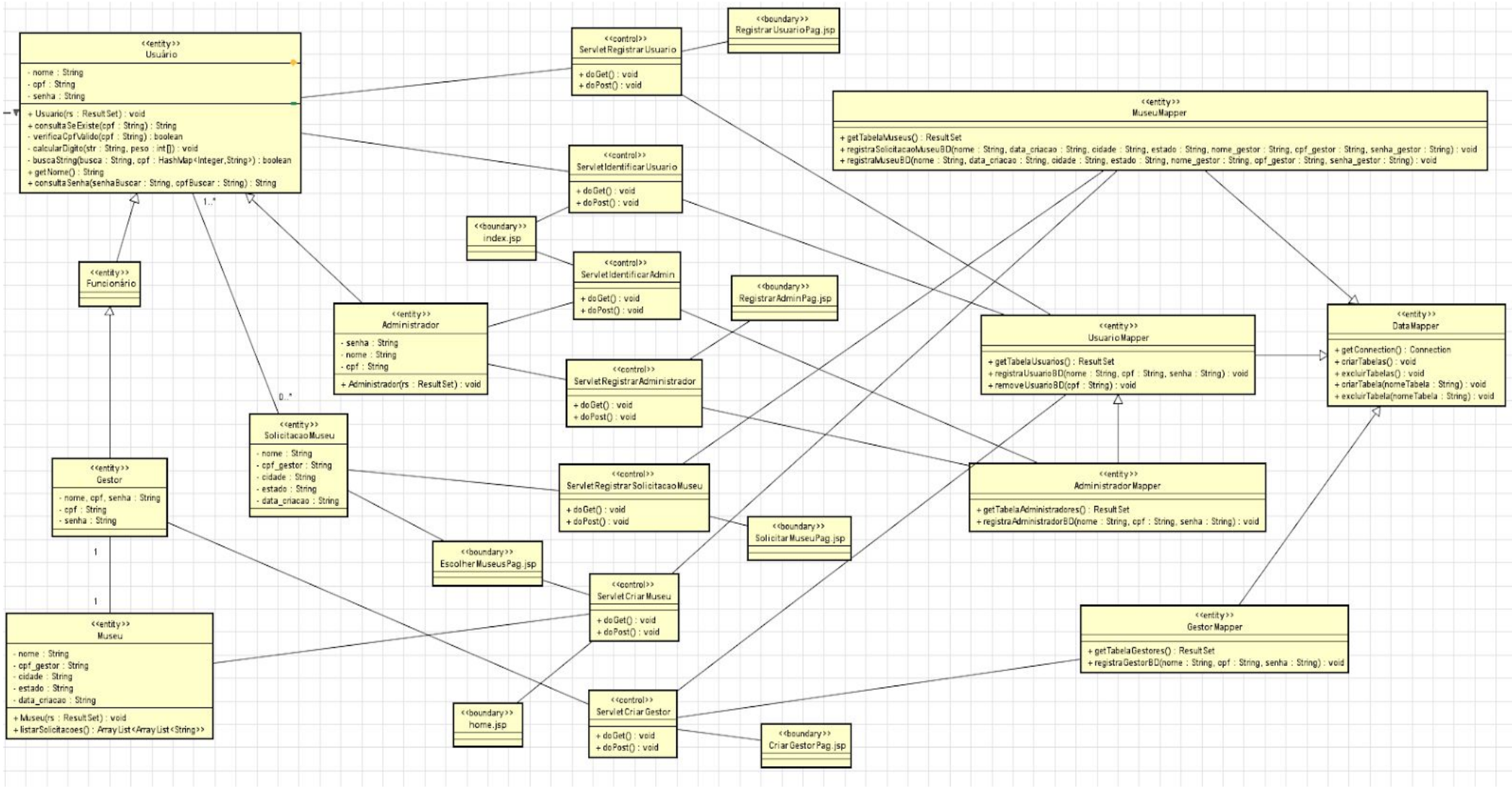
➢ Para criar um novo museu e um gestor para o museu:

1. Se identifique como Administrador preenchendo os campos CPF e Senha da página inicial e selecione “Entrar”;
2. Selecione “Criar novo museu”;
3. Uma lista com as solicitações de museu existentes será exibida;
4. Selecione uma solicitação;
5. Um formulário será exibido. Preencha os campos com o Nome do gestor (somente letras), CPF do gestor(CPF válido) e Senha do gestor(alfanumérico 6 dígitos);
6. Selecione “Confirma”;
7. Selecione “Confirma criação do museu”.

Informações úteis:

- Nome das tabelas do banco de dados: usuarios, administradores, gestores, museus.

3.Diagrama de classes (disponível em melhor qualidade em .astah)


«entity»
Usuário
- nome : String
- cpf : String
- senha : String
+ Usuario(rs : ResultSet) : void
+ consultaSeExiste(cpf : String) : String
- verificaCpfValido(cpf : String) : boolean
- calcularDigito(str : String, peso : int[]) : void
- buscaString(busca : String, cpf : HashMap<Integer,String>) : boolean
+ getNome() : String
+ consultaSenha(senhaBuscar : String, cpfBuscar : String) : String
«control»
ServletRegistrarUsuario
+ doGet() : void
+ doPost() : void
«boundary»
RegistrarUsuarioPag.jsp
«control»
ServletIdentificarUsuario
+ doGet() : void
+ doPost() : void
«boundary»
index.jsp
«control»
ServletIdentificarAdmin
+ doGet() : void
+ doPost() : void
«boundary»
RegistrarAdminPag.jsp
«entity»
MuseuMapper
+ getTabelaMuseus() : ResultSet
+ registraSolicitacaoMuseuBD(nome : String, data_criacao : String, cidade : String, estado : String, nome_gestor : String, cpf_gestor : String, senha_gestor : String) : void
+ registraMuseuBD(nome : String, data_criacao : String, cidade : String, estado : String, nome_gestor : String, cpf_gestor : String, senha_gestor : String) : void
«entity»
UsuarioMapper
+ getTabelaUsuarios() : ResultSet
+ registraUsuarioBD(nome : String, cpf : String, senha : String) : void
+ removeUsuarioBD(cpf : String) : void
«entity»
DataMapper
+ getConnection() : Connection
+ criarTabelas() : void
+ excluirTabelas() : void
+ criarTabela(nomeTabela : String) : void
+ excluirTabela(nomeTabela : String) : void
«entity»
Funcionário
«entity»
Administrador
- senha : String
- nome : String
- cpf : String
+ Administrador(rs : ResultSet) : void
«control»
ServletRegistrarAdministrador
+ doGet() : void
+ doPost() : void
«entity»
AdministradorMapper
+ getTabelaAdministradores() : ResultSet
+ registraAdministradorBD(nome : String, cpf : String, senha : String) : void
«entity»
SolicitacaoMuseu
- nome : String
- cpf_gestor : String
- cidade : String
- estado : String
- data_criacao : String
«control»
ServletRegistrarSolicitacaoMuseu
+ doGet() : void
+ doPost() : void
«boundary»
SolicitarMuseuPag.jsp
«entity»
Gestor
- nome, cpf, senha : String
- cpf : String
- senha : String
«boundary»
EscolherMuseusPag.jsp
«control»
ServletCriarMuseu
+ doGet() : void
+ doPost() : void
«entity»
Museu
- nome : String
- cpf_gestor : String
- cidade : String
- estado : String
- data_criacao : String
+ Museu(rs : ResultSet) : void
+ listarSolicitacoes() : ArrayList<ArrayList<String>>
«entity»
GestorMapper
+ getTabelaGestores() : ResultSet
+ registraGestorBD(nome : String, cpf : String, senha : String) : void
«boundary»
home.jsp
«control»
ServletCriarGestor
+ doGet() : void
+ doPost() : void
«boundary»
CriarGestorPag.jsp
1..*
0..*
1
1

4.Testes

Foram feitos os seguintes testes no projeto:

- Unitários:

- Atenção: Para não fazer alterações na classe testado original, foram criadas novas classes idênticas a mesma, porém recebendo no construtor um Mock Object, ao invés de um resultSet.

- Atenção: Para fazer alguns testes unitários, foi preciso utilizar um Mock Object para representar o ResultSet:

```
public class MockResultSet {



    private int i=-1;

    String set [][] = {{"Pedro","Joao","Mario"},
                       {"44770273894","456","789"},
                       {"teste1","teste2","teste3"}};
    public boolean next ()
    {

        this.i++;
        if(i<3)
            return true;
        else
            return false;
    }


    public String getString(String teste)
    {
        switch (teste) {
        case "nome":
                return set[0][i];
        case "cpf":
                return set[1][i];
        case "senha":
                return set[2][i];
        default:
                break;


    }

    return null;

    }


}
```

```
public class MockResultSetMuseu {

    private int i=-1;

    String set [][] = {{"Pedro","Joao","Mario"},
                       {"27/06/2019","21/05/2019","20/02/2019"},
                       {"Nova Iguacu","Queimados","Rio de Janeiro"},
                       {"Rio de Janeiro", "Rio de Janeiro","Rio de Janeiro"},
                       {"Fabio", "Caio", "Ingrid"},
                       {"47770273894", "12345678910", "45678901239"},
                       {"123","456","789"},
                       {"1", "1", "0"}
                           };

    public boolean next (){

        this.i++;
        if(i<2)
            return true;
        else
            return false;
    }

    public String getString(String teste){
        switch (teste)
        {
        case "nome":
                return set[0][i];
        case "data_criacao":
                return set[1][i];
        case "cidade":
                return set[2][i];
        case "estado":
                return set[3][i];
        case "nome_gestor":
                return set[4][i];
        case "cpf_gestor":
                return set[5][i];
        case "senha_gestor":
                return set[6][i];
        default:
                break;

        }

    return null;
    }
    public int getInt(String valor){

        if(valor=="solic")
        {
            return Integer.parseInt(set[7][i]);
        }

        return -1;

    }
```

- Método consultaSeExiste() da classe Usuario:
    - Para fazer tal teste, foi necessário instanciar um Mock Object do tipo MockResultSet para simular os dados recebidos pelo ResultSet e também instanciar um objeto da classe TesteUsuario (classe idêntica da classe

Usuario). Após isso foi feito um teste para verificar se o método de consultar se um CPF existe estava funcionando de forma correta.

```
@Test
void testaCPFErro() throws SQLException {


    MockResultSet mock = new MockResultSet();
    TesteUsuario usuario = new TesteUsuario(mock);

    String cpf = "13549993714";
    String condicao = usuario.consultaSeExiste(cpf);

    assertEquals("Nao existe", condicao);


}
```

- Método consultaSenha() da classe Usuario:
    - Para fazer tal teste, foi necessário instanciar um Mock Object do tipo MockResultSet para simular os dados recebidos pelo ResultSet e também instanciar um objeto da classe TesteUsuario (classe idêntica da classe Usuario). Após isso foi feito um teste para verificar se o método de consultar uma senha estava funcionando de forma correta.

```
@Test
void testaSenhaCorreta() throws SQLException{


    MockResultSet mock = new MockResultSet();
    TesteUsuario usuario = new TesteUsuario(mock);
    String cpf = "44770273894";
    String senha = "teste1";
    String condicao = usuario.consultaSenha(cpf, senha);

    assertEquals("Senha correta", condicao);

}

}
```

- Método listarSolicitacoes() da classe Museu:
    - Para fazer tal teste, foi necessário instanciar um Mock Object do tipo MockResultSetMuseu para simular os dados recebidos pelo ResultSet e também instanciar um objeto da classe TesteMuseu (classe idêntica da classe Museu). Após isso foi feito um teste para verificar se o

método de listar as solicitações de museu estava funcionando de forma correta.

```java
import static org.junit.Assert.assertEquals;

import java.sql.SQLException;
import java.util.ArrayList;

import org.junit.jupiter.api.Test;

class MuseuTest {

    @Test
    void test() throws SQLException {

        MockResultSetMuseu mock = new MockResultSetMuseu();
        TesteMuseu museu = new TesteMuseu(mock);

        ArrayList<ArrayList<String>> conjuntoTeste = museu.listarSolicitacoes();
        ArrayList<String> linhaTtens = conjuntoTeste.get(0);


        String nome = linhaTtens.get(0);

        assertEquals("Pedro", nome);

    }
```

- Funcionais:
  - Para o caso de uso Criar Usuário:
    - Tupla: <(nome, cpf, senha), resultado>
      - CT01: <("Fabio","884.777.725-92","1a2b3"), inválido - gerar notificação de senha incorreta>
      - CT02: <("Fabio","884.777.725-00","1a2b3c"), inválido - gerar notificação de verificador do CPF incorreto>
      - CT03: <(" "," "," "), inválido - gerar notificação de campos obrigatórios>
      - CT04: <("Fabio","884.777.725-92","1a2b3~"), inválido - gerar notificação de senha incorreta>
      - CT05: <("Caio","270.658.419-01","11a22b33c"), inválido - gerar notificação de CPF já tem usuário associado>
      - CT06: <("Fabio","884.777.725-92","1a2b3c"), válido>
  - Para o caso de uso Solicitar criação do Museu:
    - Tupla: <(nome, dataCriacao, cidade, estado, nomeGestor, cpfGestor, senhaGestor), resultado>

- CT01: <("Muzeuzin", "23/06/2019", "Nova Iguacu", "RJ", "Caio", "270.658.419-01", "11a22b33c"), válido>
- CT02: <("Muzeuzin", "23/06/2019", "Nova Iguacu", "RJ", " ", "270.658.419-01", "11a22b33c"), inválido>

➔ Para o caso de uso Criar Museu:

◆ Tupla: <(nome, dataCriacao, cidade, cpfGestor), resultado>

- CT01: <(" ", "23/06/2019", "Nova Iguacu", "270.658.419-01"), inválido - gerar notificação de nome do museu não informado>
- CT02: <("Muzeuzin", "23/06/2019", "Nova Iguacu", "270.658.419-01"), válido>

➔ Para o caso de uso Criar Gestor:

◆ Tupla <(nome, cpfGestor), resultado>

- CT01: <("Ingrid", "807.318.520-28"), válido>
- CT02: <("Caio", "270.658.419-01"), válido - gerar notificação de CPF já tem usuário associado>

5. DSSs e Diagramas de Sequência

Consulte o arquivo astah na pasta deste relatório para visualizar os diagramas.