

Relatório de Projecto

Programação Aplicada | Programação IV

Avaliação Periódica

Autor:
Guilherme Rodrigues a2020154390

Data: Maio 2025

Resumo

Este projeto tem como objetivo expandir o trabalho anterior, através da integração de um servidor para comunicação com a base de dados, bem como da implementação de uma nova interface dedicada aos fabricantes.

Palavras-chave

- Base de Dados Relacional;
- Certificação Energética;
- Controle de Acesso;
- Fluxo de Trabalho;
- Gestão de Utilizadores;
- Interface de Texto;
- Java;
- JDBC;
- Notificações;
- Persistência de Dados;
- PowerDesigner;
- PostgreSQL;
- Programação Orientada a Objetos

Índice

Resumo	3
Lista de Figuras.....	7
Lista de Tabelas	9
Lista de Acrónimos.....	11
1. Introdução 13	
2. Estado da Arte 13	
3. Objectivos e Metodologias 15	
3.1. Ferramentas e Tecnologias 15	
3.2. Planeamento 15	
4. Trabalho Desenvolvido 17	
4.1. Requisitos Implementados 17	
4.2. Classes e Packages 18	
4.3. Algoritmos 32	
4.4. Estruturas de Dados 32	
4.5. Armazenamento de Dados 33	
4.6. Procedimentos de Teste 45	
5. Conclusões 47	
5.1. Forças 47	
5.2. Limitações 47	
5.3. Trabalho Futuro 47	
6. Referências 49	
6.1. Lista de Referências 49	

Lista de Figuras

Figura 1: Método Visualizar_informações	17
Figura 2: Método Alterar_Informações 1	19
Figura 3: Método Alterar_Informações 2	20
Figura 4: Método Alterar_Informações 3	21
Figura 5: Método Adicionar_Equipamento	22
Figura 6: Método Listar_Equipamentos	23
Figura 7: Método Pesquisar_Equipamento	24
Figura 8: Método Listar_Certificação	25
Figura 9: Método Pesquisar_Certificação	26
Figura 10: Classe servidor	27
Figura 11: Classe Fabricante_Servidor 1	29
Figura 12: Classe Fabricante_Servidor 2	30
Figura 13: Modelo Entidade-Relacionamento	32
Figura 14: Modelo Físico	33

Lista de Tabelas

Não foi encontrada nenhuma entrada no índice de ilustrações.

Lista de Acrónimos

ER	Modelo Entidade-Relacionamento
SoA	State Of the Art
IDE	Integrated Development Environment
VSCODE	Visual Studio Code

1. Introdução

Este projeto tem como objetivo a integração de um servidor capaz de comunicar com clientes através de sockets. O cliente envia comandos específicos ao servidor, que, por sua vez, interpreta e executa diversas operações, tais como a inserção de novos equipamentos, a listagem dos mesmos, a pesquisa por certificações e a alteração de dados existentes.

Assim neste projeto a base de dados é acessada única e exclusivamente pelo servidor e não pelo o utilizador, como anteriormente.

2. Estado da Arte

Nos sistemas modernos de gestão distribuída, a arquitetura cliente-servidor continua a ser amplamente utilizada devido à sua escalabilidade, flexibilidade e capacidade de centralização de dados. Esta abordagem permite a separação clara entre a camada de apresentação (cliente) e a lógica de negócios e persistência (servidor e base de dados).

Tecnologias como Java Sockets são frequentemente aplicadas em projetos académicos e protótipos de sistemas distribuídos, pela sua simplicidade e controlo direto sobre a comunicação entre processos. No contexto da persistência de dados, SGBDs relacionais como o PostgreSQL ou MySQL permanecem como escolha robusta, suportando operações transacionais, integridade referencial e consultas complexas.

Além disso, a utilização de protocolos de comunicação próprios, baseados em strings estruturadas (como <comando> <entidade>), permite flexibilidade na extensão dos comandos e fácil interpretação por ambas as extremidades. Este padrão é comum em sistemas de baixo nível ou onde não é necessária a complexidade de protocolos como HTTP ou SOAP.

3. Objectivos e Metodologias

3.1. Ferramentas e Tecnologias

O IDE utilizado para a realização deste projeto foi o **VS Code**. À diferença de outros IDE's tradicionais como **Eclipse** ou **IntelliJ IDEA**, o VS Code é conhecido pela sua leveza e rapidez. Isto torna-o uma ótima opção para máquinas com recursos limitados.

O VS Code é altamente extensível, o que significa que é possível personalizá-lo para atender às necessidades específicas.

Por fim a capacidade de trabalhar com ferramentas como **Maven** e **Gradle** dentro do VS Code simplifica o gerenciamento de projetos Java.

A escolha do **PostgreSQL** como sistema de gerenciamento de bases de dados foi motivada pela familiaridade adquirida durante o semestre na disciplina de Bases de Dados II, onde o mesmo foi utilizado. A continuidade do uso visa consolidar o conhecimento e aprofundar o entendimento da ferramenta.

3.2. Planeamento

Este projeto foi desenvolvido de forma iterativa e **ad hoc**, ou seja, sem a adoção de uma metodologia formal de desenvolvimento. As funcionalidades foram implementadas progressivamente, seguindo a sequência proposta no enunciado, sem uma estrutura prévia de planeamento ou organização detalhada. Essa abordagem permitiu uma construção contínua do sistema, focando na implementação funcional de cada requisito à medida que era interpretado.

4. Trabalho Desenvolvido

4.1. Requisitos Implementados

R1	Implementado
R2	Implementado
R3	Implementado
R4	Implementado
R5	Implementado
R6	Implementado
R7	Implementado
R8	Implementado
R9	Implementado
R10	Implementado
R11	Implementado
R12	Implementado
R13	50% Implementado
R14	Implementado
R15	Não Implementado
R16	Não Implementado
R17	Implementado
R18	Implementado
R19	Implementado
R20	Implementado
R21	Implementado
R22	Implementado
R23	Implementado
R24	Não Implementado

R25	Implementado
R26	Implementado

4.2. Classes e Packages

- *Classe servidor*

A classe **servidor** implementa um servidor **TCP (Transmission Control Protocol)** que escuta conexões de clientes em uma porta especificada pelo utilizador.

O método **main** é o ponto de entrada do servidor. É primeiro solicitado ao utilizador que introduza o número da **porta** a ser utilizada pelo servidor. Em seguida, é obtido o endereço **IP** local do servidor e é criado um **ServerSocket** que escuta conexões nessa porta. Uma mensagem é impressa indicando que o servidor foi iniciado com sucesso, juntamente com a **porta** e o **IP**.

O servidor entra num **loop** infinito (**while (true)**) para aceitar novas conexões de clientes. Quando um cliente se conecta, um objeto **Socket** é criado para representar a conexão. Canais de entrada (**BufferedReader**) e saída (**DataOutputStream**) são criados para comunicação com o cliente. O servidor lê o nome de utilizador enviado pelo cliente e envia uma mensagem de boas-vindas ("<Servidor> <hello>").

O servidor então entra num **loop** para processar comandos enviados pelo cliente. Este lê uma mensagem do cliente e verifica se esta termina com um dos seguintes sufixos, indicando um comando específico: "<Info>;", "<update>;", "<inserir> <equipamento>;", "<pesquisa> <equipamento>;", "<listar> <equipamento>;", "<listar> <certificacao>;", "<certificacao> <num_serie>;", ou "<bye>;".

Dependendo do comando recebido, o servidor chama um dos seguintes métodos estáticos para processar a solicitação do cliente:

- **Visualizar_Informações:**

O método Visualizar_Informações permite que um utilizador veja as suas informações. Este recebe os canais de comunicação e lê o nome de utilizador do cliente. O método consulta a tabela utilizador para encontrar o utilizador com o nome fornecido. Se o utilizador for encontrado, as suas informações básicas (**ID**, **nome**, **username**, **email**, **tipo**) são recuperadas e formatadas. Se o utilizador for do tipo "**fabricante**", o método também consulta a tabela fabricante para obter informações adicionais específicas do fabricante (**NIF**, **telefone**, **morada**, **setor comercial**, **início da atividade**). Todas as informações são concatenadas numa **string** e enviadas ao cliente. O tratamento de erros **SQLException** garante que problemas na consulta sejam comunicados ao cliente.

```
public static void Visualizar_Informações(BufferedReader input, DataOutputStream output) throws NumberFormatException, IOException
{
    Conexao_BD conexao = new Conexao_BD();
    Connection conn = conexao.conexao();

    String username_utilizador = input.readLine();

    String sql = "SELECT * FROM utilizador WHERE username_utilizador = ?";
    PreparedStatement stmt = conn.prepareStatement(sql);
    stmt.setString(1, username_utilizador);
    ResultSet rs = stmt.executeQuery();

    if (rs.next())
    {
        //System.out.println("\n Enviando sobre do utilizador ... ");
        int id_utilizador = rs.getInt("id_utilizador");
        String tipo = rs.getString("tipo");

        String mensagem = "<Servidor> <Info> \n<ID: " + id_utilizador +
                           "\nNome: " + rs.getString("nome_utilizador") +
                           "\nUsername: " + rs.getString("username_utilizador") +
                           "\nEmail: " + rs.getString("email_utilizador") +
                           "\nTipo: " + tipo;

        if (tipo.equalsIgnoreCase("fabricante"))
        {
            String sql2 = "SELECT * FROM fabricante WHERE id_utilizador = ?";
            PreparedStatement stmt2 = conn.prepareStatement(sql2);
            stmt2.setInt(1, id_utilizador);
            ResultSet rs2 = stmt2.executeQuery();

            while (rs2.next())
            {
                mensagem += "\nNIF: " + rs2.getString("nif_fabricante") +
                            "\nTelefone: " + rs2.getInt("telefone_fabricante") +
                            "\nMorada: " + rs2.getString("morada_fabricante") +
                            "\nSector Comercial: " + rs2.getString("sector_comercial") +
                            "\nInício da Atividade: " + rs2.getString("inicio_actividade") +
                            " >";
            }
            rs2.close();
            stmt2.close();
        }
        output.writeBytes(mensagem + "\n");
        output.writeBytes("EOF\n");
    }
}
```

Figura 1: Método Visualizar_informações

- **Alterar_Informações:**

O método **Alterar_Informações** possibilita que um utilizador atualize os seus dados pessoais ou os dados da sua empresa (se for um fabricante). Ele recebe os canais de comunicação e lê o **ID** do utilizador a ser alterado. O método primeiro consulta a tabela utilizador para obter as informações atuais do utilizador e, se for um fabricante, consulta também a tabela fabricante. As informações atuais são formatadas e enviadas ao cliente. Em seguida, o método lê o número do campo que o cliente deseja alterar e o novo valor para esse campo. Uma validação básica é realizada para os campos de **telefone** e **NIF**. Uma **query SQL UPDATE** é construída dinamicamente com base no campo escolhido e executada. Uma mensagem de sucesso ou falha é enviada ao cliente, dependendo do resultado da atualização. O método inclui tratamento de erros **SQLException** para informar o cliente sobre problemas na atualização.

```
public static void Alterar_Informações(BufferedReader input, DataOutputStream output) throws NumberFormatException, IOException
{
    Conexao_BD conexaoBD = new Conexao_BD();
    Connection conn = conexaoBD.conexao();

    // Obter dados do utilizador
    String sql = "SELECT * FROM utilizador WHERE id_utilizador = ?";
    PreparedStatement stmtUtilizador = conn.prepareStatement(sql);
    stmtUtilizador.setInt(1, id_utilizador);
    ResultSet rs = stmtUtilizador.executeQuery();

    if (!rs.next())
    {
        output.writeBytes("Utilizador não encontrado.\nEOF\n");
        return;
    }

    // Mensagem base
    StringBuilder mensagem = new StringBuilder("<Servidor> <Info>");
    mensagem.append("\n==== Informações Atuais ====");
    mensagem.append("\n1. Nome: ").append(rs.getString("nome_utilizador"));
    mensagem.append("\n2. Username: ").append(rs.getString("username_utilizador"));
    mensagem.append("\n3. Email: ").append(rs.getString("email_utilizador"));

    String tipo = rs.getString("tipo");
    rs.close();
    stmtUtilizador.close();

    String sql2 = "SELECT * FROM fabricante WHERE id_utilizador = ?";
    PreparedStatement stmtFab = conn.prepareStatement(sql2);
    stmtFab.setInt(1, id_utilizador);
    ResultSet rs2 = stmtFab.executeQuery();

    if (rs2.next())
    {
        mensagem.append("\n4. NIF: ").append(rs2.getString("nif_fabricante"));
        mensagem.append("\n5. Telefone: ").append(rs2.getInt("telefone_fabricante"));
        mensagem.append("\n6. Morada: ").append(rs2.getString("morada_fabricante"));
        mensagem.append("\n7. Sector Comercial: ").append(rs2.getString("sector_comercial"));
        mensagem.append("\n8. Início da Atividade: ").append(rs2.getString("inicio_actividade"));
    }

    rs2.close();
    stmtFab.close();

    // Mostrar tudo ao utilizador
    output.writeBytes(mensagem.toString() + "\n");
    output.writeBytes("EOF\n");

    // Atualização
    String campo = input.readLine();
    String label = "";
    int campoEscolhido = 0;
```

Figura 2: Método Alterar_Informações 1

```
public static void Alterar_Informações(BufferedReader input, DataOutputStream output) throws NumberFormatException, IOException
{
    try
    {
        campoEscolhido = Integer.parseInt(campo);
    }
    catch (NumberFormatException e)
    {
        output.writeBytes("Valor invalido. Deve ser um número entre 1 e 8.\nEOF\n");
        return;
    }

    // Mapear campo
    switch (campoEscolhido)
    {
        case 1: label = "nome_utilizador"; break;
        case 2: label = "username_utilizador"; break;
        case 3: label = "email_utilizador"; break;
        case 4: label = "nif_fabricante"; break;
        case 5: label = "telefone_fabricante"; break;
        case 6: label = "morada_fabricante"; break;
        case 7: label = "sector_comercial"; break;
        case 8: label = "inicio_actividade"; break;
        default:
        {
            output.writeBytes("Campo invalido. Escolhe de 1 a 8.\nEOF\n");
            return;
        }
    }

    String novoValor = input.readLine();
    boolean valido = true;

    // Validação
    if (label.equals("telefone_fabricante") && !novoValor.matches("[923]\\d{8}"))
    {
        //output.writeBytes("Número de telefone invalido! Deve ter 9 digitos e comecar com 9, 2 ou 3.");
        output.writeBytes("<Servidor> <update> <fail>\n");
        output.writeBytes("\nEOF\n");
        valido = false;
        //System.out.println("Validação errada de telefone");
    }
    if (label.equals("nif_fabricante") && !novoValor.matches("\\d{9}"))
    {
        //output.writeBytes("NIF invalido! Deve ter 9 digitos.\nEOF\n");
        output.writeBytes("<Servidor> <update> <fail>\n");
        output.writeBytes("\nEOF\n");
        valido = false;
        //System.out.println("Validação errada de NIF");
    }

    // Preparar update
    String sqlUpdate;
```

Figura 3: Método Alterar_Informações 2

```
public static void Alterar_Informações(BufferedReader input, DataOutputStream output) throws NumberFormatException, IOException
{
    String sqlupdate,
    if (valido==true)
    {
        if (campoEscolhido < 3)
        {
            sqlUpdate = "UPDATE utilizador SET " + label + " = ? WHERE id_utilizador = ?";
        }
        else
        {
            sqlUpdate = "UPDATE fabricante SET " + label + " = ? WHERE id_utilizador = ?";
        }
        PreparedStatement stmtUpdate = conn.prepareStatement(sqlUpdate);

        if(label.equals("nif_fabricante") || label.equals("telefone_fabricante"))
        {
            int num = Integer.parseInt(novoValor);
            stmtUpdate.setInt(1, num);
        }
        else
        {
            stmtUpdate.setString(1, novoValor);
        }
        stmtUpdate.setInt(2, id_utilizador);

        int linhasAfetadas = stmtUpdate.executeUpdate();
        if (linhasAfetadas > 0)
        {
            //output.writeBytes("Informacao do utilizador atualizada com sucesso!\n");
            output.writeBytes("<Servidor> <update> <ok>\n");
            output.writeBytes("\nEOF\n");
        }
        else
        {
            //output.writeBytes("Nenhuma linha foi atualizada.\n");
            output.writeBytes("<Servidor> <update> <fail>\n");
            output.writeBytes("\nEOF\n");
        }
        output.writeBytes("EOF\n");
        stmtUpdate.close();
        conn.close();
    }
    catch (SQLException e)
    {
        output.writeBytes("Erro: " + e.getMessage() + "\nEOF\n");
    }
}
```

Figura 4: Método Alterar_Informações 3

- **Adicionar_Equipamento:**

O método **Adicionar_Equipamento** permite que um utilizador insira um novo equipamento no banco de dados. Este recebe os canais de comunicação com o cliente. O método lê sequencialmente os dados do novo equipamento enviados pelo cliente (**ID do utilizador, marca, modelo, setor comercial, potência, amperagem, número do modelo**). Um **SKU** único para o equipamento é gerado aleatoriamente. A data de submissão é automaticamente definida como a data atual. Uma **query SQL INSERT** é preparada para inserir esses dados na tabela **equipamento**. Um **PreparedStatement** é utilizado para parametrizar a **query**. Após a execução da **query**, o método verifica o número de linhas afetadas para determinar se a inserção foi bem-sucedida e envia uma mensagem de sucesso ou falha ao cliente. O tratamento de exceções **SQLException** e **IllegalArgumentException** garante que erros relacionados com o banco de dados ou com o formato dos dados sejam capturados e informados ao cliente.

```
public static void Adicionar_Equipamento(BufferedReader input, DataOutputStream output) throws NumberFormatException, IOException
{
    int id_utilizador = Integer.parseInt(input.readLine());
    try
    {
        Conexao_BD conexaoBD = new Conexao_BD();
        Connection conn = conexaoBD.conexao();

        String sql = "INSERT INTO equipamento (id_utilizador, marca_equipamento, modelo_equipamento, sector_comercial_equipamento, " +
                    "potencia, amperagem, sku_equipamento, n_modelo, data_submissao, submetido_certificacao) " +
                    "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

        PreparedStatement stmt = conn.prepareStatement(sql);

        String marca = input.readLine();
        String modelo = input.readLine();
        String setor = input.readLine();
        float potencia = Float.parseFloat(input.readLine());
        float amperagem = Float.parseFloat(input.readLine());

        Random rand = new Random();
        int n_equipamento = rand.nextInt(1000000) + 1;
        int n_modelo = Integer.parseInt(input.readLine());

        LocalDate dataAtual = LocalDate.now();
        java.sql.Date dataSubmissaoSQL = java.sql.Date.valueOf(dataAtual);

        stmt.setInt(1, id_utilizador);
        stmt.setString(2, marca);
        stmt.setString(3, modelo);
        stmt.setString(4, setor);
        stmt.setFloat(5, potencia);
        stmt.setFloat(6, amperagem);
        stmt.setInt(7, n_equipamento);
        stmt.setInt(8, n_modelo);
        stmt.setDate(9, dataSubmissaoSQL);
        //stmt.setDate(10, dataCertificacaoSQL);
        stmt.setString(10, "Nao submetido");

        int rowsAffected = stmt.executeUpdate();
        if (rowsAffected > 0)
        {
            output.writeBytes("\n <Servidor> <inserir> <equipamento> <ok>;");
            output.writeBytes("\nEOF\n");
        }
        else
        {
            output.writeBytes("\n <Servidor> <inserir> <equipamento> <fail>;");
            output.writeBytes("\nEOF\n");
        }

        stmt.close();
        conn.close();
    }
    catch (SQLException e)
    {
    }
}
```

Figura 5: Método Adicionar_Equipamento

- **Listar_Equipamento:**

O método **Listar_Equipamento** tem como objetivo recuperar e enviar informações detalhadas de um equipamento específico para o cliente. Ele recebe o **BufferedReader** e o **DataOutputStream** como parâmetros para comunicar com o cliente. O método lê o **SKU** do equipamento fornecido pelo cliente e constrói uma **query SQL SELECT** para procurar esse equipamento na tabela equipamento. Um **PreparedStatement** é utilizado para evitar injeção de **SQL**. Após executar a **query**, o método verifica se algum equipamento foi encontrado. Se não, envia uma mensagem de falha ao cliente. Caso contrário, itera sobre os resultados (**ResultSet**), formatando as informações de cada coluna do equipamento numa string e enviando-a de volta ao cliente. Finalmente, fecha os recursos de banco de dados para liberar a conexão. O tratamento de exceções **SQLException** garante que erros durante a interação com o banco de dados sejam capturados e uma mensagem de erro seja enviada ao cliente.

```
public class servidor
    public static void Listar_Equipamento(BufferedReader input, DataOutputStream output) throws NumberFormatException, IOException
    {
        try
        {
            Conexao_BD conexaoBD = new Conexao_BD();
            Connection conn = conexaoBD.conexao();

            String sql = "SELECT * FROM equipamento WHERE sku_equipamento = ?";
            int sku_equipamento = Integer.parseInt(input.readLine());

            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setInt(1, sku_equipamento);
            ResultSet rs = stmt.executeQuery();

            if(!rs.isBeforeFirst())
            {
                output.writeBytes("<Servidor> <pesquisa> <equipamento> <fail>;\n");
                output.writeBytes("\nEOF\n");
            }
            else
            {
                StringBuilder mensagem = new StringBuilder("<Servidor> <pesquisa> <equipamento>");

                while (rs.next())
                {
                    mensagem.append("\n==== Informações Equipamento ====");
                    mensagem.append("\n ID Equipamento: ").append(rs.getInt("id_equipamento"));
                    mensagem.append("\n ID Certificação: ").append(rs.getInt("id_certificacao"));
                    mensagem.append("\n ID Utilizador: ").append(rs.getInt("id_utilizador"));
                    mensagem.append("\n Marca do Equipamento: ").append(rs.getString("marca_equipamento"));
                    mensagem.append("\n Modelo do Equipamento: ").append(rs.getString("modelo_equipamento"));
                    mensagem.append("\n Setor Comercial do Equipamento: ").append(rs.getString("sector_comercial_equipamento"));
                    mensagem.append("\n Potência do Equipamento: ").append(rs.getFloat("potencia"));
                    mensagem.append("\n Amperagem do Equipamento: ").append(rs.getFloat("amperagem"));
                    mensagem.append("\n SKU do Equipamento: ").append(rs.getInt("sku_equipamento"));
                    mensagem.append("\n Número do Modelo: ").append(rs.getInt("n_modelo"));
                    mensagem.append("\n Data de Submissão: ").append(rs.getString("data_submissao"));
                    mensagem.append("\n Submetido Certificação: ").append(rs.getString("submetido_certificacao"));
                    mensagem.append("\n");

                    output.writeBytes(mensagem.toString());
                    output.writeBytes("\nEOF\n");
                }

                rs.close();
                stmt.close();
                conn.close(); // Fechar a conexão
            }
            catch (SQLException e)
            {
                //System.out.println("Erro ao listar equipamentos: " + e.getMessage());
                output.writeBytes(" <username> <pesquisa> <equipamento> <fail>\n" + "\nEOF\n");
            }
        }
    }
```

Figura 6: Método Listar_Equipamentos

- **Pesquisar_Equipamento:**

O método **Pesquisar_Equipamento** lista todos os equipamentos associados a um determinado utilizador. Este recebe os canais de comunicação e lê o **ID** do utilizador. Uma **query SQL SELECT** é preparada para recuperar todos os equipamentos cujo **id_utilizador** corresponde ao fornecido. Os resultados são iterados, e as informações de cada equipamento são formatadas e enviadas ao cliente. Se nenhum equipamento for encontrado para o utilizador, uma mensagem de falha é enviada. O tratamento de erros **SQLException** assegura que erros de banco de dados sejam reportados ao cliente.

```
public static void Pesquisar_Equipamento(BufferedReader input, DataOutputStream output) throws NumberFormatException, IOException
{
    Conexao_BD conexaoBD = new Conexao_BD();
    Connection conn = conexaoBD.conexao();

    int id_utilizador = Integer.parseInt(input.readLine());

    PreparedStatement pstmt;
    ResultSet rs;

    String sql = "SELECT * FROM equipamento WHERE id_utilizador = ?";
    pstmt = conn.prepareStatement(sql);
    pstmt.setInt(1, id_utilizador);

    rs = pstmt.executeQuery();

    StringBuilder mensagem = new StringBuilder("<username> <listar> <equipamento>\n");

    boolean encontrado = false;
    while (rs.next())
    {
        encontrado = true;
        mensagem.append("\n==== Informações Equipamento ====");
        mensagem.append("\nID Equipamento: ").append(rs.getInt("id_equipamento"));
        mensagem.append("\nID Certificação: ").append(rs.getInt("id_certificacao"));
        mensagem.append("\nID Utilizador: ").append(rs.getInt("id_utilizador"));
        mensagem.append("\nMarca do Equipamento: ").append(rs.getString("marca_equipamento"));
        mensagem.append("\nModelo do Equipamento: ").append(rs.getString("modelo_equipamento"));
        mensagem.append("\nSetor Comercial do Equipamento: ").append(rs.getString("sector_comercial_equipamento"));
        mensagem.append("\nPotência do Equipamento: ").append(rs.getFloat("potencia"));
        mensagem.append("\nAmperagem do Equipamento: ").append(rs.getFloat("amperagem"));
        mensagem.append("\nSKU do Equipamento: ").append(rs.getInt("sku_equipamento"));
        mensagem.append("\nModelo do Equipamento: ").append(rs.getInt("n_modelo"));
        mensagem.append("\nData Submissão do Equipamento: ").append(rs.getString("data_submissao"));
        mensagem.append("\nSubmetido Certificação do Equipamento: ").append(rs.getString("submetido_certificacao"));
        mensagem.append("\n");

        output.writeBytes(mensagem + "\n");
        output.writeBytes("EOF\n");
    }

    if (!encontrado)
    {
        output.writeBytes("<Servidor> <listar> <equipamento> <fail>;\n");
        output.writeBytes("\nEOF\n");
        return;
    }

    // Fechar recursos
    rs.close();
    pstmt.close();
    conn.close();
}
catch (SQLException e)
{
```

Figura 7: Método Pesquisar_Equipamento

- **Listar_Certificação:**

O método **Listar_Certificação** lista todas as certificações associadas a um determinado utilizador. Ele recebe os canais de comunicação e lê o **ID** do utilizador. Uma **query SQL SELECT** é preparada para juntar as tabelas certificacao e equipamento e recuperar as certificações associadas ao **id_utilizador** fornecido. As informações de cada certificação (**ID da certificação, ID do equipamento, estado, data de realização**) são formatadas e enviadas ao cliente. Se nenhuma certificação for encontrada, uma mensagem de falha é enviada. O tratamento de erros **SQLException** garante que problemas na consulta sejam comunicados ao cliente.

```
public static void Listar_Certificação(BufferedReader input, DataOutputStream output) throws NumberFormatException, IOException
{
    int id_utilizador = Integer.parseInt(input.readLine());
    String sql = "SELECT c.id_certificacao, e.id_equipamento, c.estado, data_realizacao FROM certificacao c JOIN equipamento e ON c.id_equipamento = e.id_equipamento WHERE c.id_utilizador = ?";

    try
    {
        Conexao_BD conexaoBD = new Conexao_BD();
        Connection conn = conexaoBD.conexao();
        PreparedStatement stmt = conn.prepareStatement(sql);
        stmt.setInt(1, id_utilizador);
        ResultSet rs = stmt.executeQuery();

        if (!rs.isBeforeFirst())
        {
            output.writeBytes("<Servidor> <listar> <certificação> <fail>;\n");
            output.writeBytes("\nEOF\n");
        }
        else
        {
            StringBuilder mensagem = new StringBuilder("==== Informações Certificação ====");

            while (rs.next())
            {
                mensagem.append("\nID Certificação: ").append(rs.getInt("id_certificacao"));
                mensagem.append("\nID Equipamento: ").append(rs.getInt("id_equipamento"));
                mensagem.append("\nEstado da Certificação: ").append(rs.getString("estado"));
                mensagem.append("\nData Realização da Certificação: ").append(rs.getString("data_realizacao"));
                mensagem.append("\n");

                output.writeBytes(mensagem.toString() + "\n");
                output.writeBytes("EOF\n");
            }

            rs.close();
            stmt.close();
            conn.close();
        }
        catch (SQLException e)
        {
            output.writeBytes("Erro ao listar pedidos de certificação: " + e.getMessage() + "\nEOF\n");
        }
    }
}
```

Figura 8: Método Listar_Certificação

- **Pesquisar_Certificação:**

O método **Pesquisar_Certificação** permite pesquisar uma certificação específica com base no seu número e no ID do utilizador. Ele recebe os canais de comunicação e lê o ID do utilizador e o número da certificação. Uma **query SQL SELECT** com uma cláusula **WHERE** para filtrar pelo ID da certificação e pelo ID do utilizador é preparada. As informações da certificação encontrada (**ID da certificação, ID do equipamento, estado, data de realização**) são formatadas e enviadas ao cliente. Se nenhuma certificação correspondente for encontrada, uma mensagem de falha é enviada. O tratamento de erros **SQLException** assegura que erros de banco de dados sejam reportados ao cliente.

```
public static void Pesquisar_Certificação(BufferedReader input, DataOutputStream output) throws NumberFormatException, IOException
{
    //System.out.println("Insira o ID do utilizador: ");
    int id_utilizador = Integer.parseInt(input.readLine());

    String sql = "";
    PreparedStatement stmt = null;
    ResultSet rs = null;

    try
    {
        Conexao_BD conexaoBD = new Conexao_BD();
        Connection conn = conexaoBD.conexao();

        int numero = Integer.parseInt(input.readLine());
        sql = "SELECT c.id_certificacao, e.id_equipamento, c.estado, c.data_realizacao " +
              "FROM certificacao c JOIN equipamento e ON c.id_certificacao = e.id_certificacao " +
              "WHERE c.id_certificacao = ? AND e.id_utilizador = ?";
        stmt = conn.prepareStatement(sql);
        stmt.setInt(1, numero);
        stmt.setInt(2, id_utilizador);

        rs = stmt.executeQuery();

        StringBuilder mensagem = new StringBuilder("<username> <pesquisa> <certificacao>\n");

        if(!rs.isBeforeFirst())
        {
            output.writeBytes("<Servidor> <pesquisa> <certificacao> <fail>;\n");
            output.writeBytes("\nEOF\n");
        }
        else
        {
            while (rs.next())
            {
                mensagem.append("== Informações da Certificação ==");
                mensagem.append("\nID Certificação: ").append(rs.getInt("id_certificacao"));
                mensagem.append("\nID Equipamento: ").append(rs.getInt("id_equipamento"));
                mensagem.append("\nEstado da Certificação: ").append(rs.getString("estado"));
                mensagem.append("\nData Realização: ").append(rs.getString("data_realizacao"));
                mensagem.append("\n");
            }
        }

        output.writeBytes(mensagem.toString() + "\n");
        output.writeBytes("EOF\n");
    }
    catch (SQLException e)
    {
        output.writeBytes("Erro ao listar pedidos de certificação: " + e.getMessage() + "\nEOF\n");
    }
}
```

Figura 9: Método Pesquisar_Certificação

Se o comando enviado pelo cliente não corresponder a nenhum dos comandos esperados, o servidor envia uma mensagem de "**Comando inválido!**". O servidor continua a ler comandos do cliente até que a conexão seja encerrada (quando o cliente envia "**<bye>;**"). Após o **loop** de comandos, os canais de entrada e saída e o socket da conexão com o cliente são fechados.

Em caso de qualquer exceção durante a execução do servidor (por exemplo, erro ao criar o socket, erro de E/S), uma mensagem de erro é impressa e a stack trace da exceção é exibida.

Cada um dos métodos chamados para processar os comandos do cliente interage com o banco de dados através da classe **Conexao_BD**. Estes preparam e executam **queries SQL** para recuperar ou modificar dados no banco de dados e enviam os resultados de volta ao cliente através do **DataOutputStream**. Estes também incluem tratamento de erros SQL e enviam mensagens de sucesso ou falha ao cliente conforme apropriado.

```
public static void main(String[] args)
{
    System.out.println("Indique o porto a ser utilizado: ");
    int porto = new Scanner(System.in).nextInt();

    String ipServidor = InetAddress.getLocalHost().getHostAddress();
    ServerSocket serverSocket = new ServerSocket(porto);
    System.out.println("Servidor TCP iniciado na porta " + porto + " com o IP " + ipServidor + "\n");

    while (true)
    {
        Socket socket = serverSocket.accept();

        // Criar canais de entrada e saída
        BufferedReader input = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        DataOutputStream output = new DataOutputStream(socket.getOutputStream());

        String username = input.readLine();
        System.out.println(username);

        output.writeBytes("<Servidor> <hello>\n");
        output.flush();

        //Comandos do cliente
        String mensagem = input.readLine();
        String comando;
        while ((comando = input.readLine()) != null)
        {
            if (mensagem.endsWith("<Info>"))
            {
                System.out.println(mensagem);
                Visualizar_Informações(input, output);
            }
            else if (mensagem.endsWith("<update>"))
            {
                System.out.println(mensagem);
                Alterar_Informações(input, output);
            }
            else if (mensagem.endsWith("<inserir> <equipamento>"))
            {
                System.out.println(mensagem);
                Adicionar_Equipamento(input, output);
            }
            else if (mensagem.endsWith("<pesquisa> <equipamento>"))
            {
                System.out.println(mensagem);
                Listar_Equipamento(input, output);
            }
            else if (mensagem.endsWith("<listar> <equipamento>"))
            {
                System.out.println(mensagem);
                Pesquisar_Equipamento(input, output);
            }
        }
    }
}
```

Figura 10: Classe servidor

- **Classe Fabricante_Servidor**

A classe **Fabricante_Servidor** tem como principal função apresentar um menu interativo para um utilizador do tipo fabricante, permitindo-lhe realizar diversas operações através de uma conexão de rede com um servidor.

- **Método Menu_Fabricante2:**

É o coração desta interação. Ele recebe o **username** do fabricante, o seu **id_utilizador** e o seu tipo como argumentos, estabelecendo o contexto da sessão.

Inicialmente, o método utiliza um **Scanner** para obter do utilizador o endereço **IP** e o número da **porta** do servidor ao qual se pretende conectar. Em seguida, tenta estabelecer uma conexão **Socket** com o servidor especificado. Após a conexão bem-sucedida, obtém o endereço **IP local do cliente** e imprime-o no console.

Canais de comunicação de saída (**DataOutputStream**) e entrada (**BufferedReader**) são criados para interagir com o servidor. Uma mensagem de saudação contendo o **username** do fabricante é enviada ao servidor. O método então entra num **loop** contínuo (**while(running)**) que apresenta um menu de opções ao fabricante. Estas opções incluem visualizar informações, alterar informações, adicionar equipamento, listar equipamentos, pesquisar equipamentos, listar certificações, pesquisar por certificação e sair.

A interação com o servidor é baseada na escolha da opção do menu. Para cada opção (exceto sair), uma mensagem formatada contendo o **username** e um identificador da operação desejada é enviada ao servidor. Dados adicionais que o servidor possa necessitar (como o campo e o novo valor para alterar informações, detalhes do equipamento para adicionar, SKU para pesquisar equipamento, etc.) são solicitados ao utilizador através do Scanner e enviados para o servidor.

O cliente então aguarda a resposta do servidor, lendo as linhas enviadas através do **BufferedReader** e imprimindo-as no console até encontrar um marcador de fim de transmissão ("EOF"). Este processo garante que toda a informação enviada pelo servidor em resposta à solicitação do cliente seja exibida ao utilizador.

A opção de sair (caso 8) envia uma mensagem de encerramento para o servidor e fecha os canais de entrada e saída, bem como o Socket, terminando a conexão e saindo do **loop** do menu, definindo a variável **running** para **false**.

Todo o bloco de código que envolve a interação de rede e a apresentação do menu está dentro de um bloco **try-catch** para capturar quaisquer exceções que possam ocorrer durante a comunicação de rede ou a entrada de dados do utilizador, imprimindo a exceção no console em caso de erro.

```
public class Fabricante_Servidor
{
    Windsurf: Refactor | Explain | Generate Javadoc | X
    public static void Menu_Fabricante(int id_utilizador, String tipo)
    {
        Scanner scanner = new Scanner(System.in);
        boolean running = true;
        int opção;

        try
        {
            System.out.println("Indique o IP do Servidor: ");
            String ipServidor = scanner.nextLine();
            System.out.println("Indique o Porto do Servidor: ");
            int porto = scanner.nextInt();
            Socket socket = new Socket("192.168.1.65", 1234);

            System.out.println("O seu IP eh: " + socket.getInetAddress().getHostAddress() + "\n");
            DataOutputStream output = new DataOutputStream(socket.getOutputStream());
            BufferedReader input = new BufferedReader(new InputStreamReader(socket.getInputStream()));

            output.writeBytes("<" + username + "> " + "<hello>;\n");
            String server = input.readLine();
            System.out.println(server);
            while(running)
            {
                System.out.println("\n☰ Menu de Fabricante ☰");
                System.out.println("1. Visualizar informacão");
                System.out.println("2. Alterar informacão");
                System.out.println("3. Adicionar Equipamento");
                System.out.println("4. Listar Equipamentos");
                System.out.println("5. Pesquisar Equipamentos");
                System.out.println("6. Listar Certificacões");
                System.out.println("7. Pesquisar por Certificação");
                System.out.println("8. Sair");
                System.out.print("Escolha uma opção: ");

                int option = scanner.nextInt();
                scanner.nextLine(); // Consumir a quebra de linha
            }
        }
    }
}
```

Figura 11: Classe Fabricante_Servidor 1

```
public static void Menu_Fabricante2(String username, int id_utilizador, String tipo)
{
    switch (option)
    {
        case 1:
            String mensagem = "<" + username + ">" + "<Info>;";
            output.writeBytes(mensagem + "\n");
            output.writeBytes(username);
            output.writeBytes(option + "\n");
            output.flush();
            output.writeBytes(username + "\n");
            output.flush();

            String resposta;
            while (!(resposta = input.readLine()).equals("EOF"))
            {
                System.out.println(resposta);
            }
            break;
        case 2:
            String mensagem3 = "<" + username + "> " + "<update>;";
            output.writeBytes(mensagem3 + "\n");
            output.writeBytes(option + "\n");
            output.writeBytes(id_utilizador + "\n");
            output.flush();

            String resposta2;
            while (!(resposta2 = input.readLine()).equals("EOF"))
            {
                System.out.println(resposta2);
            }

            System.out.println("Insira o campo que deseja alterar (1-8): ");
            String campo = scanner.nextLine();
            output.writeBytes(campo + "\n");

            System.out.println("Insira o novo valor: ");
            String novoValor = scanner.nextLine();
            output.writeBytes(novoValor + "\n");
            output.flush();

            String resposta3;
            while (!(resposta3 = input.readLine()).equals("EOF"))
            {
                System.out.println(resposta3);
            }
            break;
        case 3:
            String mensagem4 = "<" + username + "> " + "<inserir> <equipamento>;";
            output.writeBytes(mensagem4 + "\n");
            output.writeBytes(option + "\n");
            output.writeBytes(id_utilizador + "\n");
            output.flush();

            System.out.println("Insira a marca do equipamento: ");
            String marca = scanner.nextLine();
    }
}
```

Figura 12: Classe Fabricante_Servidor 2

4.3. Algoritmos

A maioria dos métodos utiliza algoritmos de consulta e manipulação de bases de dados, implementados através de instruções SQL.

As consultas SELECT são utilizadas para recuperar dados das tabelas, enquanto as instruções INSERT, UPDATE e DELETE são utilizadas para inserir, atualizar e excluir dados, respetivamente.

Os algoritmos SQL são otimizados pelo sistema de gerenciamento de bases de dados para garantir a eficiência das operações

4.4. Estruturas de Dados

- Tabelas de Bases de Dados Relacional:

A principal estrutura de dados utilizada é a base de dados relacional. As tabelas (equipamento, certificacao, licencas, utilizador, tecnico) armazenam os dados de forma estruturada, com relações definidas entre elas.

As tabelas são compostas por colunas (atributos) e linhas (registros), e as relações entre as tabelas são estabelecidas por chaves primárias e estrangeiras.

- ResultSet:

A interface ResultSet é utilizada para armazenar os resultados das consultas SQL. Ela representa um conjunto de linhas de dados recuperadas da base de dados.

O ResultSet permite iterar sobre as linhas de resultados e acessar os valores das colunas.

4.5. Armazenamento de Dados

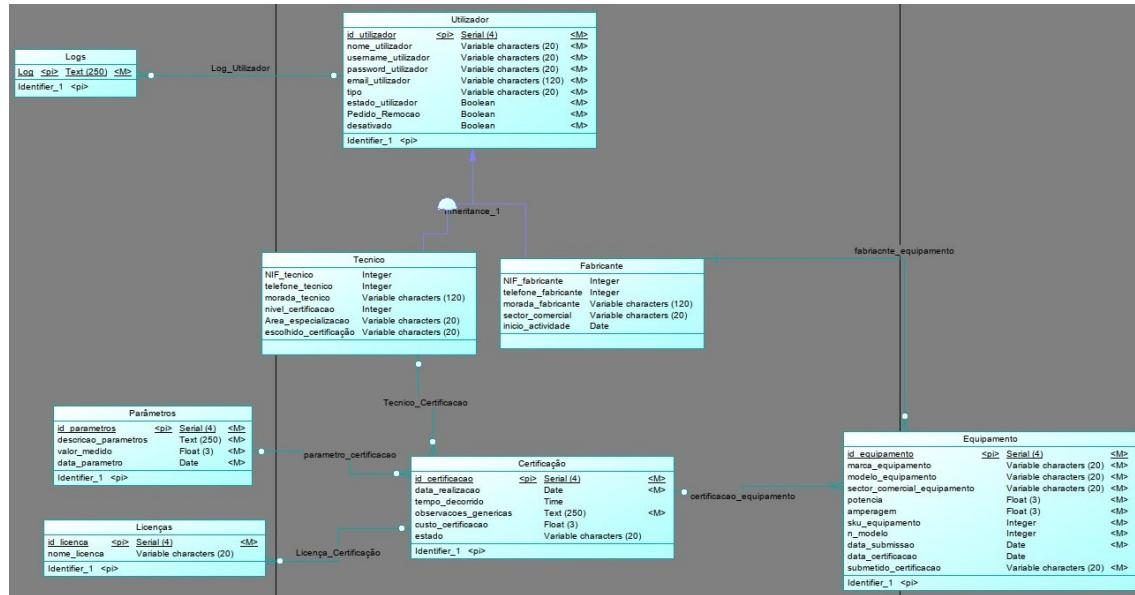


Figura 13: Modelo Entidade-Relacionamento

1. Tabelas Principais

Utilizador: Contém informações sobre os utilizadores, incluindo nome, username, senha, email e estado da conta.

Técnico: Subclasse de "Utilizador", representa técnicos com dados como NIF, telefone, morada, nível de certificação e área de especialização.

Fabricante: Armazena dados de fabricantes, como NIF, telefone, morada e setor comercial.

Equipamento: Representa equipamentos, contendo dados como marca, modelo, potência, amperagem, SKU e certificação associada.

Certificação: Contém informações sobre certificações realizadas, incluindo data, tempo decorrido, observações e estado.

2. Relacionamentos

Herança entre Utilizador e Técnico: O técnico é um tipo específico de utilizador.

Fabricante -> Equipamento: Um fabricante pode estar associado a vários equipamentos.

Equipamento -> Certificação: Equipamentos podem ter certificações associadas.

Técnico -> Certificação: Técnicos podem realizar certificações (representado por "Tecnico_Certificacao").

Certificação -> Parâmetros: Cada certificação pode ter parâmetros específicos medidos.

3. Outras Tabelas de Suporte

Logs: Mantém registros de ações dos utilizadores.

Licenças: Representa licenças associadas a certificações.

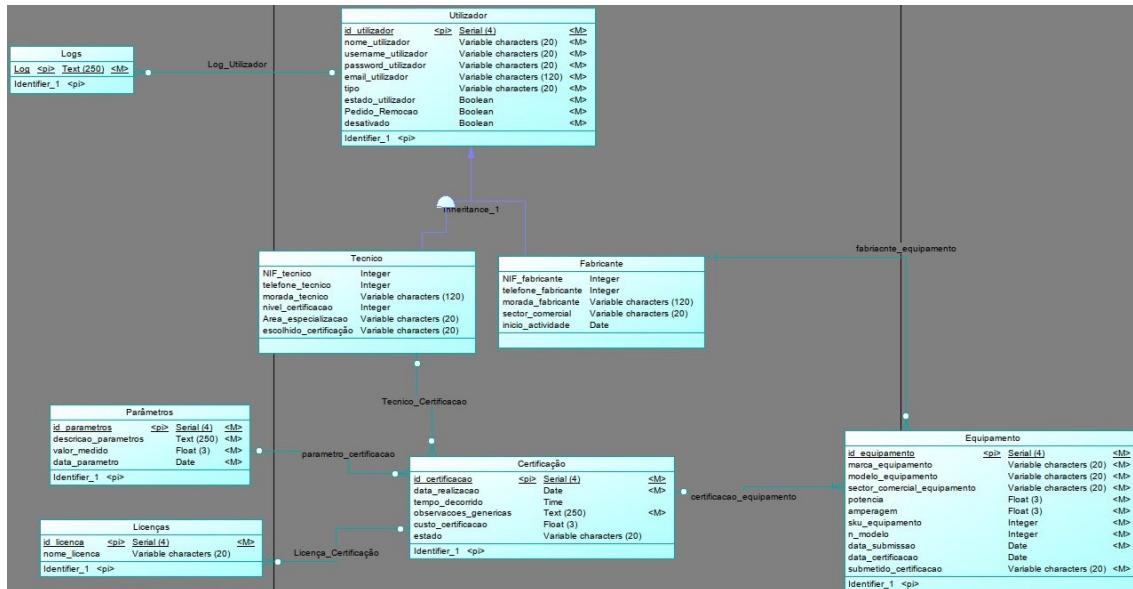


Figura 14: Modelo Físico

Deve ser descrita e explicada a estrutura em árvore do sistema de ficheiros e pastas necessários ao funcionamento do projeto.

Destes dois modelos foi possível gerar automaticamente o código da base de dados, através da aplicação powerDesigner.

Código de criação da base de dados:

```
/=====
/* DBMS name: PostgreSQL 7.3 */
/* Created on: 29/03/2025 07:56:30 */
=====/
```

```
drop index TECNICO_CERTIFICACAO_FK;
```

```
drop index PARAMETRO_CERTIFICACAO_FK;
```

```
drop index CERTIFICACAO_PK;
```

```
drop table CERTIFICACAO;
```

```
drop index FABRIACNTE_EQUIPAMENTO_FK;
```

```
drop index CERTIFICACAO_EQUIPAMENTO_FK;
```

```
drop index EQUIPAMENTO_PK;
```

```
drop table EQUIPAMENTO;
```

```
drop index FABRICANTE_PK;
```

```
drop table FABRICANTE;
```

```
drop index LICENCA_CERTIFICACAO_FK;
```

```
drop index LICENCAS_PK;
```

```
drop table LICENCAS;
```

```
drop index LOG_UTILIZADOR_FK;
```

```
drop index LOGS_PK;
```

```
drop table LOGS;
```

```
drop index PARAMETROS_PK;
```

```
drop table PARAMETROS;
```

```
drop index TECNICO_PK;
```

```
drop table TECNICO;
```

```
drop index UTILIZADOR_PK;
```

```
drop table UTILIZADOR;
```

```
/=====/*
```

```
/* Table: CERTIFICACAO */
```

```
/=====/*
```

```
create table CERTIFICACAO (
```

```
    ID_CERTIFICACAO SERIAL unique not null,
```

```
    ID_PARAMETROS INT4      null,
```

```
    ID_UTILIZADOR  INT4      null,
```

```
    DATA_REALIZACAO DATE     not null,
```

```
    TEMPO_DECORRIDO  INTERVAL   null,
```

```
    OBSERVACOES_GENERICAS TEXT     not null,
```

```
    CUSTO_CERTIFICACAO FLOAT     null,
```

```
    ESTADO        VARCHAR(20)  null,
```

```
    constraint PK_CERTIFICACAO primary key (ID_CERTIFICACAO)
```

```
);
```

```
/=====/*
```

```
/* Index: CERTIFICACAO_PK */
```

```
/=====/*
```

```
create unique index CERTIFICACAO_PK on CERTIFICACAO (
    ID_CERTIFICACAO
);

/=====
/* Index: PARAMETRO_CERTIFICACAO_FK */
=====

create index PARAMETRO_CERTIFICACAO_FK on CERTIFICACAO (
    ID_PARAMETROS
);

/=====
/* Index: TECNICO_CERTIFICACAO_FK */
=====

create index TECNICO_CERTIFICACAO_FK on CERTIFICACAO (
    ID_UTILIZADOR
);

/=====
/* Table: EQUIPAMENTO */
=====

create table EQUIPAMENTO (
    ID_EQUIPAMENTO SERIAL unique not null,
    ID_CERTIFICACAO INT4      null,
    ID_UTILIZADOR   INT4      not null,
    MARCA_EQUIPAMENTO VARCHAR(20) not null,
    MODELO_EQUIPAMENTO VARCHAR(20) not null,
    SECTOR_COMERCIAL_EQUIPAMENTO VARCHAR(20) not null,
```

```
POTENCIA      FLOAT      not null,  
AMPERAGEM     FLOAT      not null,  
SKU_EQUIPAMENTO INT4    unique      not null,  
N_MODELO       INT4      not null,  
DATA_SUBMISSAO DATE      not null,  
DATA_CERTIFICACAO DATE      null,  
SUBMETIDO_CERTIFICACAO VARCHAR(20)      not null,  
constraint PK_EQUIPAMENTO primary key (ID_EQUIPAMENTO)  
);
```

```
/=====/* Index: EQUIPAMENTO_PK */=====/  
create unique index EQUIPAMENTO_PK on EQUIPAMENTO (  
ID_EQUIPAMENTO  
);
```

```
/=====/* Index: CERTIFICACAO_EQUIPAMENTO_FK */=====/  
create index CERTIFICACAO_EQUIPAMENTO_FK on EQUIPAMENTO (  
ID_CERTIFICACAO  
);
```

```
/=====/* Index: FABRIACNTE_EQUIPAMENTO_FK */=====/  
create index FABRIACNTE_EQUIPAMENTO_FK on EQUIPAMENTO (
```

ID_UTILIZADOR

);

/=====/*

/* Table: FABRICANTE */

/=====/*

create table FABRICANTE (

 ID_UTILIZADOR INT4 not null,
 NIF_FABRICANTE INT4 unique null,
 TELEFONE_FABRICANTE INT4 unique null,
 MORADA_FABRICANTE VARCHAR(120) null,
 SECTOR_COMERCIAL VARCHAR(20) null,
 INICIO_ACTIVIDADE DATE null,

constraint PK_FABRICANTE primary key (ID_UTILIZADOR)

);

/=====/*

/* Index: FABRICANTE_PK */

/=====/*

create unique index FABRICANTE_PK on FABRICANTE (

 ID_UTILIZADOR

);

/=====/*

/* Table: LICENCAS */

/=====/*

create table LICENCAS (

 ID_LICENCA SERIAL unique not null,

```
ID_CERTIFICACAO    INT4          null,  
NOME_LICENCA       VARCHAR(20)    null,  
constraint PK_LICENCAS primary key (ID_LICENCA)  
);
```

```
/=====
```

```
/* Index: LICENCAS_PK */
```

```
/=====
```

```
create unique index LICENCAS_PK on LICENCAS (
```

```
ID_LICENCA
```

```
);
```

```
/=====
```

```
/* Index: LICENCA_CERTIFICACAO_FK */
```

```
/=====
```

```
create index LICENCA_CERTIFICACAO_FK on LICENCAS (
```

```
ID_CERTIFICACAO
```

```
);
```

```
/=====
```

```
/* Table: LOGS */
```

```
/=====
```

```
create table LOGS (
```

```
LOG        TEXT          not null,
```

```
ID_UTILIZADOR   INT4          null,
```

```
constraint PK_LOGS primary key (LOG)
```

```
);
```

```
/=====
/* Index: LOGS_PK */=====
/=====

create unique index LOGS_PK on LOGS (
LOG
);

/=====
/* Index: LOG_UTILIZADOR_FK */=====
/=====

create index LOG_UTILIZADOR_FK on LOGS (
ID_UTILIZADOR
);

/=====
/* Table: PARAMETROS */=====
/=====

create table PARAMETROS (
ID_PARAMETROS      SERIAL      unique      not null,
DESCRICAO_PARAMETROS TEXT      not null,
VALOR_MEDIDO      FLOAT      not null,
DATA_PARAMETRO     DATE      not null,
constraint PK_PARAMETROS primary key (ID_PARAMETROS)
);

/=====
/* Index: PARAMETROS_PK */=====
/=====
```

```
create unique index PARAMETROS_PK on PARAMETROS (
    ID_PARAMETROS
);

/=====
/* Table: TECNICO
=====
create table TECNICO (
    ID_UTILIZADOR      INT4          not null,
    NIF_TECNICO        INT4          unique     null,
    TELEFONE_TECNICO   INT4          unique     null,
    MORADA_TECNICO     VARCHAR(120)  null,
    NIVEL_CERTIFICACAO INT4          null,
    AREA_ESPECIALIZACAO VARCHAR(20)   null,
    ESCOLHIDO_CERTIFICACAO VARCHAR(20) null,
    constraint PK_TECNICO primary key (ID_UTILIZADOR)
);

/=====
/* Index: TECNICO_PK
=====
create unique index TECNICO_PK on TECNICO (
    ID_UTILIZADOR
);

/=====
/* Table: UTILIZADOR
=====

```

```
create table UTILIZADOR (
    ID_UTILIZADOR      SERIAL      unique      not null,
    NOME_UTILIZADOR    VARCHAR(20)   not null,
    USERNAME_UTILIZADOR VARCHAR(20) unique      not null,
    PASSWORD_UTILIZADOR VARCHAR(20)   not null,
    EMAIL_UTILIZADOR   VARCHAR(120) unique      not null,
    TIPO                VARCHAR(20)   not null,
    ESTADO_UTILIZADOR  BOOL        not null,
    PEDIDO_REMOCAO     BOOL        not null,
    DESATIVADO          BOOL        not null,
constraint PK_UTILIZADOR primary key (ID_UTILIZADOR)
);
```

```
/=====
/* Index: UTILIZADOR_PK */=====
=====

create unique index UTILIZADOR_PK on UTILIZADOR (
    ID_UTILIZADOR
);
```

```
alter table CERTIFICACAO
    add constraint FK_CERTIFIC_PARAMETRO_PARAMETR foreign key (ID_PARAMETROS)
        references PARAMETROS (ID_PARAMETROS)
        on delete restrict on update restrict;
```

```
alter table CERTIFICACAO
    add constraint FK_CERTIFIC_TECNICO_C_TECNICO foreign key (ID_UTILIZADOR)
        references TECNICO (ID_UTILIZADOR)
```

```
on delete restrict on update restrict;
```

```
alter table EQUIPAMENTO
```

```
add constraint FK_EQUIPAME_CERTIFICA_CERTIFIC foreign key (ID_CERTIFICACAO)  
references CERTIFICACAO (ID_CERTIFICACAO)  
on delete restrict on update restrict;
```

```
alter table EQUIPAMENTO
```

```
add constraint FK_EQUIPAME_FABRIACNT_FABRICAN foreign key (ID_UTILIZADOR)  
references FABRICANTE (ID_UTILIZADOR)  
on delete restrict on update restrict;
```

```
alter table FABRICANTE
```

```
add constraint FK_FABRICAN_INHERITAN_UTILIZAD foreign key (ID_UTILIZADOR)  
references UTILIZADOR (ID_UTILIZADOR)  
on delete restrict on update restrict;
```

```
alter table LICENCAS
```

```
add constraint FK_LICENCAS_LICENCA_C_CERTIFIC foreign key (ID_CERTIFICACAO)  
references CERTIFICACAO (ID_CERTIFICACAO)  
on delete restrict on update restrict;
```

```
alter table LOGS
```

```
add constraint FK_LOGS_LOG_UTILI_UTILIZAD foreign key (ID_UTILIZADOR)  
references UTILIZADOR (ID_UTILIZADOR)  
on delete restrict on update restrict;
```

```
alter table TECNICO
```

```
add constraint FK_TECNICO_INHERITAN_UTILIZAD foreign key (ID_UTILIZADOR)
    references UTILIZADOR (ID_UTILIZADOR)
    on delete restrict on update restrict;
```

4.6. Procedimentos de Teste

Os procedimentos de teste onde os testes foram feitos de forma manual e método a método.

Foi seguida a lógica de testes de Caixa Branca onde os testes são feitos de acordo com a lógica interna do sistema, ou seja, o testador (neste caso o desenvolvedor) analisa o código-fonte e testa cada caminho do programa, método por método, com base na estrutura do código.

5. Conclusões

O projeto foi terminado com 98% dos requisitos o que é muito bom. Foi bastante trabalho mas sinto que valeu todo o esforço de implementação e trabalho.

5.1. Forças

A principal força deste trabalho é a integração da linguagem de programação JAVA com modelos de base de dados relacionais e a comunicação em sockets.

5.2. Limitações

A principal limitação deste trabalho prendeu-se com o modelo de comunicação proposto pelo professor. O uso das mensagens no formato "<ack>" revelou-se ambíguo e de difícil interpretação, o que levou à sua não implementação no sistema final. Além disso, muitos dos comandos especificados no enunciado não apresentavam uma descrição clara ou detalhada sobre o seu comportamento esperado. Por esse motivo, optou-se por uma abordagem de interpretação pessoal, baseada na lógica contextual e na coerência funcional das restantes operações do sistema.

5.3. Trabalho Futuro

Com certeza a organização do tempo e definir tarefas para períodos de tempo específicos.

6. Referências

6.1. Lista de Referências