**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](#_Toc65518680)

[Lista de Figuras 7](#_Toc65518681)

[Lista de Tabelas 9](#_Toc65518682)

[Lista de Acrónimos 11](#_Toc65518683)

[1. Introdução 13](#_Toc65518684)

[2. Estado da Arte 13](#_Toc65518685)

[3. Objectivos e Metodologias 15](#_Toc65518686)

[3.1. Ferramentas e Tecnologias 15](#_Toc65518687)

[3.2. Planeamento 15](#_Toc65518688)

[4. Trabalho Desenvolvido 17](#_Toc65518689)

[4.1. Requisitos Implementados 17](#_Toc65518690)

[4.2. Classes e *Packages* 18](#_Toc65518691)

[4.3. Algoritmos 32](#_Toc65518692)

[4.4. Estruturas de Dados 32](#_Toc65518693)

[4.5. Armazenamento de Dados 33](#_Toc65518694)

[4.6. Procedimentos de Teste 45](#_Toc65518695)

[5. Conclusões 47](#_Toc65518696)

[5.1. Forças 47](#_Toc65518697)

[5.2. Limitações 47](#_Toc65518698)

[5.3. Trabalho Futuro 47](#_Toc65518699)

[6. Referências 49](#_Toc65518700)

[6.1. Lista de Referências 49](#_Toc65518701)

# Lista de Figuras

[Figura 1: Método Visualizar\_informações 17](#_Toc65518680)

[Figura 2: Método Alterar\_Informações 1 19](#_Toc65518681)

[Figura 3: Método Alterar\_Informações 2 20](#_Toc65518682)

[Figura 4: Método Alterar\_Informações 3 21](#_Toc65518683)

[Figura 5: Método Adicionar\_Equipamento 22](#_Toc65518684)

[Figura 6: Método Listar\_Equipamentos 23](#_Toc65518685)

[Figura 7: Método Pesquisar\_Equipamento 24](#_Toc65518686)

[Figura 8: Método Listar\_Certificação 25](#_Toc65518687)

[Figura 9: Método Pesquisar\_Certificação 26](#_Toc65518688)

[Figura 10: Classe servidor 27](#_Toc65518689)

[Figura 11: Classe Fabricante\_Servidor 1 29](#_Toc65518690)

[Figura 12: Classe Fabricante\_Servidor 2 30](#_Toc65518691)

[Figura 13: Modelo Entidade-Relacionamento 32](#_Toc65518692)

[Figura 14: Modelo Físico 33](#_Toc65518693)

# 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

# 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.

# 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.

# Objectivos e Metodologias

## Ferramentas e Tecnologias

O IDE utilizado para a realização deste projeto foi o **VS Code**. Á 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.

## 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.

# Trabalho Desenvolvido

## 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 |

## Classes e *Packages*

* ***Classe servidor***

A classe **servidor** implementa um servidor **TCP** (**Transmission Control Protoco**l) 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.

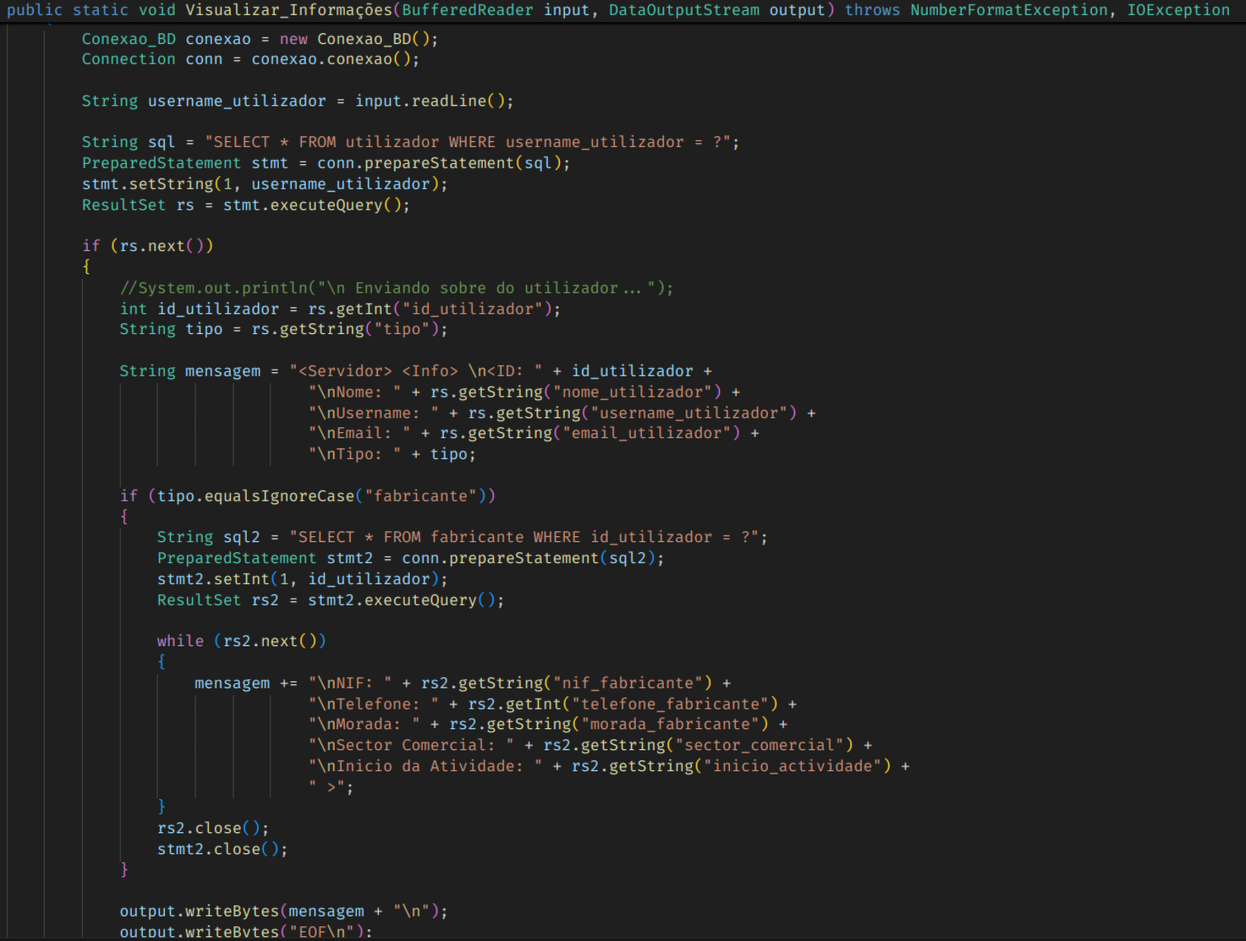


Figura 1: Método Visualizar\_informações

* **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.

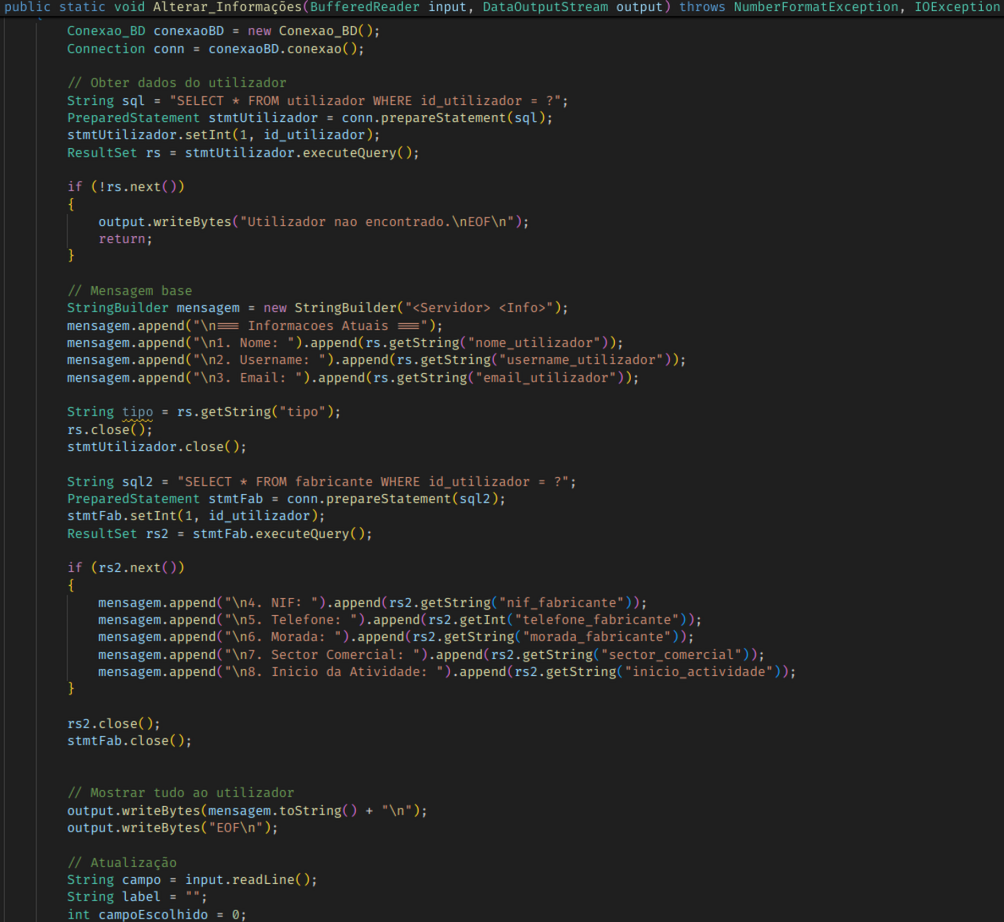


Figura 2: Método Alterar\_Informações 1

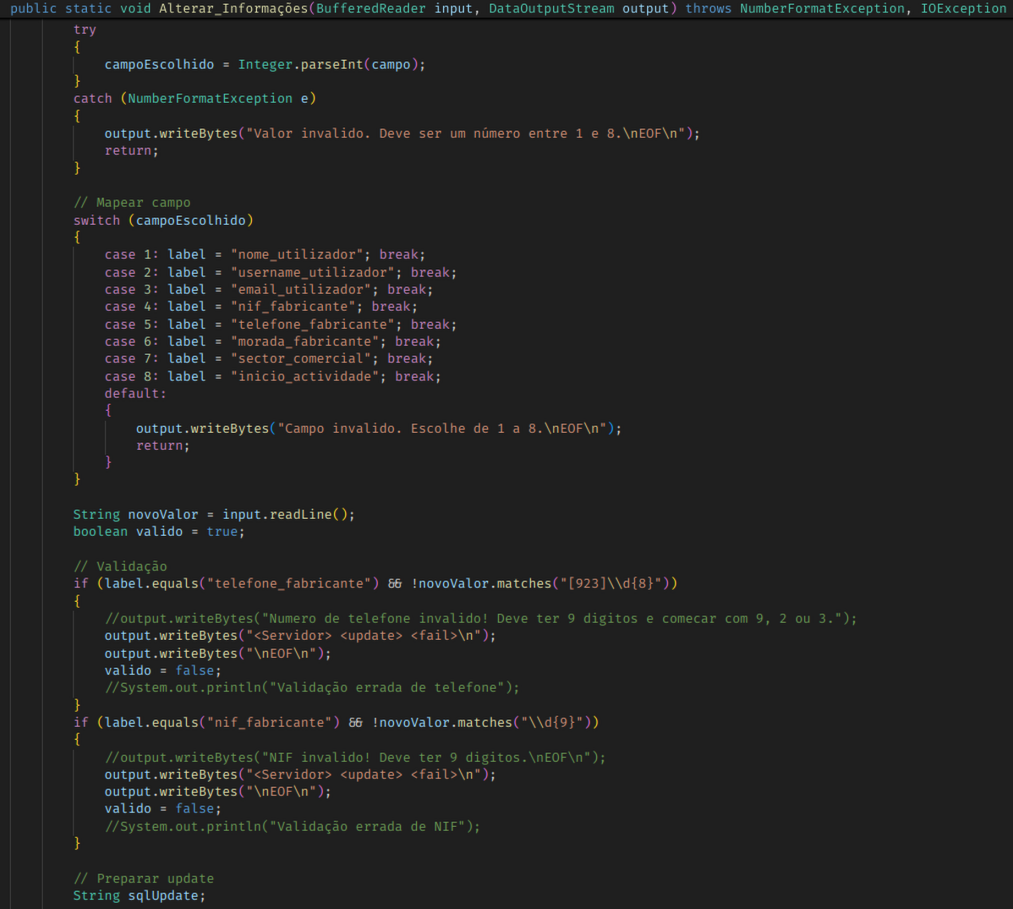


Figura 3: Método Alterar\_Informações 2

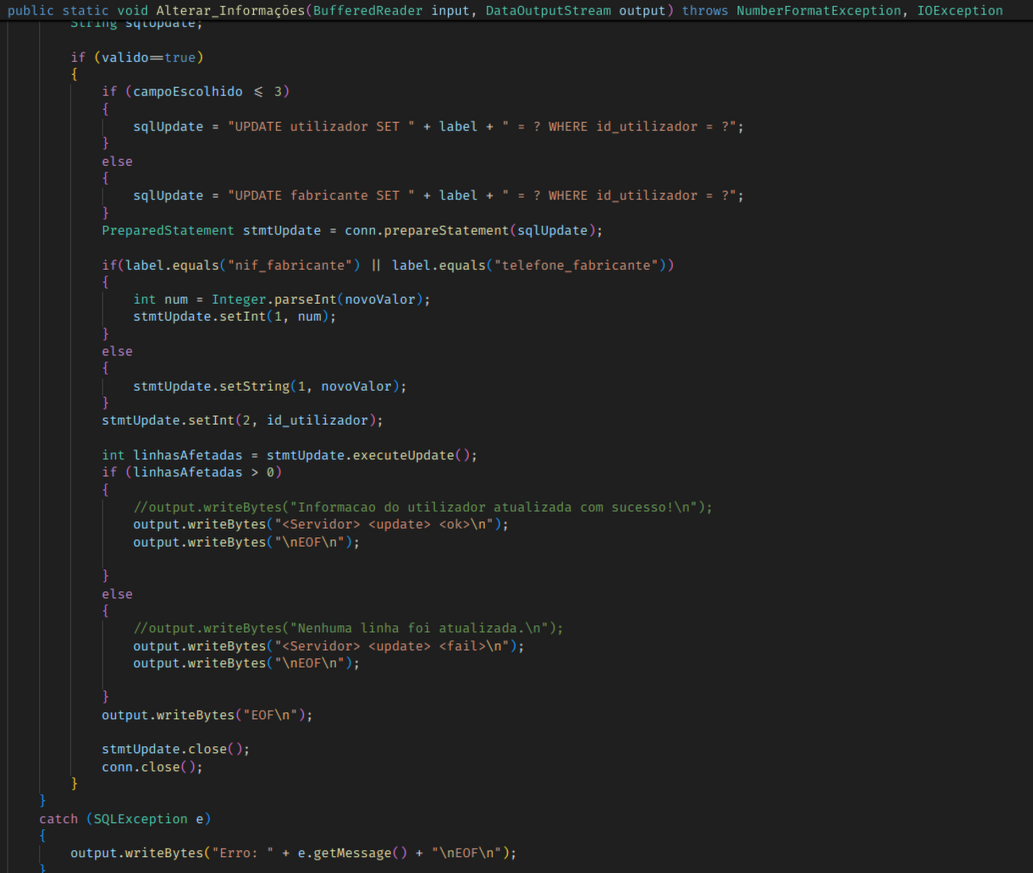


Figura 4: Método Alterar\_Informações 3

* **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 (I**D 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.

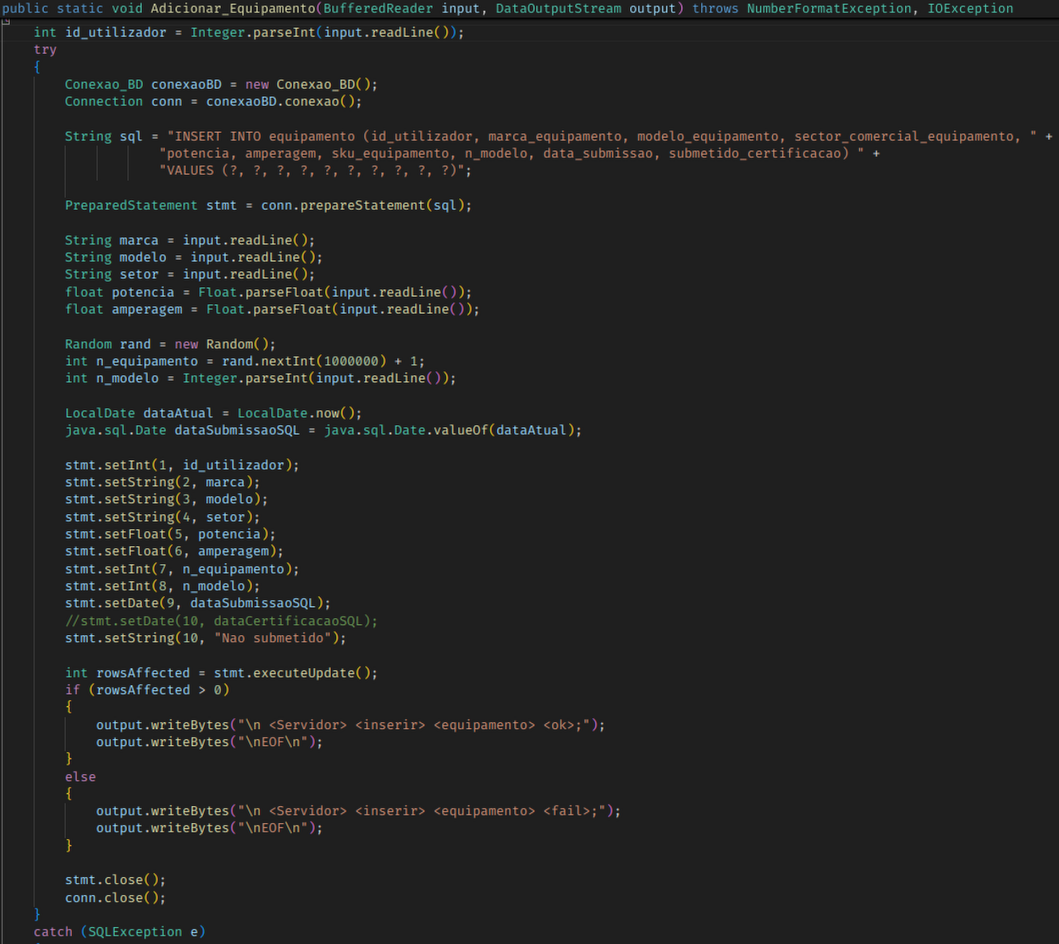


Figura 5: Método Adicionar\_Equipamento

* **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.

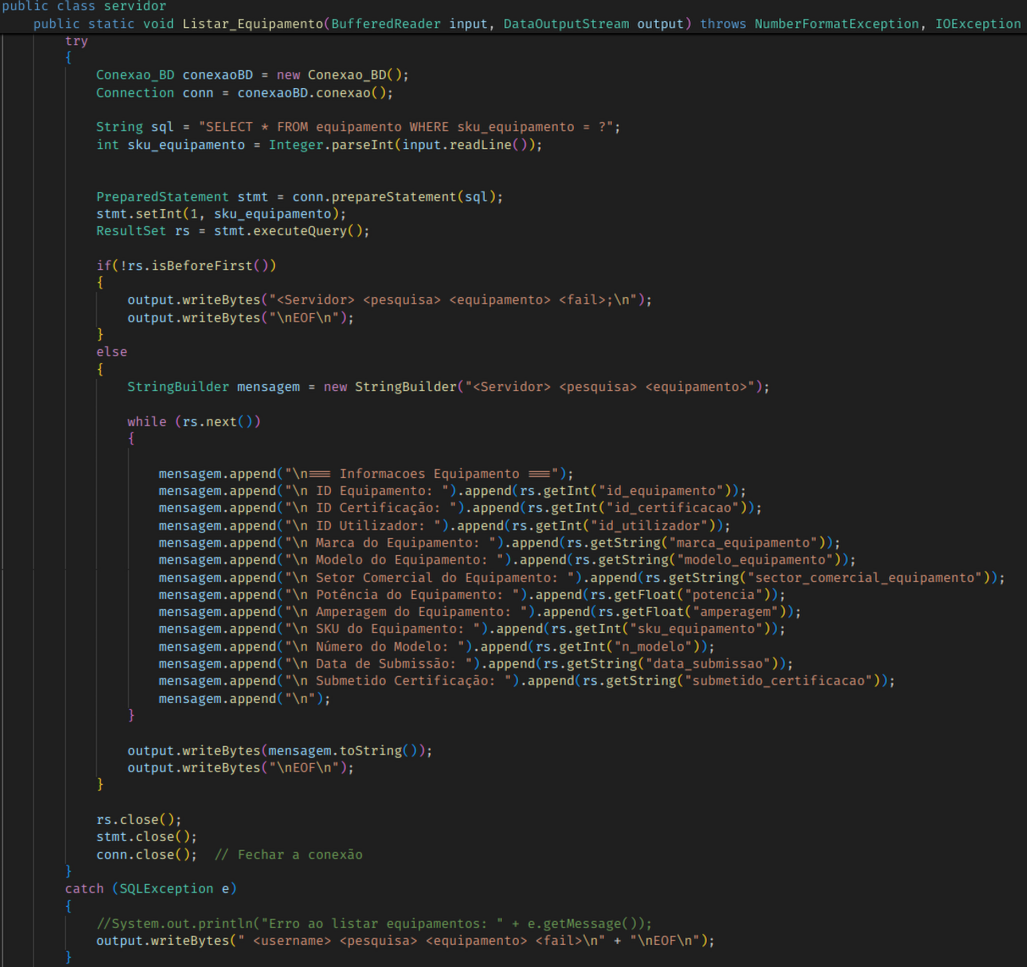


Figura 6: Método Listar\_Equipamentos

* **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.

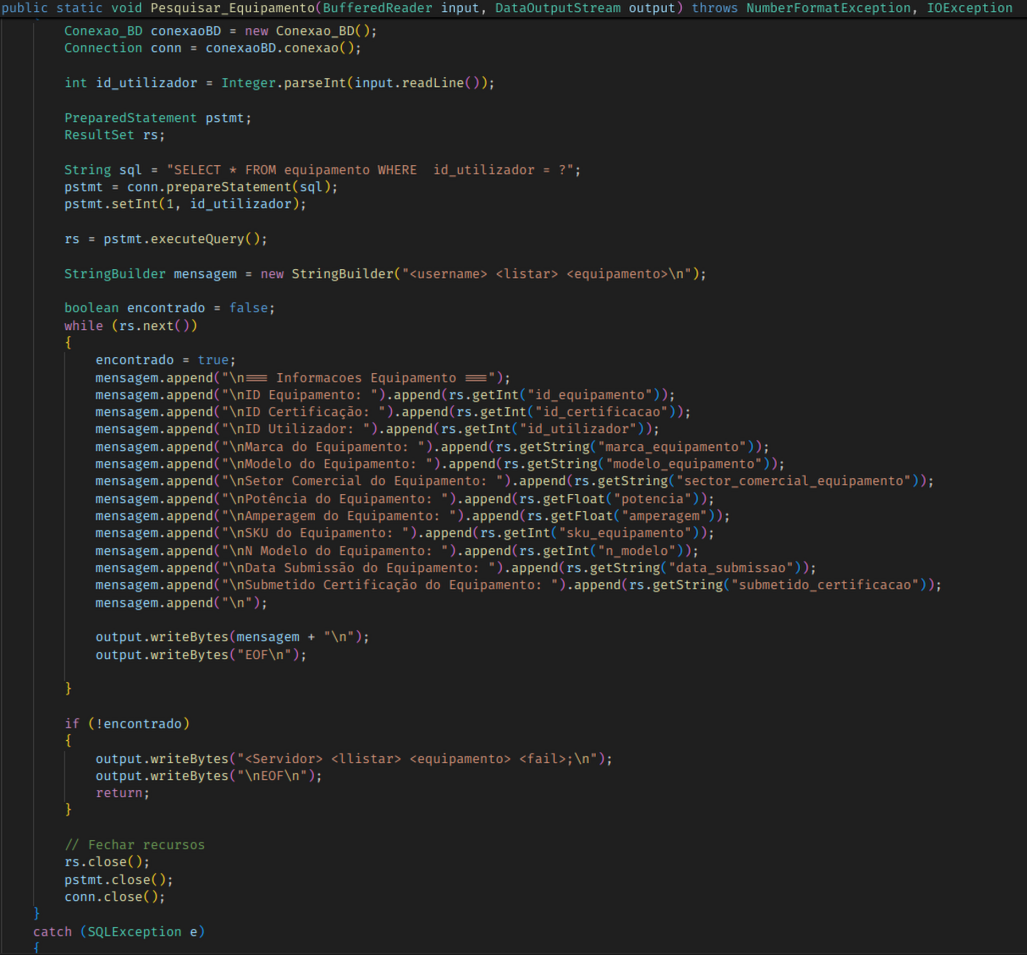


Figura 7: Método Pesquisar\_Equipamento

* **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 (I**D 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.

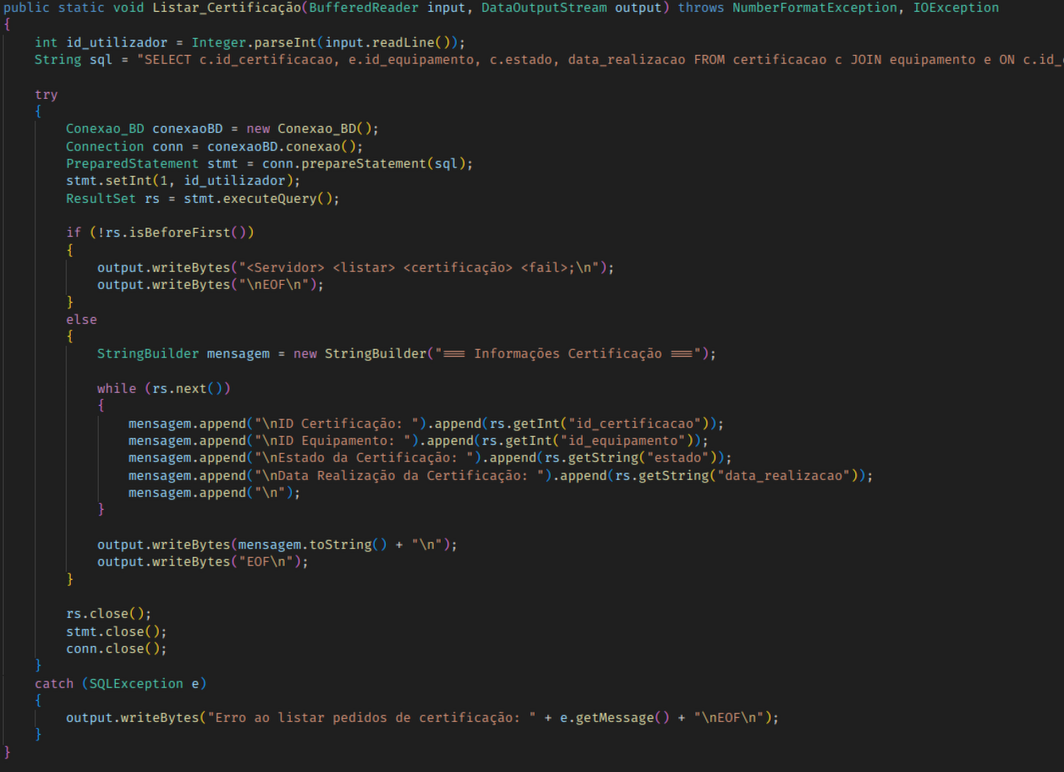


Figura 8: Método Listar\_Certificação

* **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.

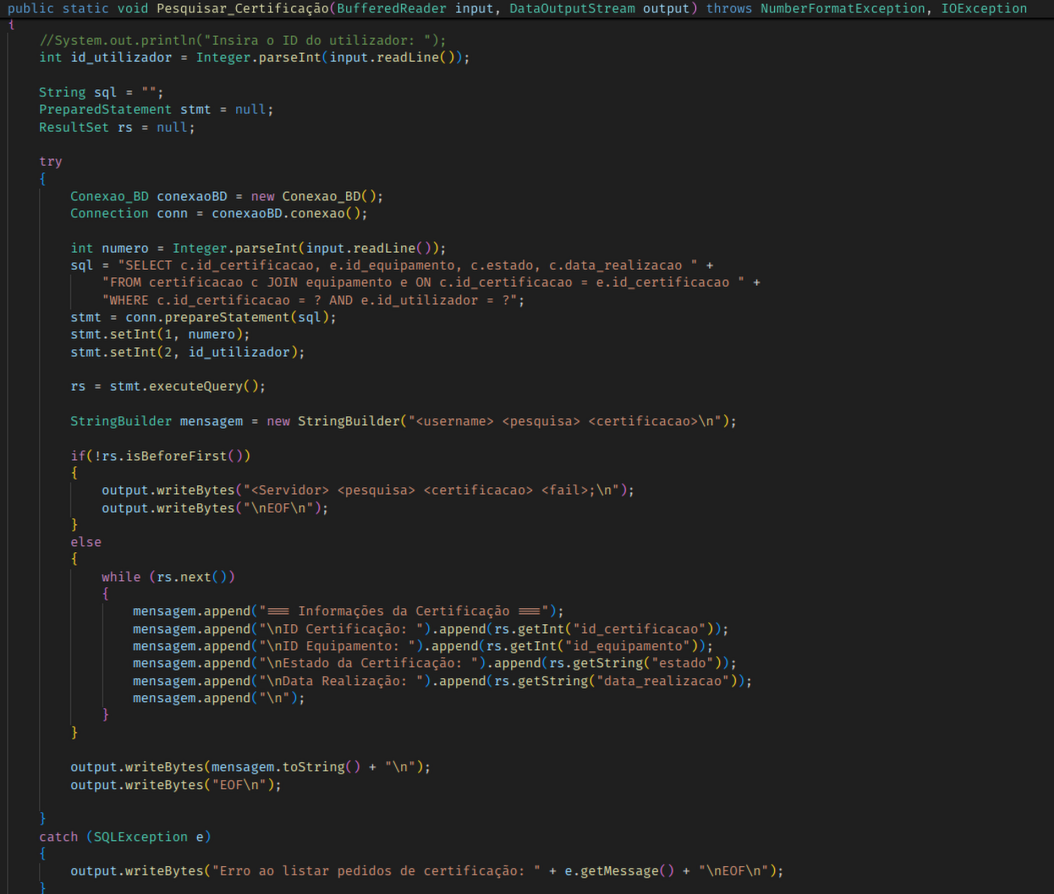


Figura 9: Método Pesquisar\_Certificação

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.



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.

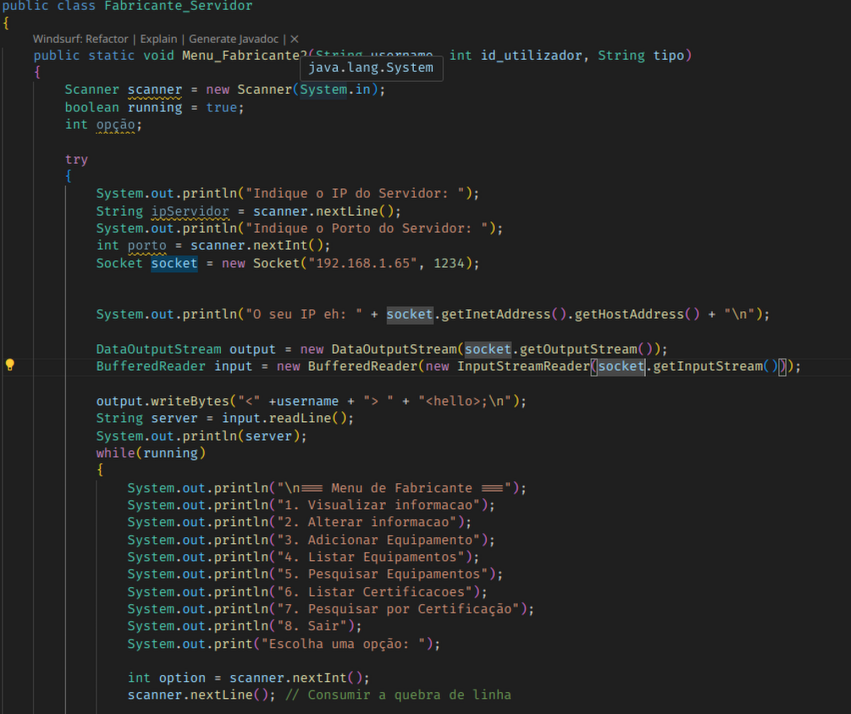


Figura 11: Classe Fabricante\_Servidor 1



Figura 12: Classe Fabricante\_Servidor 2

## 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

## 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.

## Armazenamento de Dados

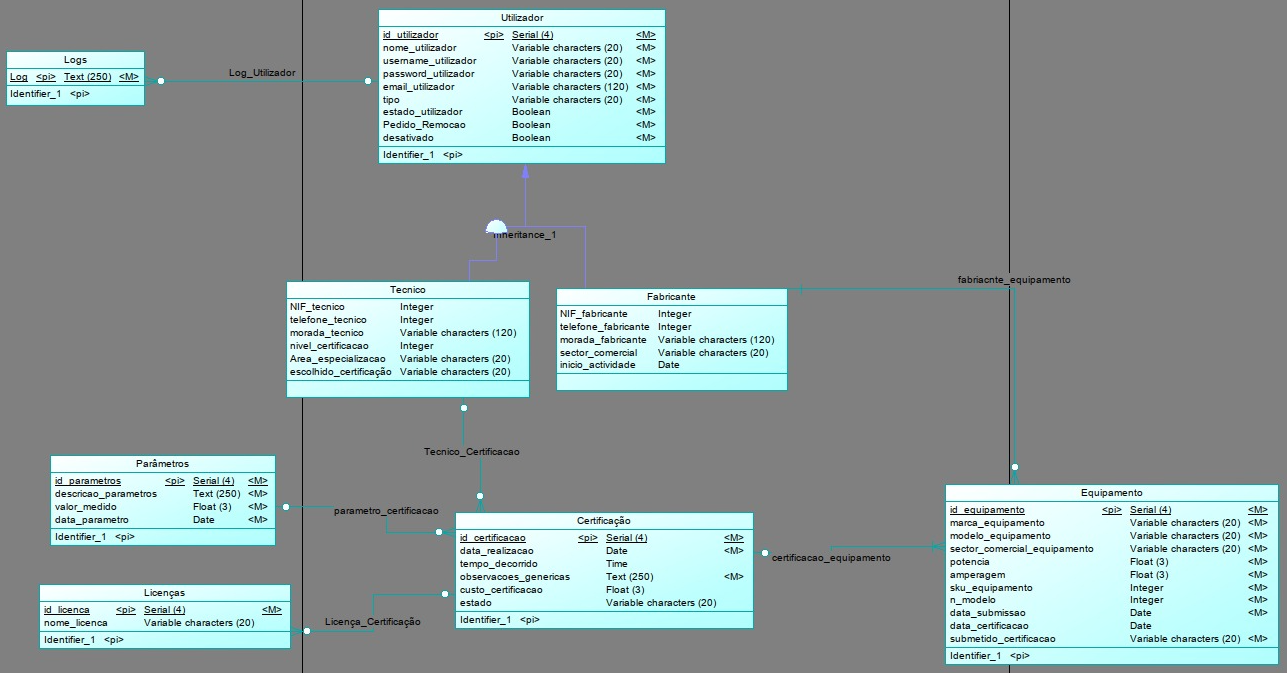


Figura 13: Modelo Entidade-Relacionamento

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.

1. 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.

1. Outras Tabelas de Suporte

Logs: Mantém registros de ações dos utilizadores.

Licenças: Representa licenças associadas a certificações.

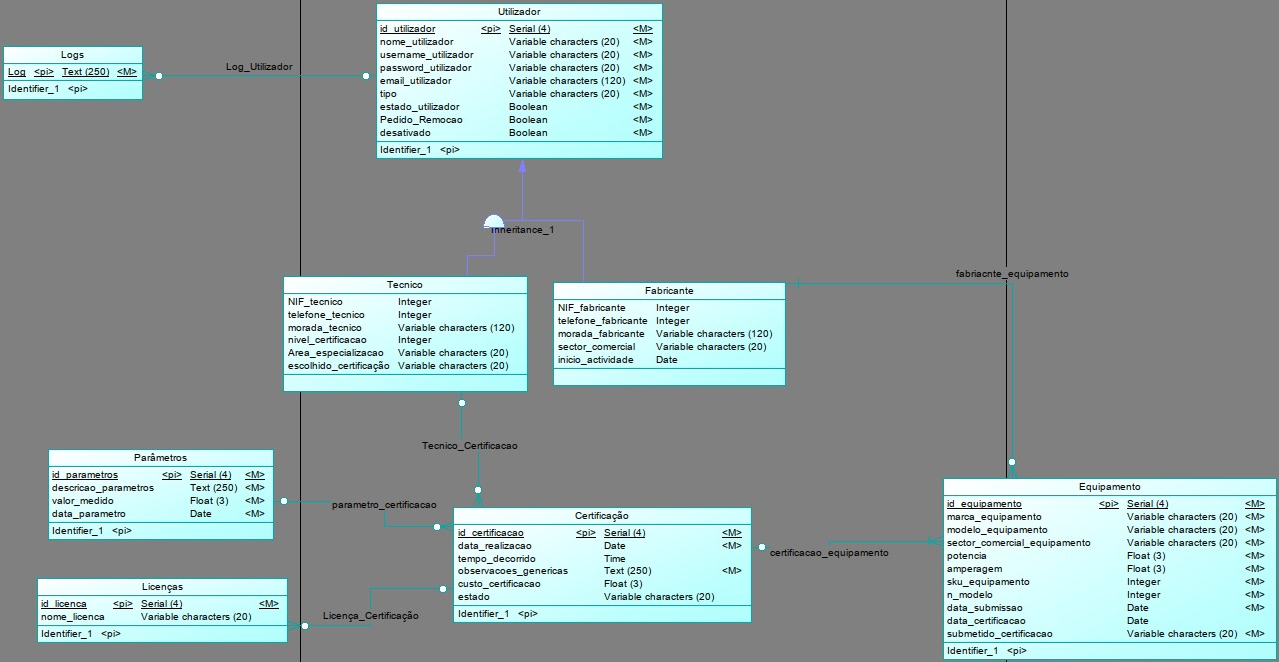


Figura 14: Modelo Físico

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;

## 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.

# Conclusões

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

## 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.

## 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.

## Trabalho Futuro

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

# Referências

## Lista de Referências