MANUAL DE MANUTENÇÃO

MLC1

2023

Sumário

1 Arquitetura

O sistema MLC1 segue uma arquitetura baseada no Framework EE, que proporciona recursos para o desenvolvimento de funcionalidades como envio, recebimento e tratamento de mensagens, acesso e persistência em banco de dados, e comunicação via Web Services utilizando o modelo REST.

A arquitetura do MLC1 é composta por um *back-end* desenvolvido em Java 8, um *front-end* baseado em Angular usando o Node 14.20.0, como será mais bem descrito no tópico 6.1.1, integração com o banco de dados Oracle e a implantação em um servidor de aplicação *WildFly* para o ambiente de Desenvolvimento e *WebSphere* para Produção.

Essa arquitetura é projetada para suportar as operações de Lingotamento Contínuo da Máquina 1, permitindo o gerenciamento eficiente das atividades e o fornecimento de informações em tempo real para os usuários.

1.1 *Back-end*

Banco de Dados, Escrita e Persistência

Começaremos com algumas informações sobre a estrutura de conexão com o banco de dados, a persistência dos dados (descrita em mais detalhes na sessão ***Persistence***) e o uso do JMS (Java Message Service). Vamos abordar cada ponto separadamente:

a) JPA, JTA e Banco de Dados:

JPA (Java Persistence API): É uma especificação da plataforma Java EE (Enterprise Edition) que define um conjunto de interfaces e anotações para padronizar o mapeamento objeto-relacional (ORM). O JPA permite que os desenvolvedores usem objetos Java para interagir com o banco de dados, abstraindo a complexidade das operações de persistência. Nesse contexto, o código usa a implementação do JPA fornecida pelo Hibernate (`***org.hibernate.ejb.HibernatePersistence***`). No ambiente de produção utiliza-se o **EclipseLink** como implementação do JPA.

JTA (Java Transaction API): É uma API que permite o gerenciamento de transações distribuídas em aplicações Java. Com o JTA, é possível garantir a atomicidade, consistência, isolamento e durabilidade (ACID) das transações, mesmo quando envolvem múltiplos recursos distribuídos. Nesse caso, a configuração indica o uso do JTA para gerenciar as transações (`***transaction-type="JTA"***`).

Banco de Dados: A fonte de dados é configurada, no ambiente de desenvolvimento, em servidores de aplicativos como o JBoss, como descrito acima, utilizamos o *WildFly*, já que o recurso é referenciado por `***java:/jboss/D_MLC_ACI***`. O JPA, através do Hibernate, é responsável por interagir com o banco de dados, mapeando as classes Java (`***com.arcelormittal.framework.core.entity***`, `***com.arcelormittal.mlc.core.entity***`, etc.) para as tabelas correspondentes.

O código apresentado nos trechos acima configura o ambiente de persistência do aplicativo com o uso do JPA (Java Persistence API) e do JTA (Java Transaction API), bem como a configuração de classes de entidade para mapeamento de objetos Java para tabelas do banco de dados. Além disso,

fazemos o uso de JMS (Java Message Service) para comunicação assíncrona, mais detalhes de sobre como o JMS é implementado e utilizado no projeto serão fornecidos na seção JMS, item 1.1.1.

b) Escrita e Persistência de Banco:

Escrita de Banco: A escrita no banco de dados é realizada quando ocorrem operações de persistência, ou seja, quando novos objetos Java são salvos ou atualizados no banco de dados. Isso geralmente é feito através dos métodos fornecidos pela API JPA, conforme explicado acima.

Persistência de Banco: A persistência de banco se refere à capacidade de manter os dados no banco de dados de forma duradoura, mesmo após o término da execução da aplicação. Através do JPA, as entidades Java (***`com.arcelormittal.framework.core.entity`***, ***`com.arcelormittal.mlc.core.entity`***, etc.) são mapeadas para entidades do banco de dados, permitindo que os dados sejam recuperados e atualizados quando necessário.

Criação, Atualização, Consulta e Exclusão no Banco: Todas essas operações são realizadas através das operações padrão de CRUD (Create, Read, Update, Delete) disponibilizadas pelo JPA. Essas operações são mapeadas para instruções SQL apropriadas pelo provedor JPA (Hibernate, neste caso) e executadas no banco de dados.

build.gradle (:MLC1AgendadorTarefas)

```
apply plugin: 'environments'

dependencies {
    compile project(':MLC1Core')
    compile project(':MLCRestApiCore')
}

processResources.dependsOn(refreshAllEnvironments)
environments {
    Main {
        sourceDir = "environments/"
        destinationDir = "/"
        defaultEnvironment = EnvDefaultEnvironment
        alwaysOverwrite = true
    }
}

task finalize(type: Copy) {
    from('build/libs/') {
        include '**/*.war'
    }
```

Figura 1 - BUILD.GRADLE BACK-END

Introdução ao *Back-end*

O *back-end* do MLC1 é organizado em vários projetos e subprojetos, cada um com uma função específica dentro do sistema, alguns detalhes importantes estão dispostos no build.gradle, Figura acima, de cada Subprojeto, por exemplo, se o módulo em questão é implantável ou não, se ele gera

um .war, quais artefatos são necessários para compilar e testar cada um deles, utilizando o *WildFly*, por exemplo. A saber, os módulos que compõem a MLC1, são:

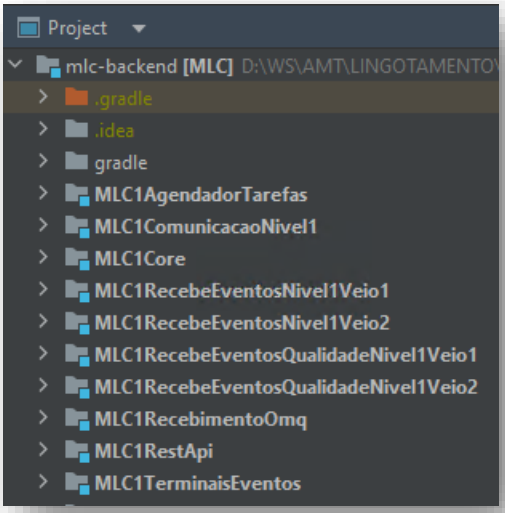


Figura 2 - ESTRUTURA PROJETO SUBPROJETOS/MÓDULOS BACK-END

- **MLC1AgendadorTarefas**: Módulo responsável por classes de monitoramentos e tratamento de processos periódicos, como as corridas no molde, caso de uso Rastreamento de Corrida e modelo de queda da temperatura.

  Módulo implantável: Sim.
  Dependências necessárias: MLC1Core e MLCRestApiCore.
  Usa *environments*: Sim.

- **MLC1ComunicacaoNivel1**: Projeto dedicado à comunicação em tempo real com o nível 1 do processo de lingotamento contínuo da Máquina 1 e classes com serviço para tratar os eventos recebidos do nível 1 referentes ao processo de Rastreamento de placas para os veios 1 e 2, Serviço para tratar os eventos recebidos do nível 1 que geram eventos de qualidade.

  Módulo implantável: Sim.
  Dependências necessárias: MLCComunicacaoNivel1Core, MLC1Core e MLCRestApiCore.
  Usa *environments*: Sim.

- **MLC1Core**: "Build de Referência", projeto raiz do MLC1, responsável por coordenar os demais subprojetos e fornecer funcionalidades gerais do sistema. Contém ENUNS, MANAGER, OMQ e ESCRITA NIVEL 1, responsáveis, entre outros, pela Entidade para tratar regras de negócio relativas ao Rastreamento de Placas, implementa os métodos para tratar o peso bruto da panela na torre e para o envio dos dados para o nível 1 para os eventos da torre.

Módulo implantável: Não.
Dependências necessárias: MLCCore e MLCMensagensCore.
Usa *environments*: Não.

- **MLC1RecebeEventosNivel1Veio1**: Módulo responsável por receber eventos do nível 1 para o veio 1, responsáveis, entre outros, pelo Serviço para tratar os eventos recebidos do nível 1 para a Máquina 1, tratamentos de processos periódicos do sistema e do processo periódico dos dados de pó fluxante, também do Scheduler que irá monitorar o contador de lingotamento e movimentar o veio e outros processos periódicos de caso de uso de Cálculos periódicos do veio 1.

  Módulo implantável: Sim.
  Dependências necessárias: MLC1Core, MLCRestApiCore e MLCComunicacaoNivel1Core.
  Usa *environments*: Sim.

- **MLC1RecebeEventosNivel1Veio2**: Módulo responsável por receber eventos do nível 1 para o veio 2, responsáveis, entre outros, pelo Serviço para tratar os eventos recebidos do nível 1 para a Máquina 1, tratamentos de processos periódicos do sistema e do processo periódico dos dados de pó fluxante, também do Scheduler que irá monitorar o contador de lingotamento e movimentar o veio e outros processos periódicos de caso de uso de Cálculos periódicos do veio 2.

  Módulo implantável: Sim.
  Dependências necessárias: MLC1Core, MLCRestApiCore e MLCComunicacaoNivel1Core.
  Usa *environments*: Sim.

- **MLC1RecebeEventosQualidadeNivel1Veio1**: Módulo responsável por receber eventos de qualidade do nível 1 para o veio 1, responsáveis, entre outros, pelo Serviço para tratar os eventos de qualidade recebidos do nível 1 para o veio 1 da Máquina 1, também do Scheduler que irá monitorar o contador de lingotamento e movimentar o veio 1.

  Módulo implantável: Sim.
  Dependências necessárias: MLC1Core, MLCRestApiCore e MLCComunicacaoNivel1Core.
  Usa *environments*: Sim.

- **MLC1RecebeEventosQualidadeNivel1Veio2**: Módulo responsável por receber eventos de qualidade do nível 1 para o veio 2, responsáveis, entre outros, pelo Serviço para tratar os eventos de qualidade recebidos do nível 1 para o veio 2 da Máquina 1, também do Scheduler que irá monitorar o contador de lingotamento e movimentar o veio 2

  Módulo implantável: Sim.
  Dependências necessárias: MLC1Core, MLCRestApiCore e MLCComunicacaoNivel1Core.

Usa *environments*: Sim.

- **MLC1RecebimentoOmq**: Módulo responsável pelo recebimento de mensagens OMQ do sistema. Contém BOF, CPCS, MLC1, MLC2, SLABID responsáveis, entre outros, pelo tratamento das mensagens específicas.

  Módulo implantável: Sim.
  Dependências necessárias: MLCRecebimentoOmqCore, MLC1Core e MLCRestApiCore.
  Usa *environments*: Sim.

- **MLC1RestApi**: Projeto responsável pela criação de uma API REST para o MLC1, permitindo a comunicação com outros sistemas e aplicações externas.

  Módulo implantável: Sim.
  Dependências necessárias: MLC1Core, MLCRestApiCore e 'com.arcelormittal.frameworkee:FrameworkEERestAPI:' + project.ext.frameworkEEVersion.
  Usa *environments*: Sim

- **MLC1TerminaisEventos**: Módulo com atribuições e Classes responsáveis pelo *Application*, que irá interceptar as requisições CORS, inicialização dos serviços, além de outras classes que fazem o recebimento das mensagens na fila da gateway e envia para o SSE, realizando o broadcast e pela criação do serviço de broadcast dos eventos via SSE.

  Módulo implantável: Sim.
  Dependências necessárias: Temos aqui, uma peculiaridade, em que dependendo do ambiente que se utilize este módulo, ele utilizará dependências externas, conforme o caso. Mostrado na imagem abaixo.

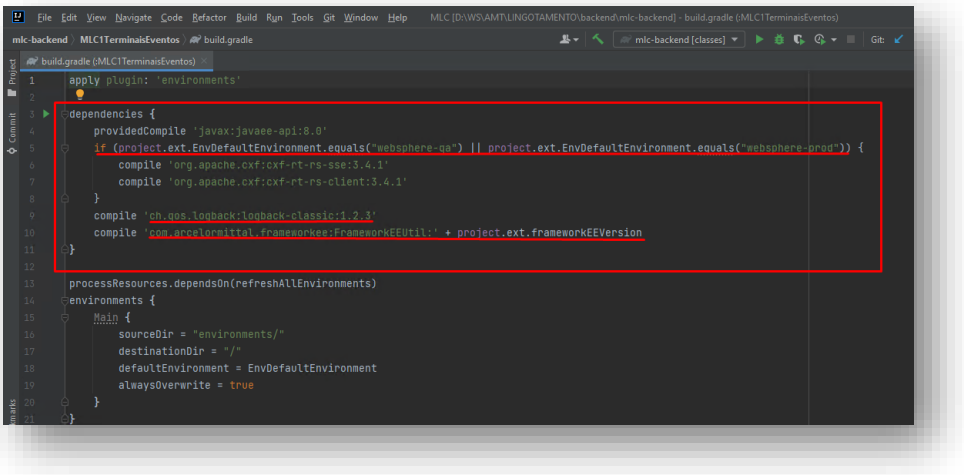


Figura 3 - DEPENDÊNCIAS EXTERNAS DO MLC1TerminaisEventos

Usa *environments*: Sim.

Environments

Localizados em cada um dos módulos descritos acima, temos um pacote **environments**, seguido dos pacotes para configuração de ambiente e tecnologia específica de cada servidor. Entendemos *WildFly-dev* como ambiente de desenvolvimento local, *WildFly-qa* é o ambiente de desenvolvimento da contratada, *WebSphere-qa* é o ambiente para desenvolvimento e testes da ArcelorMittal, e *WebSphere-prod* é o ambiente final, onde o sistema roda e opera para utilização dos usuários.

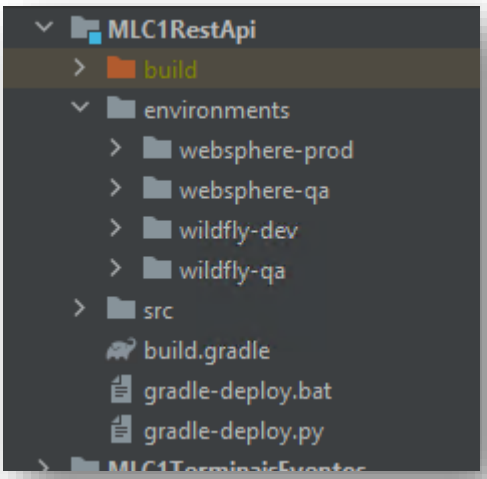


Figura 4 - ESTRUTURA ENVIRONMENTS BACK-END

O IBM *WebSphere* e o *WildFly* são servidores de aplicação Java que fornecem um ambiente de execução para aplicativos corporativos. Ambos os servidores são amplamente utilizados na indústria e possuem recursos avançados para desenvolvimento, implantação e gerenciamento de aplicativos Java. Se usa *WildFly* para teste e desenvolvimento no ambiente da contratada e *WebSphere* na ArcelorMittal. E como implementação do JPA a contratada utiliza **Hibernate** e a ArcelorMittal utiliza o **EclipseLink**, entre outras, com funções similares.

Notas:

a) *WebSphere*

O IBM *WebSphere* Application Server (ou simplesmente *WebSphere*) é um servidor de aplicação Java desenvolvido pela IBM. Ele oferece suporte a uma ampla gama de tecnologias e

recursos para implantação de aplicativos empresariais de grande escala. Alguns dos principais recursos do *WebSphere* incluem:

Escalabilidade: O *WebSphere* é conhecido por oferecer suporte a implantações de alto desempenho e escalabilidade. Ele possui recursos avançados para balanceamento de carga, clustering e dimensionamento horizontal.

Ferramentas e integração: O *WebSphere* possui uma ampla gama de ferramentas de desenvolvimento e integração, como o IBM Rational Application Developer (RAD), que oferece recursos de desenvolvimento e depuração avançados.

Suporte corporativo: O *WebSphere* é amplamente utilizado em ambientes corporativos e possui um alto nível de suporte da IBM. Ele oferece recursos de gerenciamento e monitoramento avançados, bem como suporte técnico especializado.

b) *WildFly*

O *WildFly*, é um servidor de aplicação Java de código aberto desenvolvido pela Red Hat. Ele é uma implementação do Java Enterprise Edition (Java EE) e oferece recursos avançados para implantação de aplicativos empresariais. Algumas características importantes do *WildFly* incluem:

Leve e rápido: O *WildFly* é conhecido por sua natureza leve e rápido tempo de inicialização. Ele oferece um ambiente de execução ágil e escalável para aplicativos Java.

Modularidade: O *WildFly* possui uma arquitetura modular baseada no conceito de serviços Java EE. Isso permite que os desenvolvedores escolham apenas os serviços necessários para seus aplicativos, reduzindo a sobrecarga e melhorando o desempenho.

Administração e monitoramento: O *WildFly* oferece uma interface de administração baseada na web (*WildFly* Management Console) para configurar, implantar e monitorar aplicativos. Além disso, ele possui recursos de gerenciamento avançados, como implantação e gerenciamento em tempo de execução (hot deployment) e clustering.

Principais diferenças:

Embora o *WebSphere* e o *WildFly* compartilhem algumas características semelhantes como servidores de aplicação Java, existem diferenças significativas entre eles. Algumas das principais diferenças incluem:

Licenciamento: O *WebSphere* é um produto comercial e requer licenciamento, enquanto o *WildFly* é de código aberto e licenciado sob a Licença Pública Geral GNU (GNU GPL).

Tamanho e recursos: O *WebSphere* é geralmente considerado um servidor de aplicação mais robusto e completo, com uma ampla gama de recursos e suporte corporativo abrangente. Por outro lado, o *WildFly* é conhecido por ser mais leve, rápido e modular, permitindo uma implantação mais personalizada.

Comunidade e suporte: O *WebSphere* é desenvolvido e mantido pela IBM, uma empresa líder no setor de tecnologia, com suporte técnico especializado disponível. O *WildFly* é um projeto de código aberto, com uma comunidade ativa e suporte geralmente fornecido por meio de fóruns, listas de discussão e documentação comunitária.

A escolha entre o *WebSphere* e o *WildFly* depende dos requisitos específicos do projeto, tamanho da implantação, recursos necessários e preferências da organização. Cada servidor de aplicação tem suas vantagens e considerações únicas, e é importante avaliar cuidadosamente as necessidades do projeto antes de decidir qual utilizar.

Persistence.xml

Localizados na META-INF, dentro da *resources* das "*DEV*" do *environments* de cada subprojeto mencionados acima, estão suas respectivas persistences.xml, utilizaremos a seguir, como exemplo, a persistence do **MLC1AgendadorTarefas**, que é uma configuração de persistência para um sistema que utiliza a tecnologia de mapeamento objeto-relacional (ORM). Especificamente, contém informações de JPA / JTA, o LINK das entidades relacionadas, ele define a configuração de persistência para o provedor de persistência *Hibernate* em uma unidade de persistência chamada "MLCPU".

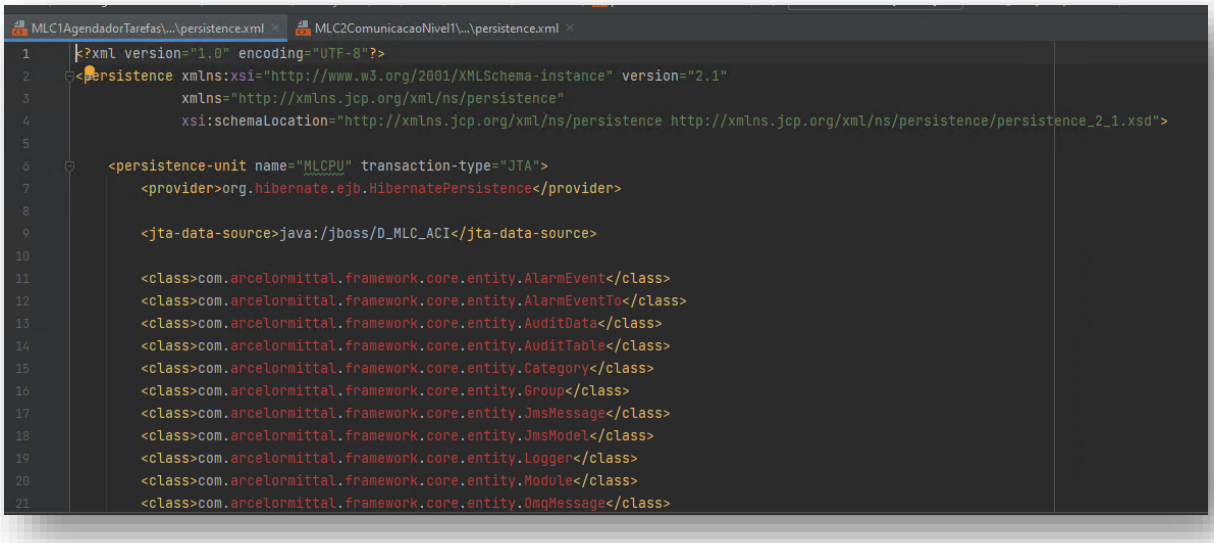


Figura 5 - PERSISTENCE DO AGENDADOR DE TAREFAS MLC1 BACK-END

Aqui está uma explicação dos principais elementos no arquivo:

- A declaração ***<?xml version="1.0" encoding="UTF-8"?>*** define a versão do XML e a codificação utilizada.
- O elemento raiz ***<persistence>*** contém a configuração global de persistência.
- O atributo ***xmlns:xsi*** define o *namespace* para o atributo ***xsi:schemaLocation***.
- O atributo ***version="2.1"*** especifica a versão da configuração de persistência.
- O atributo ***xmlns="http://xmlns.jcp.org/xml/ns/persistence"*** define o *namespace* padrão para os elementos dentro do arquivo.
- O atributo ***xsi:schemaLocation*** especifica a localização do esquema XSD usado para validar o XML.
- O elemento ***<persistence-unit>*** define uma unidade de persistência. Ele contém várias configurações relacionadas à persistência.
- O atributo ***name="MLCPU***" define o nome da unidade de persistência.

- O atributo ***transaction-type="JTA"*** especifica o tipo de transação utilizado (*Java Transaction API*).
- O elemento ***<provider>*** especifica o provedor de persistência, que neste caso é o *Hibernate*.
- O elemento ***<jta-data-source>*** define a origem de dados JTA usada para conexão com o banco de dados. Os elementos ***<class>*** listam as classes de entidade que serão gerenciadas pela unidade de persistência. Cada classe representa uma tabela ou entidade no banco de dados.
  Essa configuração descreve a estrutura do banco de dados e as entidades mapeadas para ele. É usada pelo provedor de persistência para **criar**, **atualizar**, **consultar** e **excluir** objetos persistentes no banco de dados com base nas classes de entidade definidas.
  A consulta de entidades na persistência é feita utilizando a API do *Hibernate*, que é o provedor de persistência mencionado em nosso documento. O *Hibernate* oferece uma série de recursos para consultar e manipular objetos persistentes.
  Após configurar corretamente a unidade de persistência e definir as classes de entidade no arquivo de configuração apresentado, você pode usar a *API* do *Hibernate* para executar consultas.

Cada projeto possui suas próprias características e funcionalidades exclusivas. Eles compartilham uma biblioteca base chamada "MLCCore", que contém propriedades comuns a todos os projetos, mapeamento de entidades, objetos DTO (Data Transfer Objects) utilizados na troca de dados e funcionalidades gerais que são aplicáveis em todo o sistema.

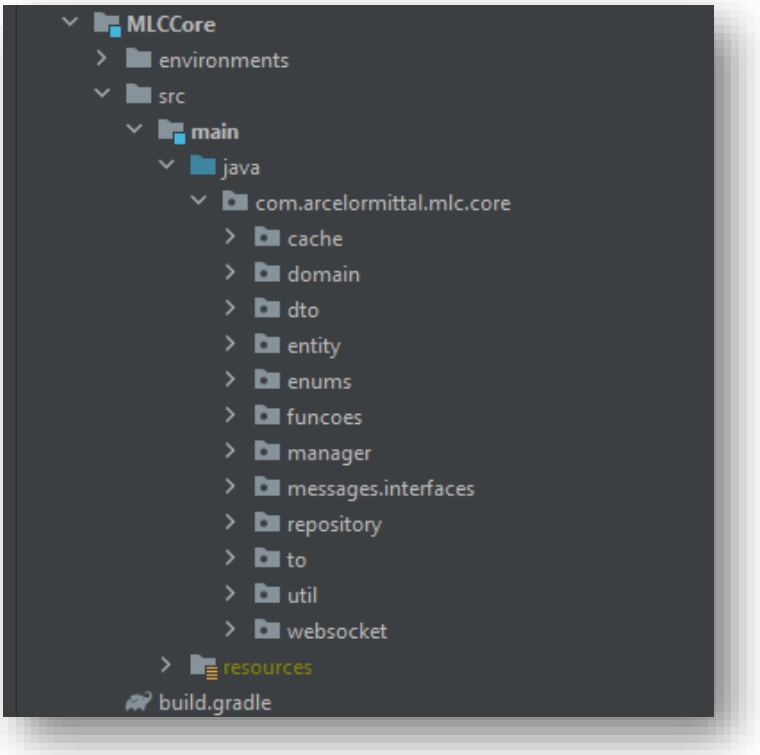


Figura 6 – ESTRUTURA DA BIBLIOTECA BASE MLCCORE

Essa abordagem modular permite uma maior reutilização de código e uma organização eficiente das funcionalidades específicas de cada projeto do MLC1.

O sistema MLC1 faz ainda o uso de duas bibliotecas base essenciais:

- **MLCCore**: É uma biblioteca compartilhada entre todos os projetos do MLC. Ela contém propriedades comuns, mapeamento de entidades e objetos DTO utilizados para troca de dados entre os sistemas. Além disso, ela oferece funcionalidades gerais que podem ser utilizadas em todos os projetos do MLC, como transformação de objetos em formatos específicos (por exemplo, XML).
- **MLC1Core**: Essa biblioteca é importada e utilizada exclusivamente pelos sistemas relacionados às máquinas de lingotamento contínuo 3. Ela contém controladores responsáveis por receber requisições HTTP, mensagens recebidas pelas máquinas de diferentes fontes, objetos DTO exclusivos da Máquina 1 e classes de repositórios que implementam consultas específicas ao banco de dados relacionadas também a MLC1.

Essas bibliotecas base fornecem uma base sólida para o desenvolvimento e garantem a consistência das funcionalidades comuns em todo o sistema, incluindo a MLC1.

A decisão de algo ser parte do core do sistema é baseada na funcionalidade e na sua importância para o sistema como um todo. O core geralmente contém as funcionalidades principais e essenciais que são comuns a todas as máquinas. Essas funcionalidades devem ser implementadas de forma genérica, de modo que possam ser reutilizadas por todas as máquinas do sistema.

O que determina algo ser do core ou não, basicamente, é aquilo que é utilizado em comum, aquilo que precisa estar nas regras de todo o sistema, casos utilizados, tratados ou chamados pela MLC1, MLC2 e/ou MLC1, e para ser específico de cada máquina, o tratamento deve ser exclusivo para ela. Temos o exemplo das TAGs de evento, que é separado por máquina, mas geralmente temos as classes e os métodos que tratam esses eventos iguais para todas as máquinas, logo, os eventos das TAGS são exclusivos, mas os métodos de tratamento são comuns, ficando estes últimos à cargo do Core.

Para criação de algo no core, temos que criar a interface em MLCCORE, depois criar a classe específica para cada máquina "Ex. MLC1" implementando essa interface mencionada, do core.

No entanto, cada máquina também pode ter requisitos específicos que não são aplicáveis a todas as outras máquinas. Nesses casos, é necessário implementar esses requisitos específicos em cada máquina individualmente, de acordo com suas necessidades.

Resumindo, para configurar a especialização de um processo em relação ao core, é necessário criar uma interface no core que define os métodos e as funcionalidades comuns que cada máquina deve implementar. Cada máquina específica então implementa essa interface e fornece sua própria lógica personalizada. Isso permite que o core se comunique com as diferentes máquinas de forma padronizada e execute as operações necessárias em cada uma delas.

Inserindo uma TAG

Uma TAG para o sistema MLC1 é um endereço de memória do PLC, aponta para um valor a ser lido ou escrito pelo Nível 1.

É preciso inserir a TAG no OPC, em nosso caso, utilizamos o *Kepserver*, que é o servidor utilizado no ambiente de desenvolvimento da contratada, responsável por armazenar, receber e enviar as TAGs, fica conectado no servidor, depois disto criamos o caminho dela no Banco de Dados para o sistema entender que ela é uma TAG do nosso sistema.

Utilizando TAGs:

Explicando três tipos básicos de TAGs, em nosso sistema, temos a TAG de Eventos, que possuí mapeamento através das classes.

Ex.: @Tag(value = {"EVENTO_QUALIDADE_VALVULA_LONGA_PROBLEMA"})

Quando houver alteração nessas TAGs, o código será acionado indicando o início de uma ação no código referente essa TAG, existem ainda o caso de recebermos o valor de algo, através de uma TAG.

Ex.: DADO_COMPRIMENTO_PLACA_VEIO_X

Essa TAG é preenchida no Nível1 e utilizada para estabelecer o comprimento da placa específica, quando necessário.

E por último tem a TAG que recebe valores, escrevemos nela, por assim dizer, onde geralmente o *Back-end* escreve algo para o *Front-end* trabalhar com essas informações.

EX.: SET_POINT_PANELA_PESO_TARA

Em suma, a utilização de TAGs no sistema MLC1 é essencial para estabelecer comunicação e intercâmbio de informações entre os diferentes níveis do sistema. As TAGs servem como endereços de memória no PLC, permitindo a leitura e escrita de valores pelo Nível 1. A integração dessas TAGs no OPC, através do servidor *Kepserver*, é fundamental para o funcionamento adequado do ambiente de desenvolvimento da contratada.

Apresentamos três tipos básicos de TAGs utilizadas no sistema. As TAGs de Eventos são mapeadas por meio de classes e acionam o código correspondente quando ocorre alguma alteração. As TAGs de recebimento de valores são utilizadas para obter informações específicas, como o comprimento de uma placa. Por fim, as TAGs que recebem valores são escritas para permitir a comunicação do *Back-end* com o *Front-end*, possibilitando o trabalho com as informações obtidas.

A adoção adequada e organizada de TAGs no sistema MLC1 contribui para a eficiência e a eficácia das operações, permitindo a troca de dados de forma clara e estruturada entre as diferentes partes do sistema.

Comunicação OPC e MQ MLC1

No caso específico da MLC1, é necessário considerar a comunicação com o *front-end*, OPC (OLE for Process Control) e MQ (Message Queue).

Comunicação com o *front-end* - O sistema MLC1 deve fornecer APIs ou serviços de *back-end* para permitir a comunicação e integração com o *front-end*. Feitas através de protocolo REST, usando

requisições como GET e POST, isso pode incluir a definição de serviços RESTful, endpoints específicos, troca de dados por meio de JSON ou XML, autenticação e autorização de usuários, etc.

Comunicação com OPC - O OPC é um padrão de comunicação amplamente utilizado em sistemas de automação e controle industrial. O MLC1 utiliza a comunicação com OPC para interagir com dispositivos, sensores e sistemas de controle industriais passado através do Nível 1. Isso pode envolver a leitura de dados de sensores, o envio de comandos de controle e a integração com sistemas de supervisão e controle. Como descrito anteriormente em 'Testando envios/recebimentos de mensagens na MLC1 '.

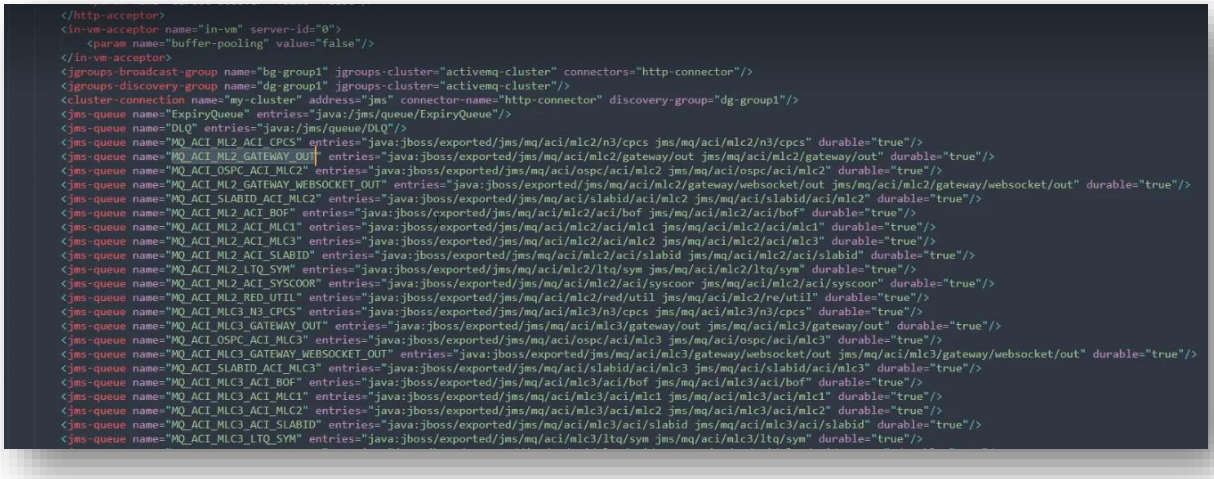


Figura 7 - STANDALONE CONTENDO AS FILAS DE JMS

Comunicação com MQ - O uso de um sistema de mensageria, como MQ, permite que o MLC1 se comunique de forma assíncrona e confiável com outros sistemas, módulos ou componentes. Isso pode envolver o envio e recebimento de mensagens para troca de informações, eventos ou notificações entre diferentes partes do sistema, na MLC1, utilizamos o JMS, que possui as filas configuradas no Standalone-full-ha e o restante, rodado pelo próprio Java, como será mais bem descrito, abaixo em nossa documentação.

1.1.1 Mensageria JMS

O sistema MLC1 utiliza a tecnologia de mensageria JMS (Java Message Service) para a troca assíncrona de mensagens entre os diferentes componentes do sistema. O JMS é um padrão de comunicação baseado em mensagens, que permite o envio e recebimento de mensagens de forma confiável e assíncrona.

No contexto do MLC1, a mensageria JMS é utilizada para o envio e recebimento de mensagens entre as máquinas de lingotamento contínuo, o sistema de controle e monitoramento, e outros sistemas externos que interagem com o MLC1, como os chamados de Nível1 e Nível3.

Através da JMS, é possível estabelecer uma comunicação assíncrona e distribuída entre os componentes do sistema, permitindo uma integração eficiente e em tempo real das diferentes partes envolvidas no processo de lingotamento contínuo.

Mensagem - Recebimento e Envio de Mensagem (JMS)

O código possuí a presença de entidades relacionadas ao JMS (Java Message Service) como:

- ***com.arcelormittal.framework.core.entity.JmsMessage***
- ***com.arcelormittal.framework.core.entity.JmsModel***

O JMS é uma API Java que permite que as aplicações enviem e recebam mensagens assíncronas de forma confiável, permitindo a comunicação entre diferentes partes do sistema de forma assíncrona. Essas mensagens são enviadas para filas ou tópicos (destinos JMS) e podem ser processadas por consumidores (receptores) assincronamente.

Fluxo de mensagens do JMS no projeto

A estrutura e o fluxo de mensagens do JMS no projeto para enviar e/ou receber mensagens. Vamos analisar os detalhes exatos sobre como isso é implementado:

Configuração do JMS:

A implementação do JMS é fornecida pelo framework utilizado, pois o projeto possui classes específicas relacionadas ao JMS.

Figura 8 - SendCD – ENVIO DA MENSAGEM CD PELA OMQ DA MLC1

Enviando Mensagens JMS:

No sistema MLC1, o envio de mensagem é configurado através da utilização de JMS (Java Message Service). O trecho de código relevante é a classe SendCD.

A classe SendCD é responsável por enviar mensagens JMS. Ela estende a classe AbstractSendCD<MensagemCDTemplate> e implementa a interface ISendCD. O método send nesta classe é utilizado para enviar uma mensagem CD (Cast Sequence Data) usando o JMS. A mensagem é preenchida usando o objeto MensagemCDTemplate, que é uma classe específica para definir o formato da mensagem CD. A mensagem é enviada utilizando o recurso sendMessageResource.

Para enviar uma mensagem, são necessários os seguintes passos:

- A classe SendCD implementa a interface ISendCD, que define os métodos para envio de mensagem.
- É feita a injeção de uma instância de ISendMessageResource na classe SendCD através da anotação @Inject.
- No método *send*, uma instância de MensagemCDTemplate é preenchida com os dados da sequência e da máquina de lingotamento, e em seguida, é enviada utilizando o método sendMessageResource.send(mensagem).

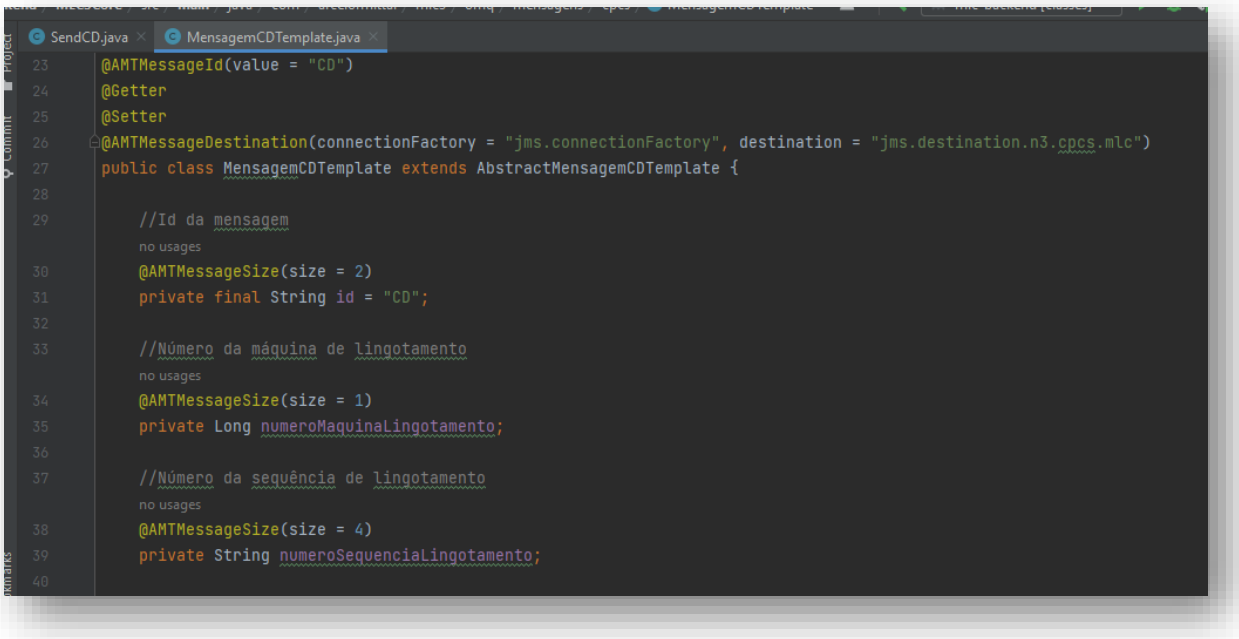


Figura 9 - TEMPLATE CD – FORMATO DA MENSAGEM CD PELA OMQ DA MLC1

Classe MensagemCDTemplate:

A classe MensagemCDTemplate é uma classe de modelo (template) que define o formato da mensagem CD - Cast Sequence Data. Ela possui diversos campos que representam os atributos da mensagem. Alguns desses campos têm anotações, como @AMTMessageSize, @AMTMessageDateFormat, @AMTMessagePow, etc., que podem ser usadas para definir tamanhos, formatos de data e operações de potência em campos numéricos para a serialização e desserialização da mensagem.

Interface ISendMessageResource:

Essa interface define o contrato para o envio de mensagens. Através desse contrato, a implementação do JMS (que é fornecida pelo framework) é abstraída, permitindo que o método send da classe SendCD utilize a interface para enviar mensagens sem se preocupar com detalhes de implementação específicos.

A configuração do envio de mensagens depende do sistema de mensageria utilizado. Existem várias tecnologias e protocolos disponíveis, como MQ (Message Queue), AMQP (Advanced Message Queuing Protocol), *Kafka*, entre outros, em nosso sistema MLC1, utilizamos o JMS, como descrito na sessão JMS

Pontuações genéricas sobre o assunto:

Para configurar o envio de mensagens, geralmente são necessários os seguintes passos:

- Definir uma conexão com o sistema de mensageria, especificando os detalhes de configuração, como endereço do servidor, portas, credenciais, etc.
- Criar uma fila ou tópico de destino para a mensagem.
- Configurar o produtor de mensagens para enviar a mensagem para a fila ou tópico adequado, fornecendo os dados necessários para a mensagem.
- Enviar a mensagem por meio do produtor, que será responsável por entregá-la ao sistema de mensageria.

A forma de enviar a mensagem pode variar dependendo da tecnologia utilizada. Pode-se utilizar APIs específicas fornecidas pelo sistema de mensageria ou bibliotecas de terceiros que facilitem o envio de mensagens.

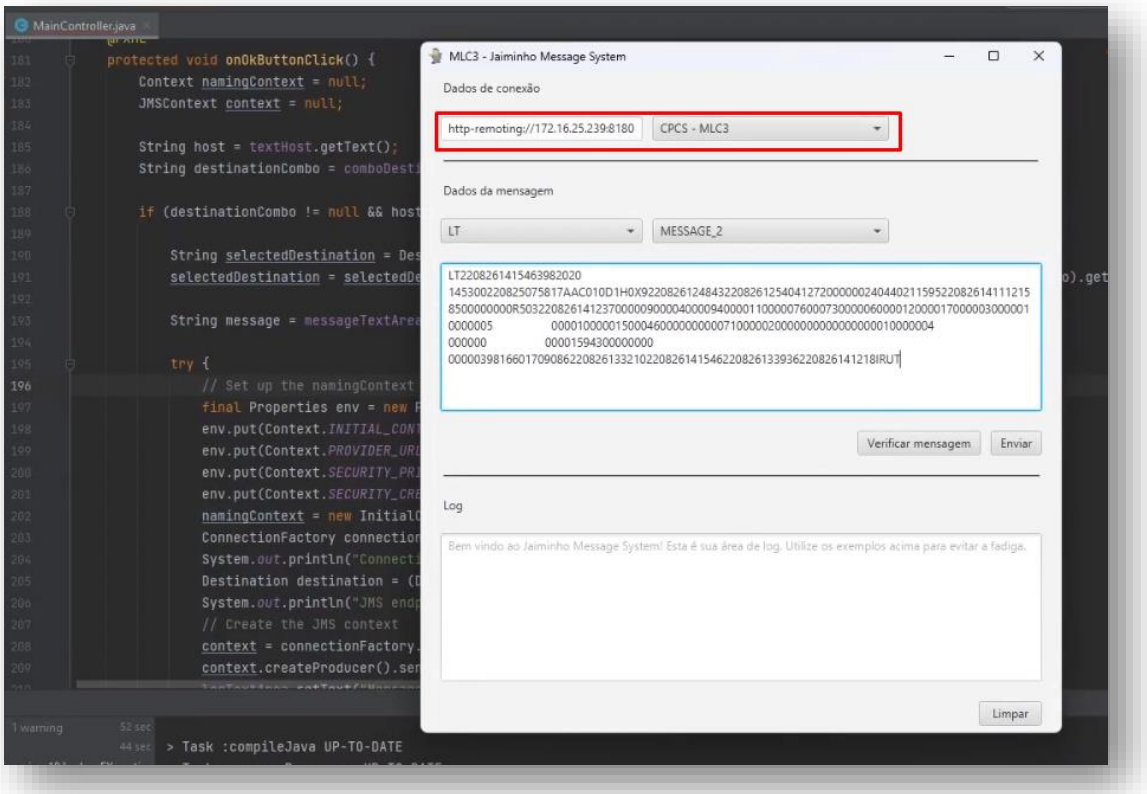


Figura 10 - "Jaiminho" ENVIANDO MENSAGEM LT

Testando envios/recebimentos de mensagens na MLC1

Para testar utilizando OPC / *KEPSERVER*, utilizaremos o *Prosys*, que por sua vez, se conecta a um servidor *Kepserver*, onde escrevemos nas TAGs, a aplicação que estará rodando, por intermédio das anotações, ex. @Tag e @Observable, detectam se houve mudança e executam o método chamado para cada caso.

Veja mais no Item 3.2 – Comunicação entre Níveis.

Utilizamos o "Jaiminho", API, desenvolvida para envios de mensagens para o N2, no ambiente de testes e desenvolvimento da contratada, em seu código, basicamente, se conecta no *WildFly*, fazendo uma autenticação, contendo Usuário e Senha. Em sua aplicação, definimos para qual fila e qual mensagem será enviada, fazendo uma comparação de formato e tamanho, tamanho e formato de cada mensagem é definido em seu *Template*, tratado no item 1.1.1, sendo o tamanho divergente, é impeditivo para o prosseguimento do teste, caso passe pelo tamanho, o próximo tratamento será na Classe de recebimento da mensagem, em nosso exemplo, RecebimentoLTDomainMLC1, que é um *Listner* contendo *Observable*, que em caso de alguma alteração, informa ao sistema.

Anotações e Configurações:

As anotações como @AMTMessageProduces, @AMTMessageDestination, @AMTMessageSize, etc., são utilizadas para configurar e mapear a estrutura da mensagem JMS, definindo seu formato e destino.

Implementação de JMS:

A classe ISendMessageResource declara os métodos *send*, *resend*, mas a lógica de envio efetiva está encapsulada em uma implementação concreta fornecida pelo framework.

Em resumo, o projeto utiliza o JMS para enviar mensagens do tipo CD (Cast Sequence Data). A classe SendCD é responsável por preencher o conteúdo da mensagem usando o objeto MensagemCDTemplate e enviá-la através do recurso sendMessageResource. A estrutura específica de configuração do JMS, como definição de filas/tópicos, está configurada pelo framework utilizado.

1.1.2 Gradle e Publicação

O sistema MLC1 utiliza o sistema de *build* Gradle para gerenciar suas dependências e realizar a compilação do código. Ele possui um arquivo de configuração chamado "**build.gradle**" que define as tarefas de build e publicação.

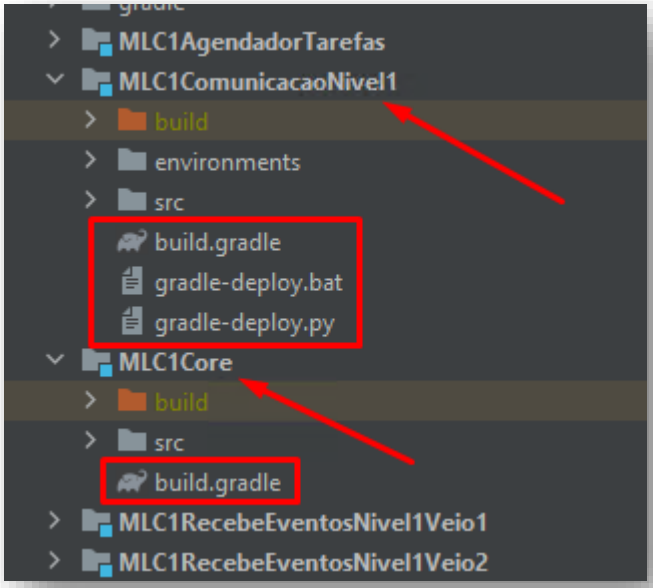


Figura 11 - ONDE ENCONTRAR OS ARQUIVOS GRADLE

O arquivo "**build.gradle**" é utilizado no projeto MLC1 para automatizar e gerenciar o processo de construção (*build*) do sistema. Ele contém configurações e dependências necessárias para compilar, testar e empacotar o código-fonte do projeto. No caso específico do MLC1, este arquivo define tarefas como a criação de documentação JavaDoc a partir do código-fonte, atualização da versão do sistema, publicação em um servidor *WebSphere*, entre outras.

Por exemplo, há a tarefa "publish*WebSphere*" que é responsável pela publicação do sistema no servidor de aplicação *WebSphere* do ambiente da Arcelor. Ela é executada em conjunto com outras tarefas específicas de cada projeto relacionado ao MLC1. Essas tarefas automatizadas facilitam o desenvolvimento e a implantação do sistema, garantindo uma execução consistente e controlada das etapas de construção do MLC1.

Implantando o sistema *back-end*

A implantação do sistema envolve a instalação e configuração de todos os componentes necessários para que o sistema funcione corretamente em um ambiente de produção.

Para implantar o sistema, é necessário seguir alguns passos:

- Preparar o ambiente de hospedagem, configurando os servidores, redes e infraestrutura necessários.
- Realizar a instalação dos componentes do sistema, como servidores de aplicação, bancos de dados, caches, etc.
- Configurar as variáveis de ambiente e as configurações do sistema para refletir o ambiente de produção.

- Realizar o *deploy* do código-fonte do sistema nos servidores de aplicação.
- Executar scripts de inicialização e configuração para preparar o sistema para a operação.
- Realizar testes de integração e verificação para garantir que o sistema foi implantado corretamente e está funcionando adequadamente.

O trecho do arquivo build.gradle mostra algumas configurações relacionadas à construção do projeto.

No *WebSphere*, é feita através de uma interface gráfica, a partir do arquivo gerado ".war" , e faz se o Deploy.

Pelo *WildFly*, pode-se também fazer a partir da interface gráfica, manualmente, em seu Management, por exemplo, acessando via browser pelo endereço `172.16.25.239:9990/console/index.html#deployments` onde é possível fazer o Deploy e Undeploy por ele.

Também é possível fazermos utilizando o arquivo ".war" gerado na pasta deployments, dentro do standalone do *WildFly*, mas, em via de regra, utilizamos o Pipeline, da Azure, configurada para nossos ambientes, essa Pipeline é previamente configurada, contendo o passo a passo do que deve ser feito para que a aplicação "suba", seja feito o seu Deploy. Simplificando, o Pipe faz o Checkout, em seguida busca as alterações que subiram pelo Git através da Branch "DEV-SUPERO", faz a Build nos servidores, uma vez a Build passando, é a vez do comando Publish, responsável em "publicar" a aplicação, onde é lançado o arquivo ".war" para deployments.

1.2 *Front-end*

A arquitetura do font-end do MLC1 é dividida em duas partes, sendo elas:

- **Shell**: Onde se encontra a aplicação Framework, que é responsável pela autenticação dos usuários.
- **MLC**: Onde se encontram as telas da aplicação, os modelos de objetos, popups e os serviços que fazem a comunicação com o *back-end*.

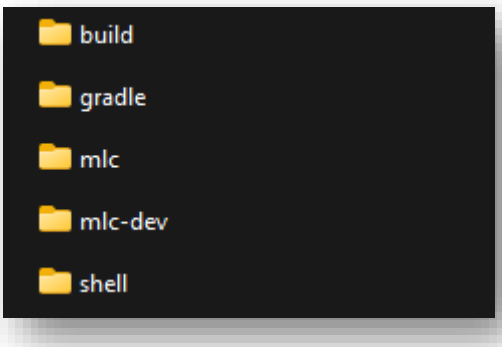


Figura 12 - ORGANIZAÇÃO DO FRONT-END NO DIRETÓRIO DE CÓDIGO-FONTE

2 Diagramas de Pacotes

O projeto como um todo possui uma hierarquia de pacotes bastante complexa, de modo que nem todos se enquadrem no escopo deste manual, pois são apenas auxiliares. Os diagramas mostrados nesta seção levam em conta apenas os pacotes mais relevantes para manutenção das funcionalidades do sistema.

2.1 *Back-end*

Abaixo segue a estrutura da biblioteca MLCCore:

Figura 13 - DIAGRAMA DE PACOTES DO BACK-END DO SISTEMA MLC1

Segue uma breve descrição do conteúdo desses pacotes:

- **controllers**: neste pacote ficam os serviços responsáveis por receber requisições da tela, processá-las e devolver a resposta;
- **models**: contém a modelagem das entidades da aplicação;
- **repositories**: define métodos de abstração de consultas ao banco de dados;

- **resources**: pacote que oferece recursos de persistência, comunicação, entre outros;
- **dto**: contém modelos de dados para tráfego síncrono entre sistemas;
- **interfaces**: contém abstrações de funcionalidades do sistema relacionadas ao tráfego de dados;
- **enums**: pacote contendo classes de enumeração de informações contidas em um conjunto finito e invariável (por exemplo, o status de placa contido no conjunto {“Boa”, Sucata”, “Suspensa” e “Desclassificada”}). Definido apenas no MLC1Core e acessível por todos os sistemas;
- **messages**:
    - **externo ao pacote “resources”**: apresenta modelos de mensagens JMS e OMQ para comunicação entre MLC1 e sistemas em outros níveis, como o N1 e N3;
    - **interno ao pacote “resources”**: contém classes preparadas para observar a chegada de mensagens de entrada e invocar o respectivo método de tratamento.

2.2 *Front-end*

Assim como o *back-end*, o *front-end* também apresenta uma estrutura organizada de forma diferente.

As imagens a seguir apresentam o diagrama de pacotes para os projetos Shell e MLC.

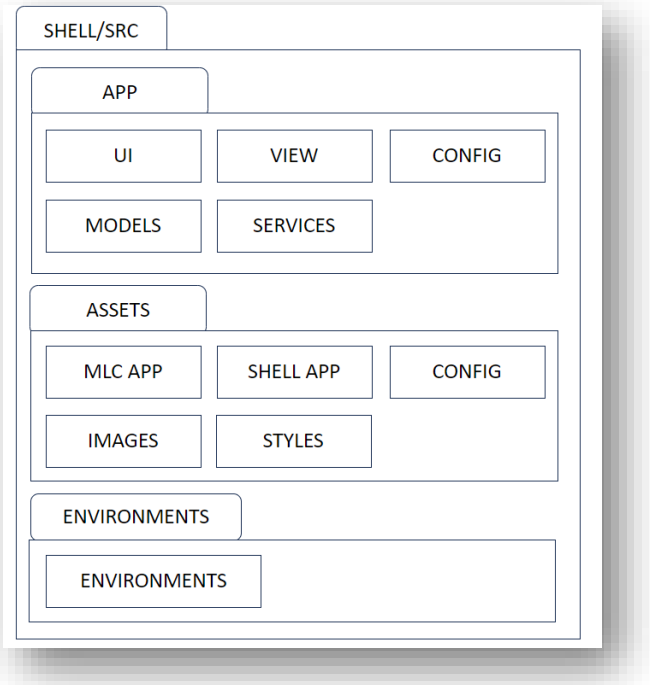


Figura 14 - DIAGRAMA DE PACOTES DO PROJETO SHELL

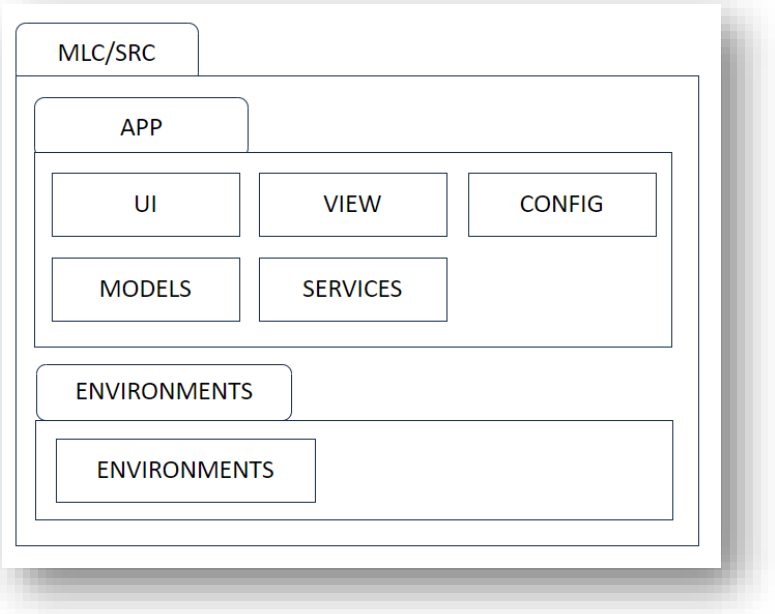


Figura 15 - DIAGRAMA DE PACOTES DO PROJETO MLC

A seguir uma breve descrição de cada um dos pacotes:

- **UI**: Pacote que contém componentes de elementos de layout do sistema, como componentes de busca ou atalhos rápidos.
- **VIEW**: pacote contendo os componentes das telas do sistema.
- **MODELS**: Pacote que contém os modelos de objetos e enumerados.
- **SERVICES**: Pacote que contém os serviços utilizados para comunicação com o *back-end* ou de auxílio.
- **ENVIRONMENTS SHELL**: Pacote que contém as informações dos Environments relativos a cada MLC em cada um dos seus contextos de execução.
- **ENVIRONMENTS MLC**: Pacote que contém a estrutura básica dos Environments, que é carregado com as informações do contexto atual da MLC selecionada.

3 Conhecendo o Sistema

Demandas presentes e futuras podem exigir que novos recursos sejam adicionados ao sistema, incluindo desde novas entidades de dados até novas telas com funcionalidades em particular. Além disso, pode ser necessário modificar partes já existentes do projeto, a fim de adaptá-las às novidades, eliminar conteúdo desnecessário ou implementar novos parâmetros e funcionalidades. Esta seção é destinada ao conhecimento sobre as formas mais importantes de se criar ou modificar recursos dos sistemas.

3.1 Visão Geral

O diagrama mostrado na abaixo esquematiza o fluxo da troca de informações entre *front-end* e *back-end*. Conhecê-lo é importante para que se possa rastear os possíveis pontos de manutenção do sistema em análise.

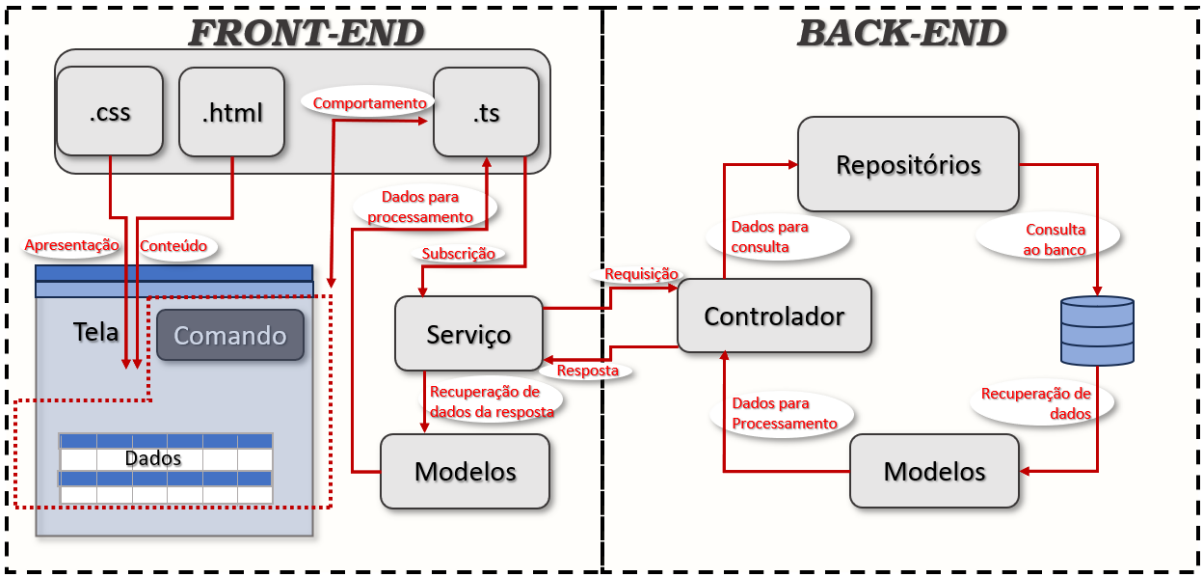


Figura 16 - DIAGRAMA DE COMUNICAÇÃO ENTRE FRONT-END E BACK-END

3.1.1 Telas

Conforme sugere a figura, os componentes de tela do *front-end* são uma composição de um arquivo CSS, um arquivo HTML e um arquivo TypeScript. A parte visual e usável da tela é estabelecida pelo arquivo CSS, que define parâmetros de apresentação como dimensões e cores, e pelo arquivo HTML, que confere conteúdo à tela em forma de botões, tabelas, ferramentas de pesquisa, labels, etc., bem como define os valores/comandos a serem recuperados/executados por esses elementos.

As imagens abaixo mostram exemplos de arquivos CSS e HTML e a tela resultante (Tela Situação de Máquina).

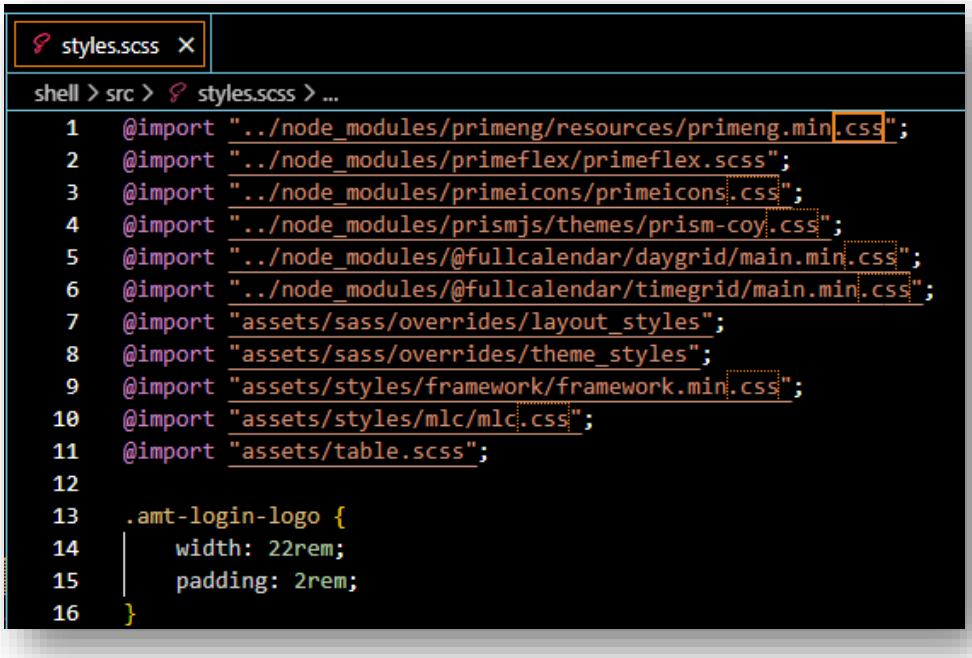


Figura 17 - EXEMPLO DE ARQUIVO CSS DE UM COMPONENTE DE TELA

Figura 18 - EXEMPLO DE ARQUIVO HTML DE UM COMPONENTE DE TELA

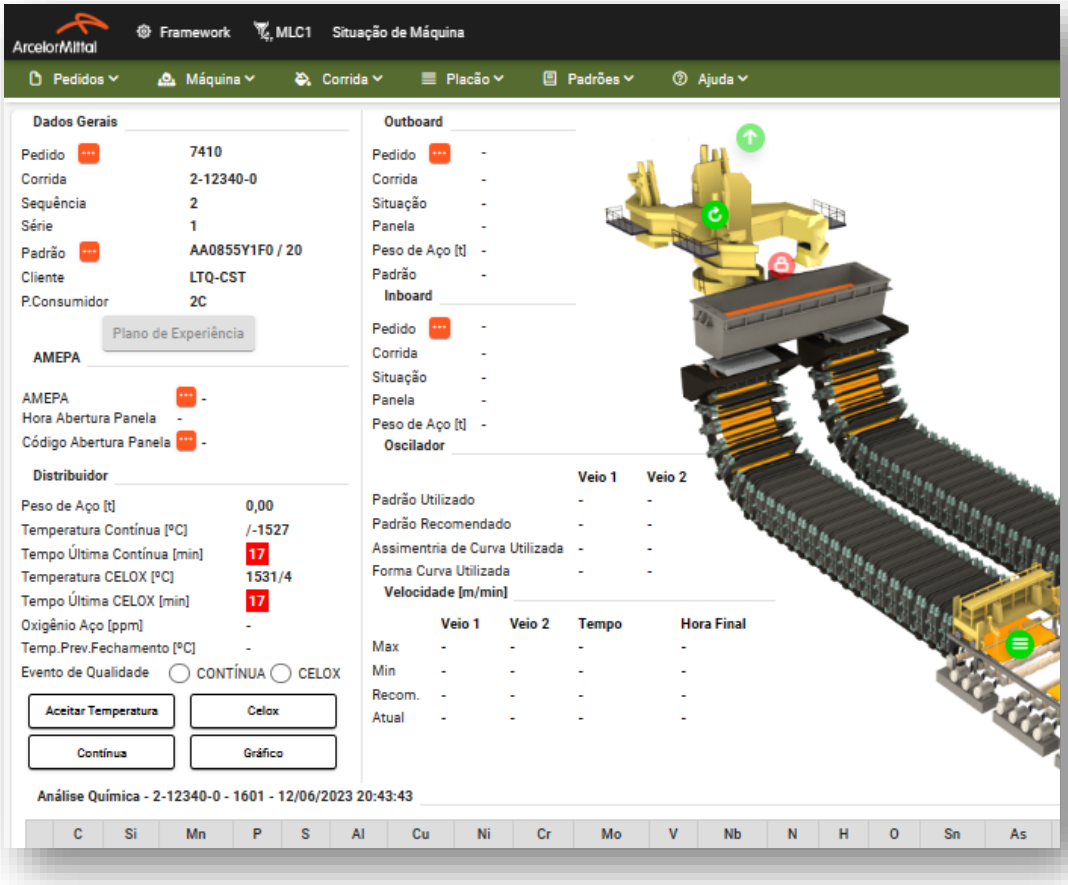


Figura 19 - TELA CONSTRUÍDA PELOS CÓDIGOS CSS E HTML EXEMPLIFICADOS

Quando ocorre uma solicitação de dados ou comandos por meio da interface, os métodos relacionados no arquivo TypeScript entram em ação. Esses métodos são responsáveis por definir o comportamento do sistema em resposta à solicitação. É nesse ponto que entra em jogo o conceito de serviços.

Os serviços desempenham um papel fundamental no processamento das solicitações feitas pela interface. Eles são responsáveis por executar a lógica necessária para atender às solicitações e retornar os dados apropriados. Os serviços podem envolver comunicação com o *back-end*, realização de cálculos, manipulação de dados e outras tarefas relacionadas.

Ao utilizar serviços, é possível separar a lógica de negócios da interface do usuário, promovendo uma arquitetura mais modular e reutilizável. Isso facilita a manutenção, o teste e a escalabilidade do sistema.

Portanto, os serviços desempenham um papel importante ao definir o comportamento do sistema em resposta às solicitações feitas pela interface, proporcionando uma estrutura organizada e eficiente para lidar com as operações necessárias.

3.1.2 Serviços

Quando se faz necessária a comunicação com o *back-end*, os métodos TypeScript invocam um serviço de comunicação através da subscrição, fornecendo os dados relevantes. Esse serviço, então, envia uma requisição REST através de um caminho definido, passando todos os parâmetros necessários.

Note, no código HTML da imagem abaixo, que a tela está configurada para invocar automaticamente no metodo ngOnInit() o método getGradesRecentes(). A imagem exibe a definição do método no arquivo TypeScript até o ponto em que ele subscreve o serviço para solicitar os dados dos Grades.

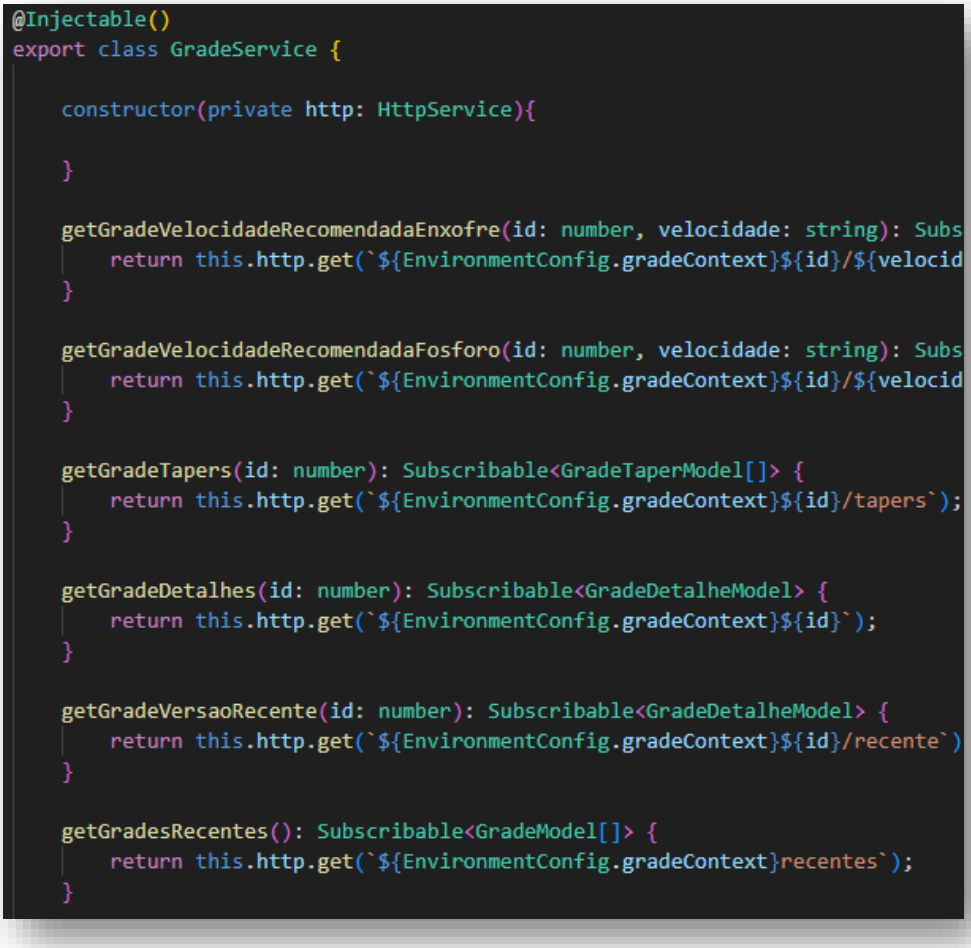
```
ngOnInit(): void {
    this.pageTitleService.setPageTitle('Conversão Quente Frio');

    this.loadClientes();
    this.loadGrades();
    this.loadData();
}

loadClientes() {
    this.clientesService
        .getAll()
        .pipe(
            map((data) => {
                return data.sort((a, b) =>
                    a.nome > b.nome ? 1 : a.nome < b.nome ? -1 : 0
                );
            })
        )
        .subscribe((data) => {
            this.clientes = data;
        });
}

loadGrades() {
    this.gradesService.getGradesRecentes().subscribe((data) => {
        this.padroes = data;
    });
}
```

Figura 20 - MÉTODO TYPESCRIPT COM SUBSCRIÇÃO DE SERVIÇO



```
@Injectable()
export class GradeService {

    constructor(private http: HttpService){

    }

    getGradeVelocidadeRecomendadaEnxofre(id: number, velocidade: string): Subs
        return this.http.get(`${EnvironmentConfig.gradeContext}${id}/${velocid
    }

    getGradeVelocidadeRecomendadaFosforo(id: number, velocidade: string): Subs
        return this.http.get(`${EnvironmentConfig.gradeContext}${id}/${velocid
    }

    getGradeTapers(id: number): Subscribable<GradeTaperModel[]> {
        return this.http.get(`${EnvironmentConfig.gradeContext}${id}/tapers`);
    }

    getGradeDetalhes(id: number): Subscribable<GradeDetalheModel> {
        return this.http.get(`${EnvironmentConfig.gradeContext}${id}`);
    }

    getGradeVersaoRecente(id: number): Subscribable<GradeDetalheModel> {
        return this.http.get(`${EnvironmentConfig.gradeContext}${id}/recente`)
    }

    getGradesRecentes(): Subscribable<GradeModel[]> {
        return this.http.get(`${EnvironmentConfig.gradeContext}recentes`);
    }
```

Figura 21 - SERVIÇO COM REQUISIÇÃO WEB SERVICE PARA COMUNICAÇÃO ATRAVÉS DE UM CAMINHO DEFINIDO

3.1.3 Controladores

No *back-end*, os controladores são classes Java responsáveis por receber e processar as requisições vindas do *front-end*, solicitando dados do banco de dados e fornecendo respostas adequadas. Um exemplo de controlador, ilustrado na Figura 15, mostra a injeção de dependência JPA e o serviço associado.

Esses controladores recebem a requisição e iniciam o processo de processamento necessário. Geralmente, é necessário recuperar dados do banco de dados, e é nesse momento que os controladores injetam dependências das JPA Domain Stores. Por meio dessas dependências, eles têm acesso aos métodos das classes de repositórios, que permitem a realização de consultas e manipulações nos dados do banco.

Essa arquitetura permite uma separação clara de responsabilidades, em que os controladores se concentram na lógica de processamento das requisições e os repositórios fornecem métodos para interagir com o banco de dados. Essa abordagem facilita a manutenção, o teste e a escalabilidade do sistema, garantindo uma organização eficiente do código.

Portanto, os controladores Java no *back-end* desempenham um papel fundamental no processamento das requisições do *front-end*, interagindo com os repositórios para acessar e manipular os dados do banco de dados.

Criando um serviço “end-point”

Criamos uma classe service onde ela começa com uma anotação em cima do nome @Path(caminho/) com o caminho do end point. A criação de um serviço (end-point) geralmente envolve a definição de uma rota ou URL específica que o sistema deve responder. Essa rota é associada a um controlador ou classe responsável por lidar com as solicitações recebidas nesse end-point.

Depois disso criamos os métodos com as anotações que vão definir o tipo de requisições, revisando, para criar um serviço, é necessário seguir alguns passos:

- Definir a rota ou URL do serviço, especificando o caminho e os parâmetros, se aplicável.
- Criar um controlador ou classe que tratará as solicitações recebidas nesse end-point.
- Implementar a lógica necessária dentro do controlador para processar a solicitação e retornar a resposta adequada.
- Configurar o sistema para direcionar as solicitações feitas para o end-point correto para o controlador associado.

Exemplo:

Vamos ter um método abrir panela, onde queremos receber informações referentes a panela, então o tipo da requisitarão *HTPP* será *GET*, para isso definimos um anotação @GET em cima do método, abaixo @*Path*(/abrirpanela), o tipo da mensagem que vamos receber vai ser um *JSON*, poderia ser também um JMS Text, então definimos uma outra anotação @*Produces*(MediaType.APPLICATION_JSON) e por último a permissão sobre essa requisição que definimos com o @*Permissions*(moduleAllowed = true).

3.1.4 Repositórios

A classe "AnaliseAcoRepository" é um exemplo de um repositório no contexto do sistema apresentado. Essa classe Java é responsável por abstrair as consultas ao banco de dados relacionadas às análises de aço, conforme a imagem abaixo:

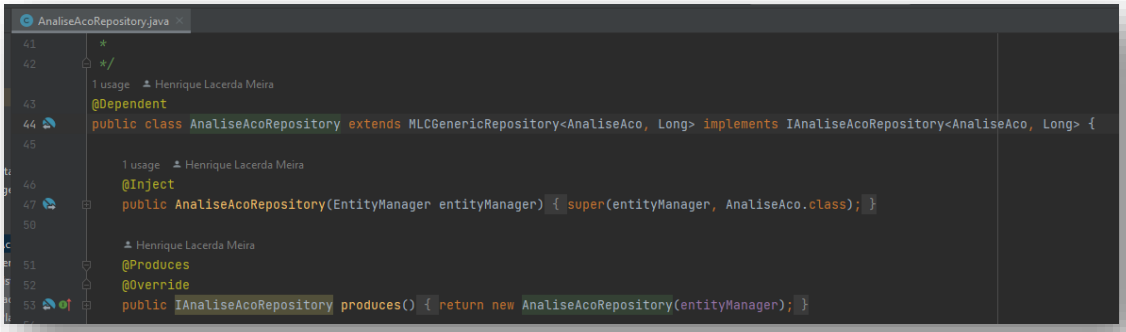


Figura 22 - MÉTODO DEFINIDO EM CLASSE REPOSITÓRIO PARA CONSULTA AO BANCO DE DADOS

Ao herdar a classe "MLCGenericRepository", esse repositório implementa a interface "IAnaliseAcoRepository" e define o tipo de entidade (AnaliseAco) e o tipo da chave primária (Long). A injeção de dependência do "EntityManager" é feita pelo construtor.

3.1.5 Entidades de Dados

Quando os modelos estão relacionados a tabelas do banco de dados, eles herdam modelos existentes no MLC1Core que são configurados com anotações Java para representar as tabelas correspondentes no banco. Em seguida, o controlador prossegue com a recuperação e processamento dos dados até chegar ao momento de enviar a resposta final para o *front-end*. Durante esse processo, os controladores utilizam os modelos de entidades para manipular os dados, realizar consultas e executar as operações necessárias para atender às requisições do *front-end*.

A classe "ConversaoFrioQuente" é uma entidade que representa as conversões de frio para quente no contexto do sistema. Essa classe está configurada por meio de anotações Java para mapear a tabela correspondente no banco de dados.

A classe estende a classe base "MLCGenericRepository" e inclui as anotações necessárias, como "@Entity" e "@Table", para indicar que é uma entidade persistente e especificar o nome da tabela no banco de dados.

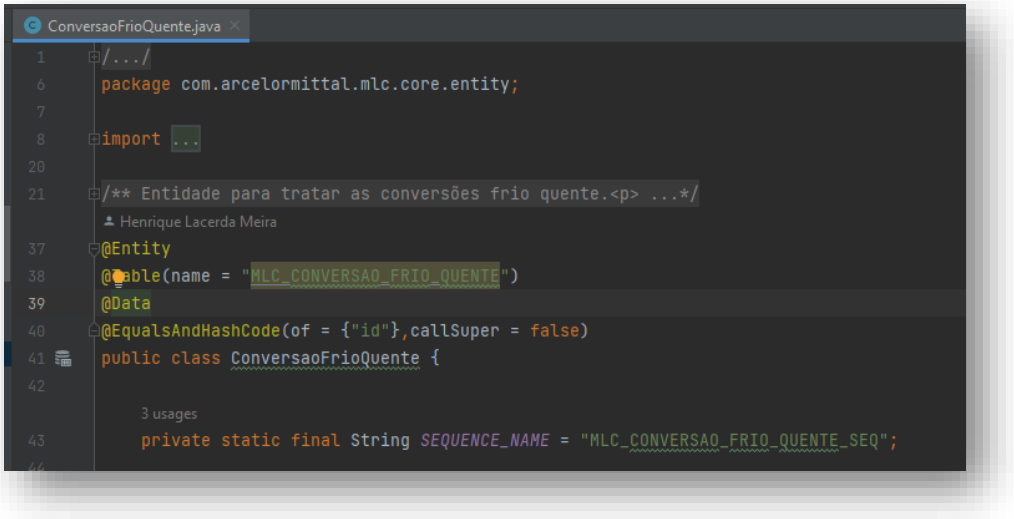


Figura 23 - MODELO DE ENTIDADE DO BACK-END

Criando uma entidade

A criação de uma entidade geralmente envolve a definição de uma classe que representa um objeto de negócio no sistema. Essa classe contém os atributos e métodos relacionados a essa entidade, permitindo manipular e armazenar os dados correspondentes.

Para criar uma entidade, é necessário seguir alguns passos:

- Definir os atributos da entidade, como nome, idade, endereço, etc.
- Implementar os métodos necessários para acessar e modificar os atributos da entidade.
- Estabelecer as associações e relacionamentos entre as entidades, se necessário, utilizando anotações ou configurações específicas do framework utilizado.
- Configurar a persistência da entidade, se aplicável, definindo as anotações ou configurações adequadas para mapeá-la em uma tabela ou documento no banco de dados.

A entidade representa a tabela no banco de dados, exemplo:

Tabela MLC_PEDIDOS criamos uma classe dentro do projeto MLCCORE na pasta entity e essa classe vai se chamar Pedidos.

Para definir essa classe como entidade definimos uma anotação no começo dela como *@Entity*, *@Table*(name ="MLC_PEDIDOS") dentro dessa anotação passamos o nome da tabela no banco de dados para mapear a aquela entidade com o banco. Nesta classe os atributos representam cada coluna da tabela.

3.1.6 Processamento da resposta

No *front-end*, os dados recebidos do *back-end* são convertidos no Model, no exemplo abaixo, a resposta do *back-end* é convertida em um Array (Lista) do tipo GradeModel.

```
export class GradeModel {
    constructor(){}
    id: number;
    versaoId: number;
    codigo: string;
    familia: number;
    cliente: string;
    data: Date;
    comentario: string;
    instrucoesEq1: string;
    instrucoesEq2: string;
    instrucoesEq3: string;
    instrucoesEq4: string;
}
```

Figura 24 - EXEMPLO DE MODEL

3.2 Comunicação Nível1 (PLC – COMMON, Instrumentation, RUNOUT e STRAND) / Nível2 (MLC1) / Nível 3 (Sistema Gerencial)

No nível 1 da pirâmide de automação industrial, encontram-se os dispositivos de campo, como atuadores, sensores e transmissores, responsáveis pela aquisição de dados e controle manual. Já no nível 2, temos os equipamentos que realizam o controle automatizado das atividades da planta, como CLPs (Controladores Lógicos Programáveis), SDCDs (Sistemas Digitais de Controle Distribuído) e relés.

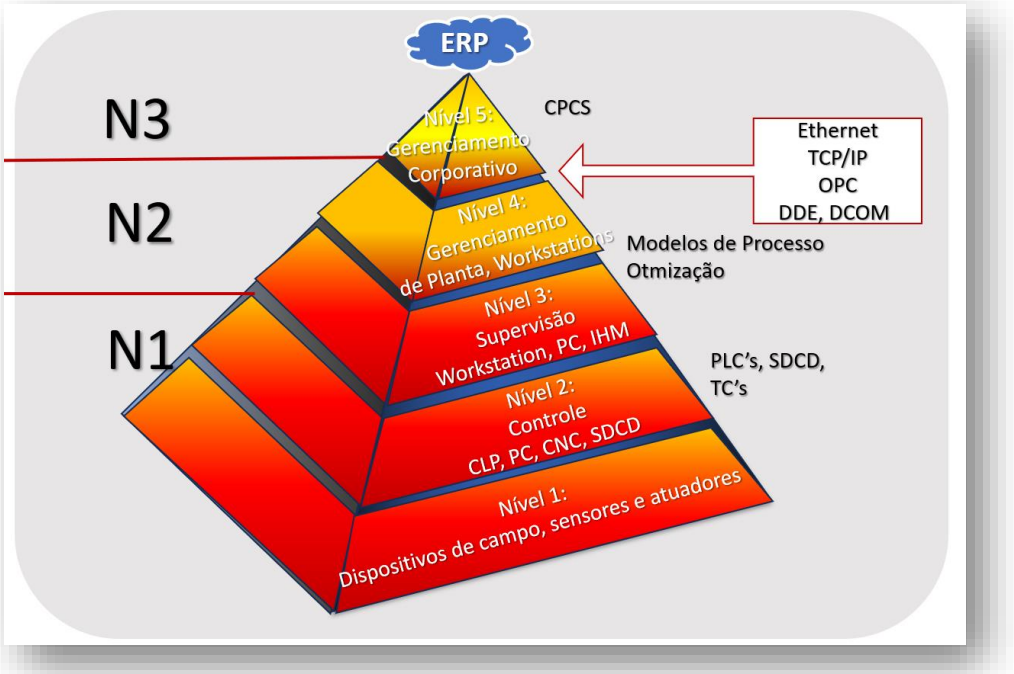


Figura 25 - PIRÂMIDE DE AUTOMAÇÃO INDUSTRIAL DA ARCELORMITTAL

No contexto da comunicação entre os níveis 1, 2 e 3

A comunicação entre o N1, PLCs (ou CLPs) e nosso sistema MLC1(N2) é feita através das TAGs, utilizadas pelo OPC / *Kepserver* – Em nosso ambiente de desenvolvimento utilizamos o Prosys OPC UA Browser, que conectasse a um *Kepserver*, onde estão as especificações das TAGs, onde podemos adicionar, remover ou alterá-las a partir deste *Kepserver*EX. A configuração de qual servidor *Kepserver* a plicação estará usando, é feita através das informações em uma tabela, de nosso Banco de Dados, a saber, FWK_SERVIDORES, contendo o endereço de qual servidor OPC estará sendo utilizado.

Kepserver, conhecido como Kepware (anteriormente conhecido como *Kepserver*EX) é um software da PTC que atua como um servidor de comunicação industrial e é amplamente usado na automação industrial e em sistemas de controle. Ele é um servidor OPC (OLE for Process Control), que é um padrão amplamente adotado na indústria para facilitar a comunicação e a troca de dados entre dispositivos e sistemas de diferentes fabricantes em um ambiente industrial.

Em resumo, o *Kepserver* OPC é uma plataforma que permite conectar dispositivos industriais, como controladores de automação, sensores, medidores e outros equipamentos, a sistemas de supervisão e controle (SCADA - Supervisory Control and Data Acquisition), bem como a outros aplicativos que necessitam acessar e interagir com os dados de campo. Ele facilita a troca de informações entre diferentes dispositivos e aplicativos, permitindo que os dados sejam compartilhados de forma eficiente e confiável, facilitando a automação e a monitorização de processos industriais.

Enquanto que para a comunicação entre os sistemas de nível 2, como a MLC1, MLC2 e os sistema de nível 3 N3, utilizam filas de mensagens JMS para se comunicarem entre si. Além disso, eles

também se comunicam com o sistema legado por meio do protocolo OMQ. Essas mensagens são tratadas como modelos Java, onde cada atributo representa uma informação específica da mensagem. Cada modelo é nomeado de acordo com o identificador da mensagem, como ilustrado na imagem abaixo:

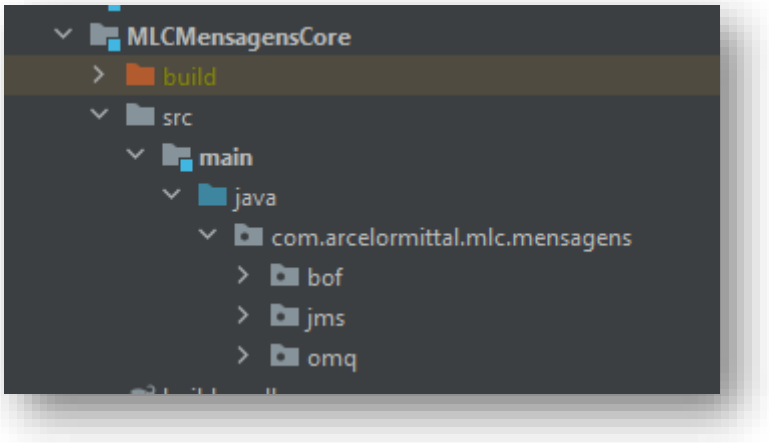


Figura 26 - PACOTES DE MODELOS DE MENSAGENS JMS E OMQ DOS SISTEMAS DOS MLC1

Portanto, a comunicação entre os diferentes níveis da automação industrial é realizada por meio de filas de mensagens JMS e OMQ, permitindo o intercâmbio de informações e comandos entre os dispositivos de campo, os sistemas da MLC1 e o sistema gerencial N3. Essa comunicação é essencial para o funcionamento integrado e coordenado de todos os componentes da planta industrial.

As mensagens enviadas e recebidas são tratadas como modelos Java, onde cada atributo guarda uma informação da mensagem. Cada modelo é nomeado de acordo com o identificador da mensagem.

Cada sistema possui mensagens de entrada e mensagens de saída. A saída de mensagens pode ocorrer em qualquer parte do código *back-end*, onde seja necessário. Essa saída é realizada por meio do recurso SendMessage, geralmente injetado pelos controladores, mas na MLC1, utilizamos classes de envio exclusivas para cada mensagem, conforme exemplificado na imagem abaixo:

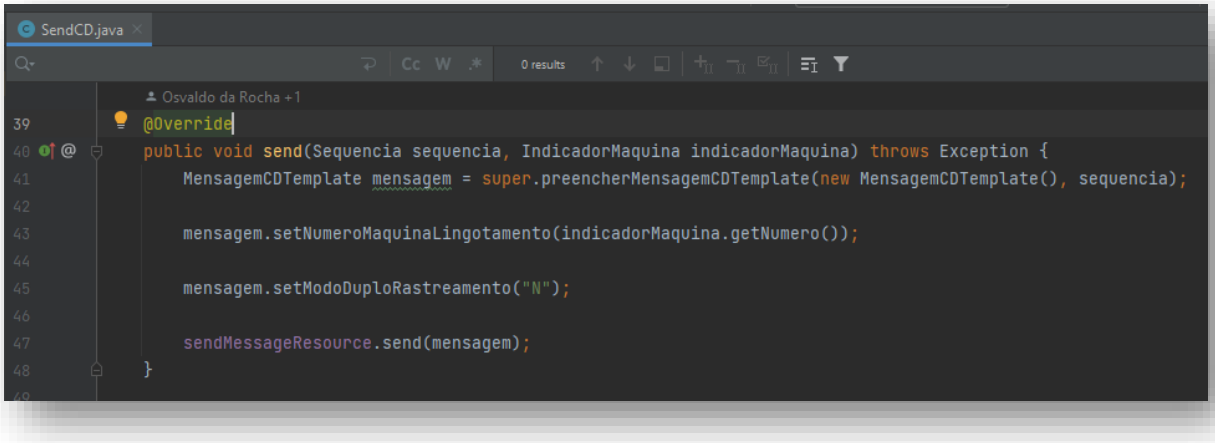


Figura 27 - INSTANCIANDO E ENVIANDO UM MODELO DE MENSAGEM JMS

A recepção de mensagens de entrada pode ocorrer a qualquer momento, e, portanto, o sistema deve estar sempre preparado para a leitura. Para isso, são definidas, para cada modelo, classes de recursos que contêm listeners para invocar um método de tratamento da mensagem lida. Essas classes de recursos são nomeadas com o identificador da mensagem seguido do termo "Resource", como mostrado na imagem abaixo:

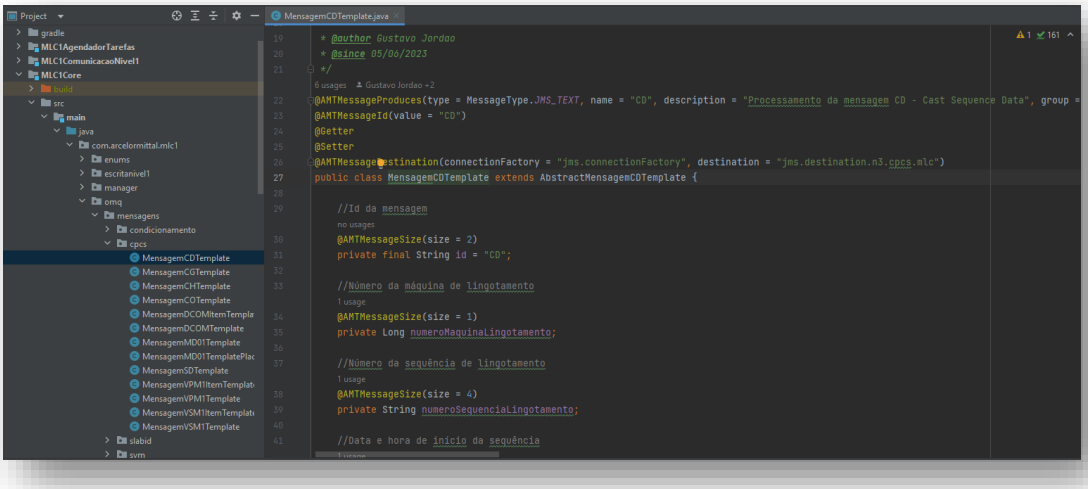
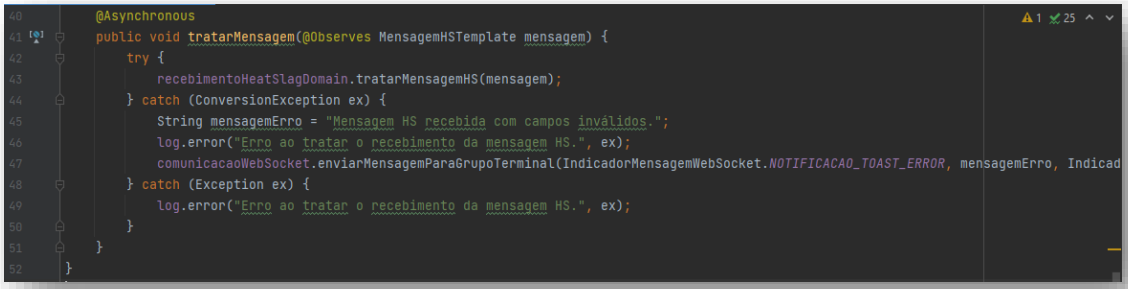


Figura 28 - PACOTES DE RECURSOS DE MENSAGENS JMS E OMQ DO SISTEMA MLC1

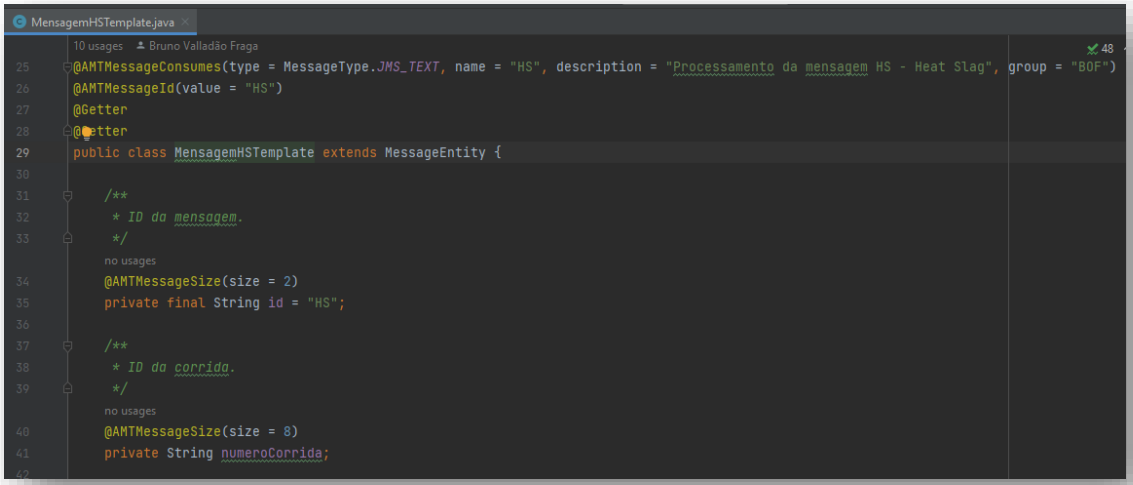
A imagem a seguir exibe um método contido em um destes recursos que observa a leitura de uma mensagem e inicia seu processamento.



```
@Asynchronous
public void tratarMensagem(@Observes MensagemHSTemplate mensagem) {
    try {
        recebimentoHeatSlagDomain.tratarMensagemHS(mensagem);
    } catch (ConversionException ex) {
        String mensagemErro = "Mensagem HS recebida com campos inválidos.";
        log.error("Erro ao tratar o recebimento da mensagem HS.", ex);
        comunicacaoWebSocket.enviarMensagemParaGrupoTerminal(IndicadorMensagemWebSocket.NOTIFICACAO_TOAST_ERROR, mensagemErro, Indicad
    } catch (Exception ex) {
        log.error("Erro ao tratar o recebimento da mensagem HS.", ex);
    }
}
}
```

Figura 29 - LISTENER DE MENSAGEM HS QUE DEFINE TRATAMENTO A SER EXECUTADO QUANDO A MENSAGEM É LIDA

Na imagem acima e na seguinte, nota-se que o tratamento está definido no próprio modelo:



MensagemHSTemplate.java

```
10 usages  Bruno Valladão Fraga
@AMTMessageConsumes(type = MessageType.JMS_TEXT, name = "HS", description = "Processamento da mensagem HS - Heat Slag", group = "BOF")
@AMTMessageId(value = "HS")
@Getter
@Setter
public class MensagemHSTemplate extends MessageEntity {

    /**
     * ID da mensagem.
     */
    no usages
    @AMTMessageSize(size = 2)
    private final String id = "HS";

    /**
     * ID da corrida.
     */
    no usages
    @AMTMessageSize(size = 8)
    private String numeroCorrida;
```

Figura 30 - MÉTODO DE TRATAMENTO DA MENSAGEM HS, CHAMADO DE TEMPLATE, CONTENDO A SEQUÊNCIA, TIPO E TAMANHO DOS DADOS

4 Detectando Problemas

O sistema da MLC1, diante de sua complexidade, pode ser acometido por problemas das mais diversas naturezas. Alguns deles têm relação clara com o *front-end*, como erros de visualização em telas; outros são diretamente relacionados ao *back-end*, como problemas de consulta ao banco de dados; outros, ainda, demandam maior investigação para que se descubra sua origem.

O sistema pode ser preparado para lidar com alguns problemas em potencial, tratando-os ou notificando sua ocorrência. Em outras situações, o sistema pode operar normalmente e fornecer uma resposta incorreta, de modo que o problema seja detectado apenas pelo usuário. As subseções

seguintes orientam sobre as principais formas de lidar com cada uma dessas situações.

Como detectar os erros

A detecção de erros geralmente é feita por meio de técnicas de monitoramento e registro de logs. O sistema MLC1 é configurado para registrar informações detalhadas sobre eventos, erros e exceções que ocorrem durante a execução.

Nos casos de *Back-end*, utilizamos, geralmente, entre outros, blocos de códigos, chamados Try/Catch, colocamos no início e fim de cada ação no código e o ERRO é "disparado" dentro do catch, quando ocorre um erro dentro do código. **Mais sobre o Log na sessão 4.1.1**.

Para criar esse log inserimos no começo das classes como dependência: protected final Logger log = LoggerFactory.getLogger(this.getClass());

Para detectar erros, é necessário:

- Configurar um mecanismo de logging para registrar as informações relevantes.
- Implementar tratamentos de erros adequados, como exceções, para capturar e lidar com problemas que ocorrem durante a execução.
- Analisar os logs gerados para identificar padrões de erros, exceções recorrentes ou comportamentos anômalos que indiquem problemas no sistema.
- Utilizar ferramentas de monitoramento em tempo real para detectar e alertar sobre erros em tempo hábil.

Os tipos de Logs do sistema

Os logs utilizados no sistema MLC1 desempenham um papel fundamental para o monitoramento e identificação de problemas no sistema, tanto no *front-end* quanto no *back-end*. Eles são registrados por meio de técnicas de *logging* e fornecem informações detalhadas sobre eventos, erros e exceções que ocorrem durante a execução do sistema.

No contexto do *back-end*, os logs são utilizados para acompanhar o fluxo das operações, permitindo que desenvolvedores e administradores possam analisar e entender o comportamento do sistema, além de detectar possíveis falhas e anomalias. Os logs geralmente são categorizados em níveis como DEBUG, INFO, WARN e ERROR, conforme a gravidade da mensagem registrada.

Os logs de nível DEBUG são usados para registrar informações detalhadas e auxiliam na depuração do sistema, fornecendo detalhes sobre variáveis, parâmetros e fluxos de execução. São úteis para identificar o caminho exato que o código percorre e verificar valores de variáveis e propriedades.

Os logs de nível WARN indicam eventos potencialmente problemáticos que não causam falha no sistema, mas podem merecer atenção e correção. Essas mensagens podem ser úteis para antecipar problemas futuros e agir preventivamente.

Por fim, o log de nível ERROR é de extrema importância, pois registra problemas e exceções que afetam o funcionamento adequado do sistema. Quando ocorre um erro no código, um bloco de código Try/Catch é acionado, e a mensagem de erro é registrada no log ERROR, indicando a causa do problema. É importante ressaltar que os logs de erro fornecem informações essenciais para identificar a origem de falhas e possibilitam que os desenvolvedores tomem as ações necessárias para corrigir e melhorar o sistema.

Os logs são armazenados em arquivos, como o server.log, localizado no diretório correspondente ao servidor de aplicação WildFly 26.1.1. O log de servidor system também pode ser utilizado para registrar informações gerais sobre o funcionamento do sistema e eventos importantes.

Para facilitar a detecção e análise de problemas, é fundamental que os logs sejam configurados adequadamente e contenham informações relevantes. Além disso, ferramentas de monitoramento em tempo real podem ser utilizadas para alertar sobre erros imediatamente após sua ocorrência, permitindo uma resposta ágil aos problemas.

A análise dos logs é uma prática essencial para manter a estabilidade e confiabilidade do sistema MLC1, permitindo a detecção precoce de problemas e contribuindo para a melhoria contínua do sistema. Com os registros detalhados fornecidos pelos logs, a equipe de desenvolvimento pode investigar e corrigir problemas de forma eficiente, garantindo a operação adequada do processo de lingotamento contínuo de aço.

4.1 Problemas reconhecidos pelo sistema

Quando o sistema é submetido a uma condição de erro, uma notificação é exibida na tela, conforme a imagem abaixo:

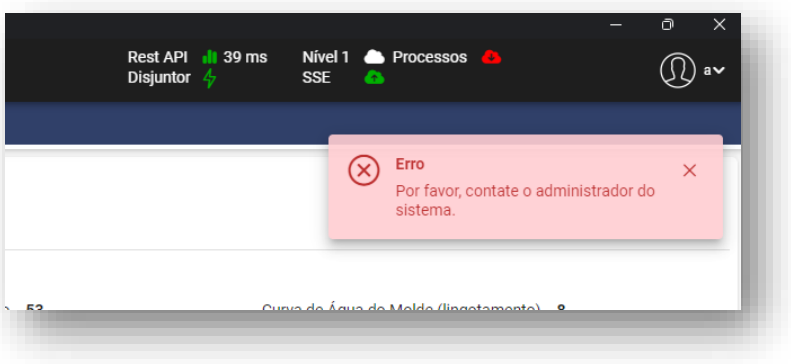


Figura 31 - EXEMPLO DE ERRO GENÉRICO NO SISTEMA DA MLC1

A imagem acima ilustra um caso de erro genérico causado por condições inesperadas, como demandar acesso a um atributo ou método de um objeto nulo. Sua origem pode estar ligada tanto ao *front-end* quanto ao *back-end*. Em outros casos, no entanto, o sistema é previamente preparado para lidar com problemas em especial, como o acesso a um dado inexistente no banco, exemplificado na imagem abaixo:

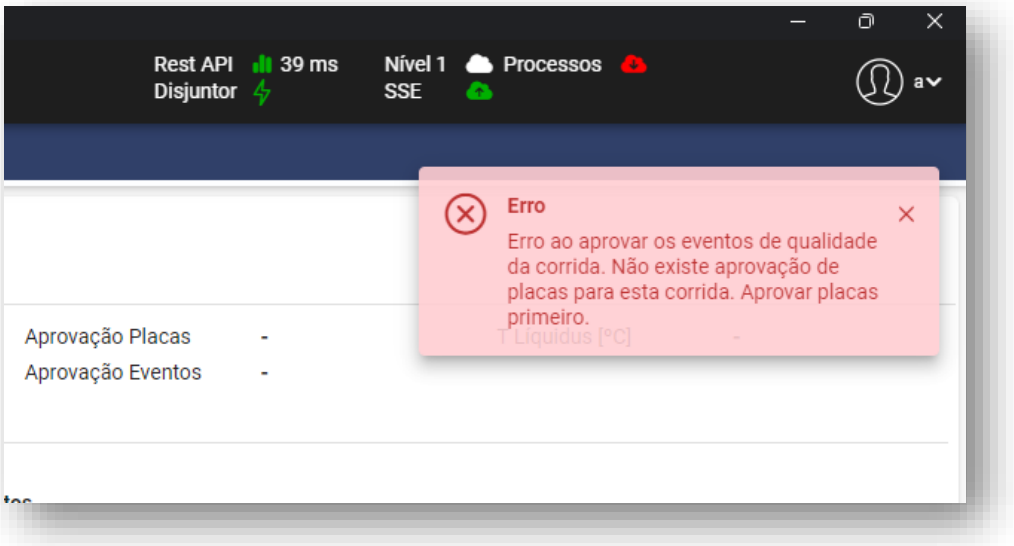


Figura 32 - EXEMPLO DE ERRO ESPECÍFICO RECONHECIDO PELO SISTEMA DA MLC1

A detecção de problemas em cada um destes casos pode ser explorada separadamente.

4.1.1 Problemas genéricos

Quando uma notificação de erro é lançada, o primeiro passo para investigação é recorrer às Ferramentas de Desenvolvedor disponibilizadas pelo navegador, acessíveis pelo atalho F12 no teclado e mostradas na imagem abaixo:

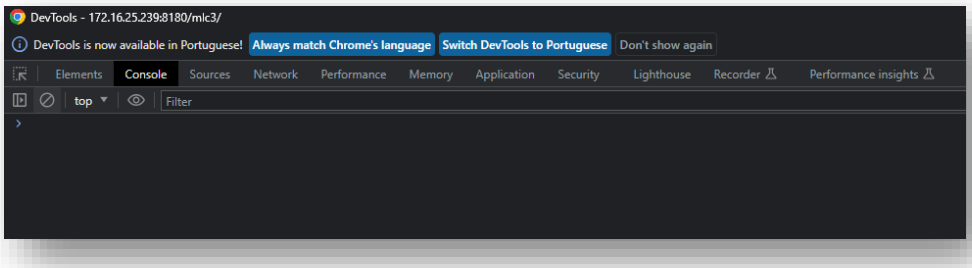


Figura 33 - FERRAMENTAS DE DESENVOLVEDOR DISPONIBILIZADAS PELO NAVEGADOR GOOGLE CHROME

A aba “Console” registra informações sobre o funcionamento do sistema, incluindo erros. Problemas de *front-end* tem a causa, o componente e o método apontados diretamente, conforme mostra a imagem abaixo:

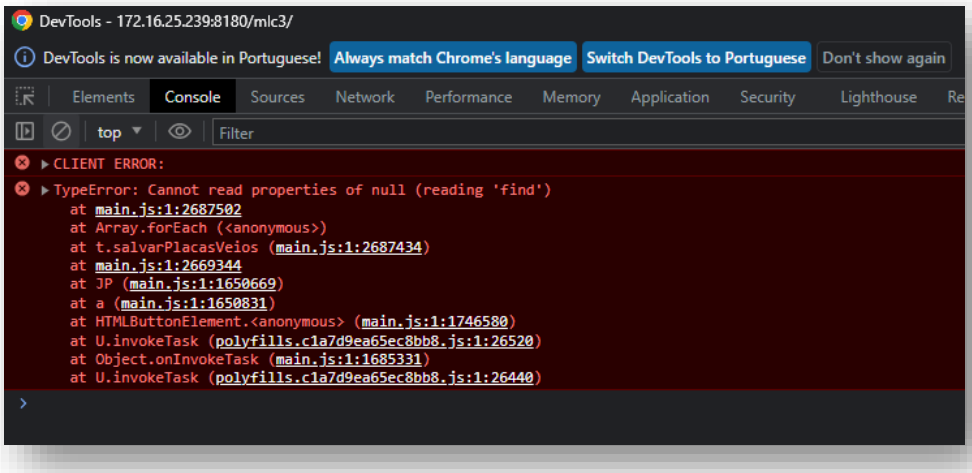


Figura 34 - EXEMPLO DE ERRO OCORRIDO NO FRONT-END

Para investigar como o erro ocorreu, a aba "Sources" oferece acesso ao código-fonte do *front-end* através do atalho Ctrl+P. Além disso, são fornecidas ferramentas para depuração, como ilustrado na imagem abaixo:

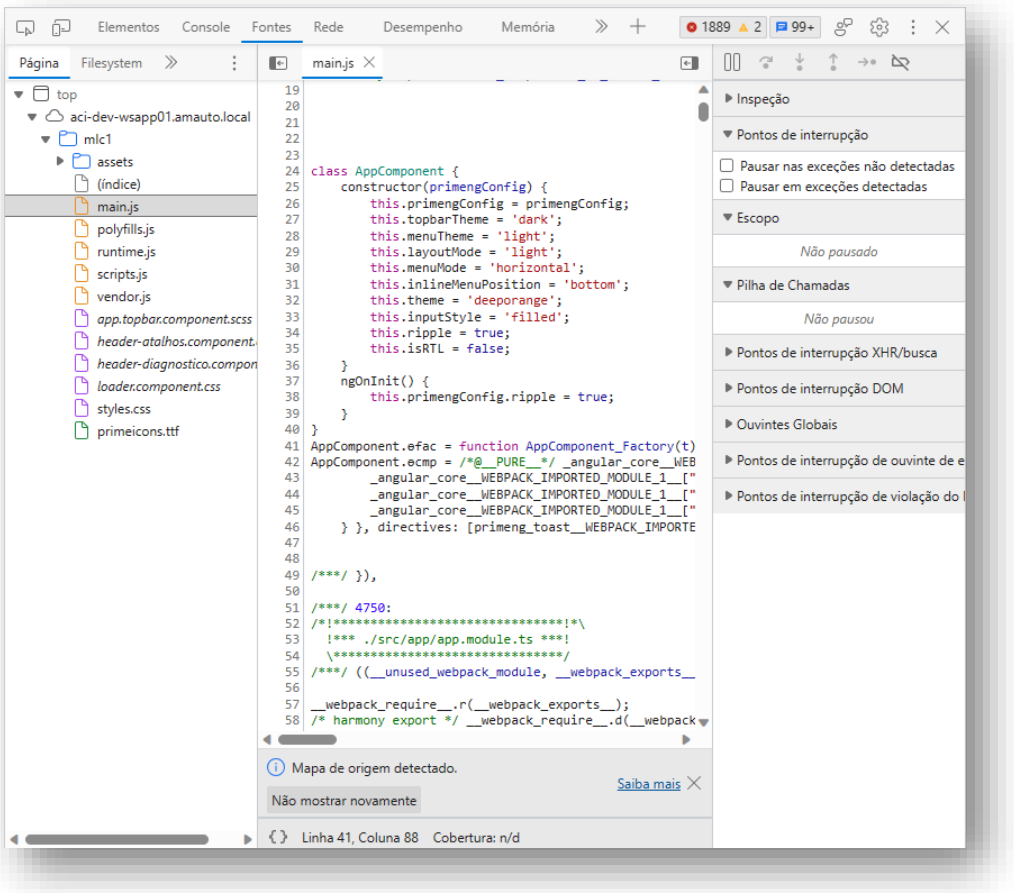


Figura 35 - RECURSO DE DEPURAÇÃO DAS FERRAMENTAS DE DESENVOLVEDOR

Em outros casos, o console aponta não o trecho de código que gera o problema, mas sim o caminho Web Service de comunicação com o *back-end*, como mostra a imagem abaixo. Trata-se de um indicativo de que o problema ocorreu a nível de *back-end* e foi propagado e enviado ao *front-end* como resposta à requisição.

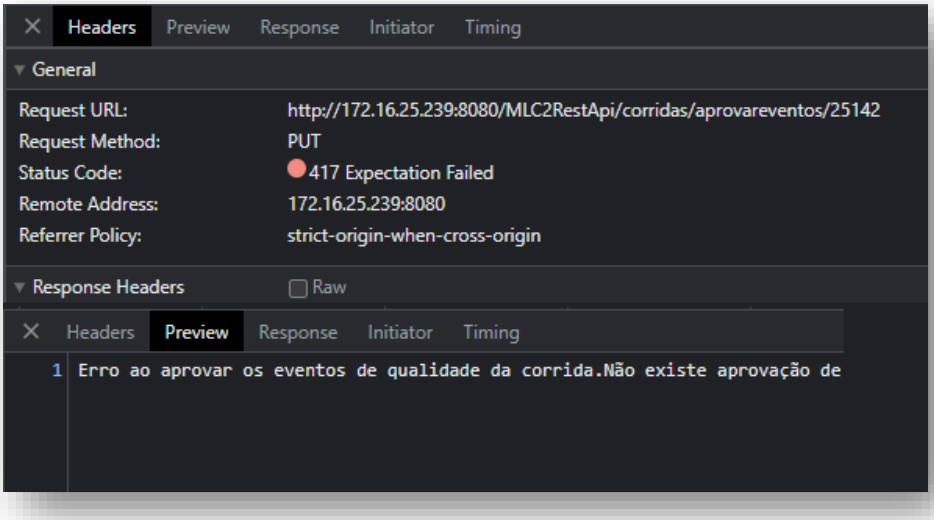


Figura 36 - ERRO DE BACK-END APONTADO COM O CAMINHO DA REQUISIÇÃO

Figura 37 - OUTRO EXEMPLO DE ERRO DE BACK-END SENDO APONTADO COM O CAMINHO DA REQUISIÇÃO

A investigação do foco deste tipo de problema pode ser possível consultando-se o registro de atividades (log) do *back-end*. Considerando que se utilize o servidor de aplicação *WildFly* 26.1.1 para o desenvolvimento, este registro é salvo no arquivo server.log, localizado no caminho standalone/log dentro do diretório correspondente ao servidor. Quando o *WildFly* é executado em integração com a IDE Netbeans, o log é exibido e atualizado em tempo real no console de saída desta.

```
:MLCTwinOperationSimulation:compileJavaNote: Hibernate JPA 2 Static-Metamodel Generator 5.4.33.Final
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

:MLCTwinOperationSimulation:refreshAllEnvironments
:MLCTwinOperationSimulation:environment> Processing Main environment
:MLCTwinOperationSimulation:environment> Main environment successfully updated
:MLCTwinOperationSimulation:processResources UP-TO-DATE
:MLCTwinOperationSimulation:classes

BUILD SUCCESSFUL

Total time: 1 mins 8.952 secs
18:11:26: Execution finished 'classes'.
```

Figura 38 - EXEMPLO DE UM REGISTRO DO CONSOLE PARA A CONSTRUÇÃO DO MÓDULO

É importante destacar que problemas no *back-end* podem ter origem não apenas no próprio *back-end*, mas também ser causados por parâmetros incorretos enviados pelo *front-end*. É possível avaliar os valores dos parâmetros passados para a requisição e localizar o serviço do *front-end* que se comunica com o *back-end*, seguindo o caminho indicado no console das Ferramentas de Desenvolvedor. Se for constatado que os parâmetros são passados incorretamente para o serviço, também é possível localizar o ponto de invocação do serviço no arquivo TypeScript do componente correspondente.

4.1.2 Problemas específicos

Notificações de erros com mensagens específicas para cada situação oferecem uma facilidade adicional para identificar onde o problema ocorre. Essas mensagens estão registradas nas classes especializadas “component.ts”, localizadas no diretório do sistema, normalmente nomeadas como “Management” pelo *front-end* do MLC1, se o erro ocorrer em um sistema MLC específico. No arquivo, cada mensagem é atribuída a uma propriedade específica, que permite localizar onde o erro é lançado. Consulte a imagem abaixo para visualizar um exemplo:

Figura 39 - MENSAGEM DE ERRO ENCONTRADA NO ARQUIVO DE PROPRIEDADES, E PROPRIEDADE CORRESPONDENTE ENCONTRADA EM TRECHO DE CÓDIGO COM LANÇAMENTO DE ERRO

Na imagem acima é possível observar uma mensagem de erro encontrada no arquivo de propriedades e, em seguida, a propriedade correspondente é identificada durante o lançamento de uma exceção. Isso evidencia como e em qual situação o erro é lançado pelo sistema. Resta investigar a origem do problema, conforme detalhado na seção **4.1.1**.

4.2 Problemas não reconhecidos pelo sistema

Existem situações em que a resposta do sistema não é conforme a esperada pelo usuário, porém também não é adversa ao tipo de resposta esperado pelo sistema, de modo que este não seja capaz de acusar um erro. A seguir, são apresentadas as principais situações em que isso ocorre e como detectar os problemas.

4.2.1 Problemas de visualização

Em alguns casos, pode ser necessário corrigir ou alterar aspectos visuais das telas do sistema, tais como títulos, labels de botões, posições de tabelas e figuras, etc., definidos nos arquivos HTML e CSS do respectivo componente da tela no *front-end*.

Antes de modificar quaisquer elementos, verifique se a resolução utilizada pelo Usuário é 1920 x 1080.

Os resultados de se alterar propriedades como labels e ordens de elementos são definidos e previsíveis, conforme desejado pelo desenvolvedor. Porém, alterar parâmetros como dimensões e posições de elementos na tela pode demandar várias tentativas até que se obtenha resultado satisfatório. Para casos como este, as Ferramentas de Desenvolvedor, introduzidas na seção 4.1.1, permitem inspecionar elementos visuais e testar de imediato os efeitos de alterações em suas propriedades. Tome como exemplo a alteração do *Label* em tela da imagem abaixo:

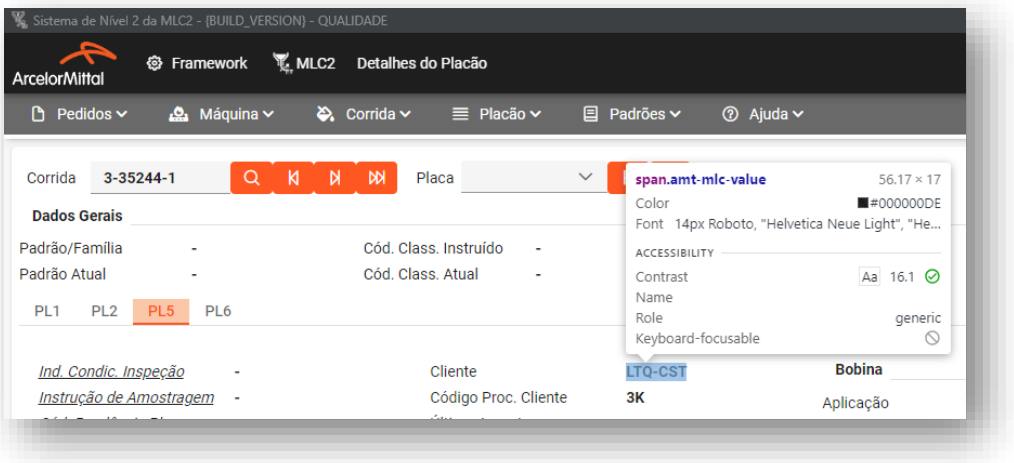


Figura 40 - LABEL A TER DIMENSÕES DUPLICADAS ATRAVÉS DAS FERRAMENTAS DE DESENVOLVEDOR

No canto superior esquerdo da tela das Ferramentas de Desenvolvedor, está disponível a opção de inspeção de elemento, necessária para identificar as propriedades do botão. A imagem abaixo exibe a opção selecionada.

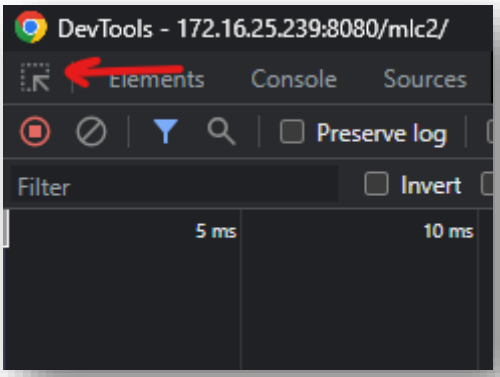


Figura 41 - OPÇÃO DE INSPEÇÃO DE ELEMENTO SELECIONADA NA TELA DE FERRAMENTAS DE DESENVOLVEDOR

Então, o elemento a ser inspecionado, no caso o *Label*, é selecionado.

Como mostra a imagem abaixo, a aba “Elements” da tela de Ferramentas de Desenvolvedor exibe a definição do *Label*, presente no arquivo HTML do componente. Note que o elemento pertence à classe “amt-mlc-value”. Essa informação ajuda a identificar, na aba “Styles”, a definição das propriedades visuais do *Label*. Os “Styles” podem ser adicionados e removidos para testar qual a melhor solução.

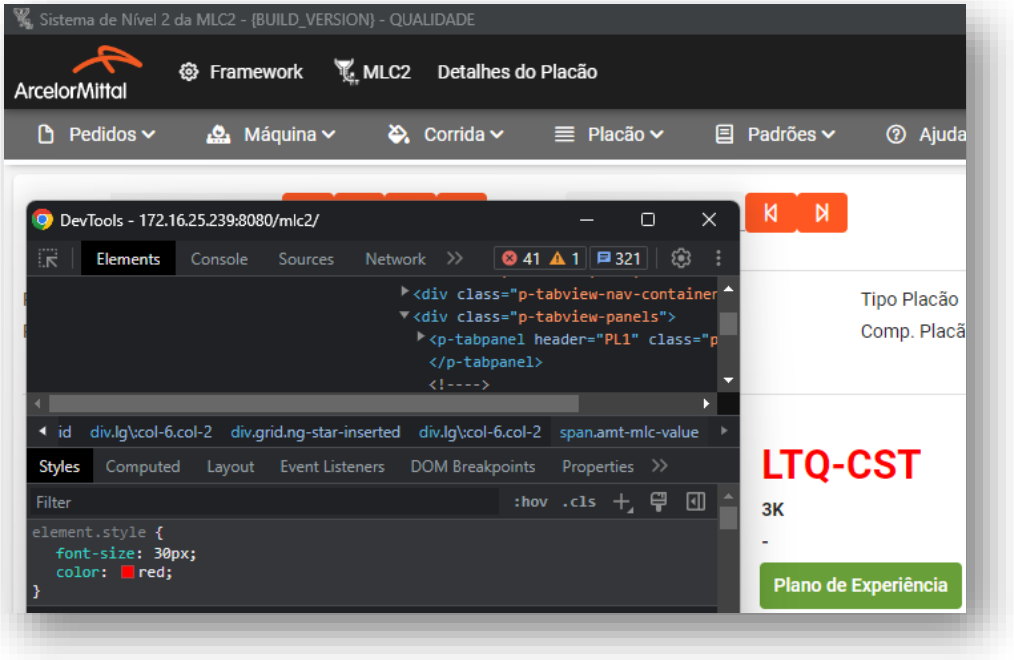


Figura 42 - IDENTIFICAÇÃO DAS PROPRIEDADES DO LABEL NA TELA DE FERRAMENTAS DE DESENVOLVEDOR

Alterando-se os valores “font-size” e “color” é possível observar imediatamente os efeitos, conforme mostra a imagem acima.

4.2.2 Problemas de recuperação de dados

Em algumas telas, é necessário obter dados específicos de entidades, geralmente filtrados com base em atributos específicos. No entanto, pode ocorrer de apenas alguns ou nenhum dado ser retornado, mesmo havendo dados disponíveis no banco que atendam às características solicitadas.

Nesses casos, é importante verificar se os dados estão sendo perdidos no *back-end*. É recomendado iniciar a investigação a partir do ponto em que o controlador invoca o método do repositório para acessar os dados. A execução desse método pode resultar em duas situações possíveis:

a) Os dados são recuperados de forma incompleta: nesse caso, o problema está relacionado à montagem da consulta ao banco. Isso pode ocorrer devido a pelo menos um dos três fatores básicos: os parâmetros (ou seus valores) passados para a consulta estão incorretos, o método do repositório não monta corretamente a consulta ou está sendo invocado um método equivocado do repositório nesse ponto.

b) Os dados são recuperados completamente: isso indica que o problema não está na recuperação dos dados em si, mas sim no seu tratamento. Nessa situação, é necessário depurar o restante do método do controlador e verificar se há algum ponto em que as informações estão sendo indevidamente descartadas.

É possível que a investigação constate que não há perda de dados no *back-end*. Nesse caso, alguns pontos principais devem ser verificados:

a) Transferência de dados entre *back-end* e *front-end*: na maioria dos casos, os dados são transmitidos do *back-end* para o *front-end* no formato JSON. Para o envio, as entidades são traduzidas para o formato JSON, respeitando os nomes dos atributos e seus valores. No *front-end*, o objeto JSON é adaptado para um modelo TypeScript com os mesmos atributos e valores. Se algum desses atributos não existir no modelo ou não estiver nomeado de acordo com o que é passado pelo objeto JSON, a informação correspondente será perdida.

b) Processamento da subscrição: o processamento dos dados recebidos pelo método TypeScript pode estar operando de forma equivocada, descartando informações de maneira inadequada.

c) Valores e métodos necessários para os elementos de tela: também é importante verificar se o arquivo HTML do componente da tela faz referência correta aos valores a serem exibidos em elementos de visualização, como tabelas, e aos métodos a serem invocados por elementos de ação, como botões.

Essa análise em relação à perda de dados pode ser aplicada de forma semelhante para o caso de recebimento de dados adulterados.

4.2.3 Problemas de atualização

4.2.3.1 Problemas de atualização de dados

Muitas ações nos sistemas envolvem criar ou modificar dados no banco, como mover uma placa em um pátio. Por meio da JPA Domain Store, esses sistemas lidam com essas ações criando ou recuperando os dados do banco como objetos Java, atualizando seus atributos e persistindo o novo objeto sobre sua versão anterior, se houver. Portanto, para detectar problemas na atualização do banco, é necessário verificar os valores atribuídos aos objetos persistidos.

No entanto, também é comum que um sistema precise ter seus dados atualizados com base em informações provenientes de outro sistema. Nesses casos, é importante avaliar se a troca de mensagens JMS/OMQ entre os sistemas está ocorrendo corretamente.

O envio de mensagens com informações incorretas ou ausentes compromete o processamento pelo sistema receptor. Portanto, problemas em funcionalidades que dependam de mensagens, como diferentes tipos de movimentações de placas, podem ser causados pelo envio de uma mensagem com dados incorretos ou por uma falha no seu tratamento.

Para detectar problemas na mensagem enviada, é necessário localizar onde o modelo da mensagem está sendo instanciado e enviado. Após isso, é possível avaliar os valores atribuídos ao modelo e verificar a implementação do modelo em si. Caso não seja identificado nenhum problema no envio, é necessário analisar o tratamento da mensagem pelo sistema receptor.

4.2.3.2 Problemas de atualização de tela

Durante a utilização do sistema, surgem desafios em garantir a atualização imediata de certos elementos visuais quando determinadas ações são executadas. Por exemplo:

Ao movimentar uma placa entre endereços no mapa da Máquina, é necessário que o ícone correspondente exiba a transição na tela.

Após a conclusão de um carregamento, a lista de veículos disponíveis para descarregamento precisa ser atualizada.

Para solucionar esses problemas e permitir atualizações em tempo real, é possível utilizar a comunicação via WebSocket e enviar mensagens socket, no contexto apresentado, a classe responsável por tratar isso é a ComunicacaoWebSocket. Essa classe oferece métodos para enviar mensagens de diferentes tipos para terminais, estações ou grupos de terminais específicos. Um trecho de código relacionado ao envio de uma mensagem socket é mostrado na imagem abaixo:

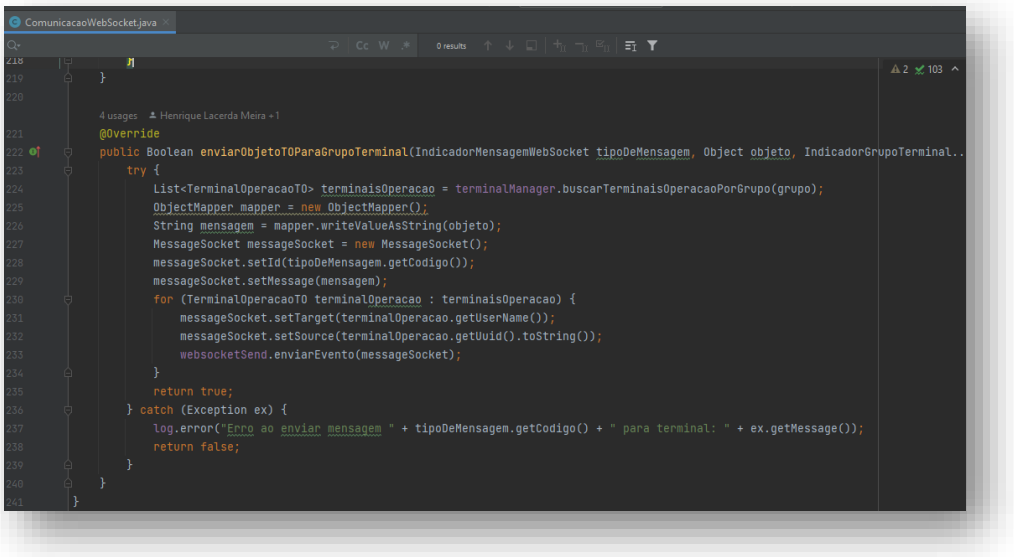


Figura 43 - CONSTRUÇÃO E ENVIO DE MENSAGEM SOCKET

O exemplo acima ilustra a construção de uma mensagem socket para atualizar a posição de uma informação, utilizados por outras classes para a chegada e tratamento desta, de medição Celox e para a TAG DEVOLUCAO_PANELA_FECHAMENTO no rastreamento de placas.

A Classe RastreamentoPlacaService é responsável por tratar a comunicação via websocket no contexto específico do rastreamento de placas. Ela possui métodos para enviar mensagens de diferentes tipos para terminais, estações ou grupos de terminais específicos. O código apresentado na Classe RastreamentoPlacaService demonstra a utilização do recurso ComunicacaoWebSocket para enviar mensagens de notificação popup em determinadas situações.

Essa abordagem de comunicação por meio de mensagens socket facilita a atualização em tempo real dos elementos visuais, garantindo uma melhor experiência do usuário e uma maior eficiência na interação com o sistema.

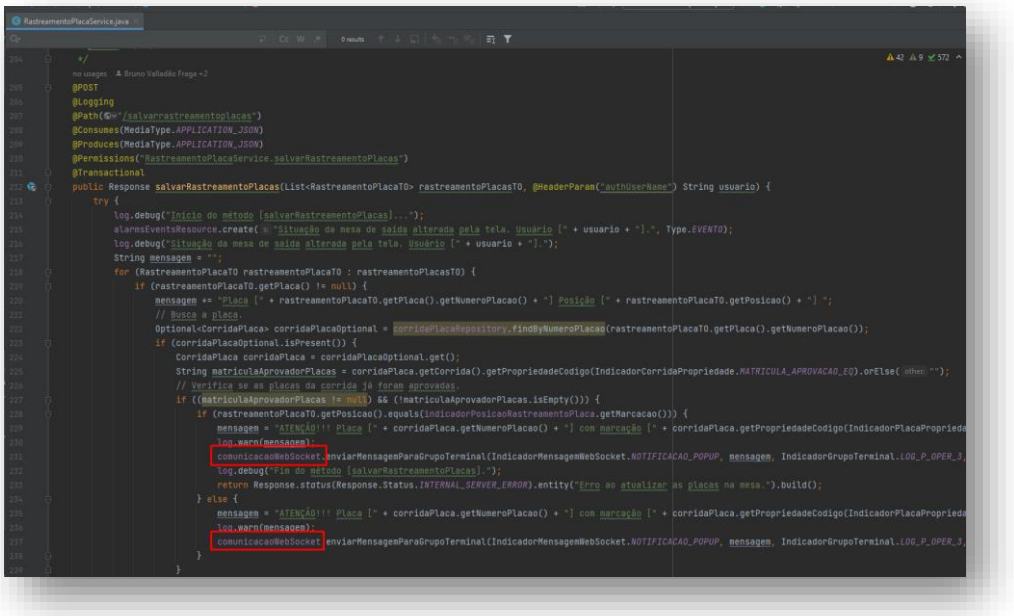


Figura 44 – CLASSE RASTREAMENTOPLACASERVICE UTILIZANDO A COMUNICACOWEBSOCKET

5 Corrigindo Problemas

Após identificar os problemas conforme descrito na seção 4, é necessário realizar as correções adequadas. No entanto, simplesmente modificar o código no local onde o problema ocorre nem sempre é suficiente. Nesta seção, abordaremos os procedimentos e cuidados a serem considerados durante a manutenção corretiva.

5.1 *Back-end*

No caso dos projetos MLC1Core e MLCCore, é geralmente suficiente realizar alterações pontuais no código para concluir a manutenção com êxito. No entanto, é importante destacar que os arquivos de propriedades específicos de cada sistema de MLC, como o Management mencionado na seção 4.1.2 e o ValidationMessages que será abordado em mais detalhes nesta seção, também podem exigir alterações. Essas alterações podem ser necessárias principalmente para incluir novas propriedades relacionadas ao sistema, como mensagens de notificação.

Uma vez que esses arquivos de propriedades estão definidos separadamente para cada projeto MLC, é essencial garantir que as alterações feitas em um deles sejam reproduzidas em todos os projetos nos quais a respectiva propriedade seja aplicável. Dessa forma, evitamos inconsistências entre os sistemas e garantimos que as modificações sejam refletidas de maneira consistente em todo o ambiente MLC.

5.2 *Front-end*

Uma vez que o projeto foi "clonado", a manutenção do sistema se limita a correção de código no projeto.

O projeto pode ser clonado com o **Git Bash** através do seguinte repositório: https://tfsams.visualstudio.com/DefaultCollection/AMT_N2_MLC/_git/mlc-*front-end*

Para abrir o Git Bash basta clicar com o botão direito na pasta onde o projeto será clonado e clicar em "**git bash here**". No caso do Windows 11, é necessário clicar em "Mais Opções" antes. Lembrando que é preciso ter o *git* instalado no computador onde será feito o desenvolvimento.

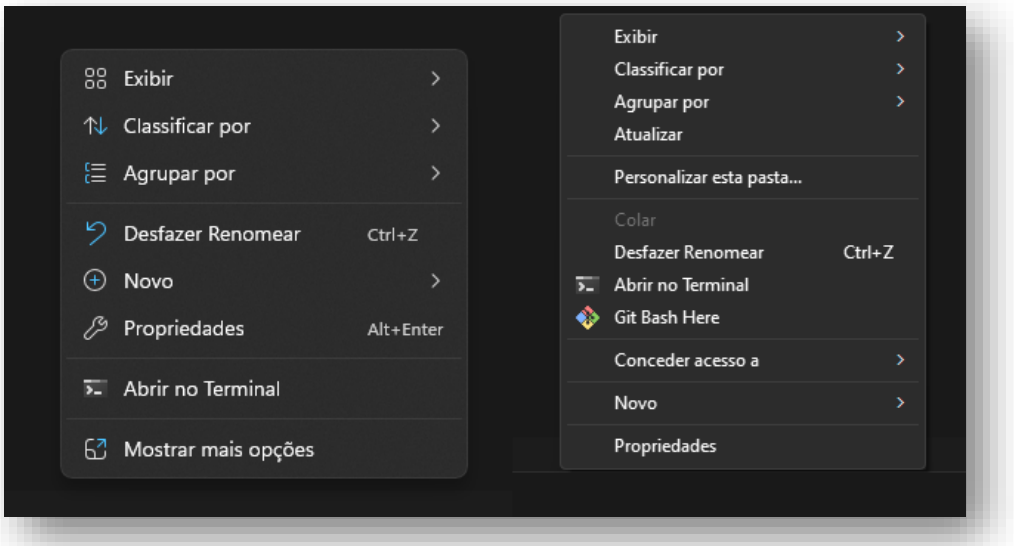


Figura 45 - ABRINDO O GIT BASH

Ao abrir o terminal do Git Bash, digitar: git clone "https://tfsams.visualstudio.com/DefaultCollection/AMT_N2_MLC/_git/mlc-*front-end*", e pressionar "Enter".

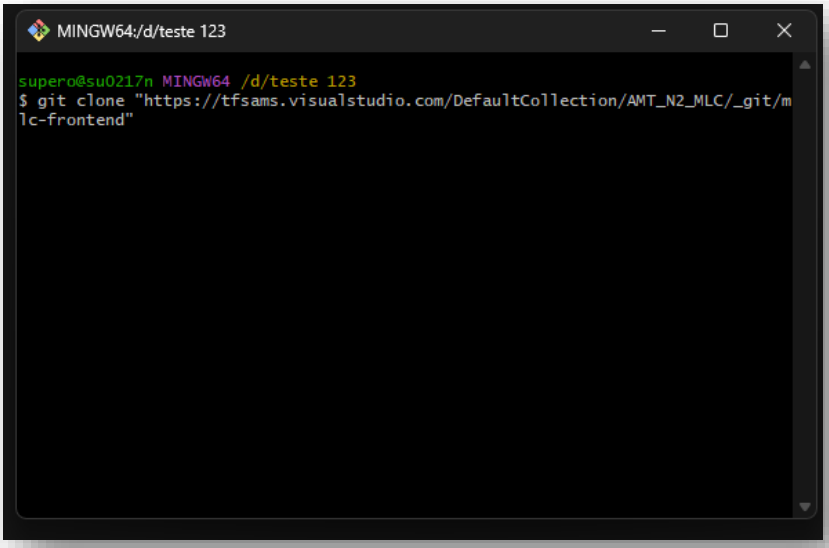


Figura 46 - REALIZANDO O CLONE DO PROJETO (COMANDO “GIT CLONE”)

Para manipular o repositório da ArcelorMittal Tubarão e necessário ter uma credencial autorizada, e com posse desta, basta fornecer as informações de login na janela “Git Credential Manager”.

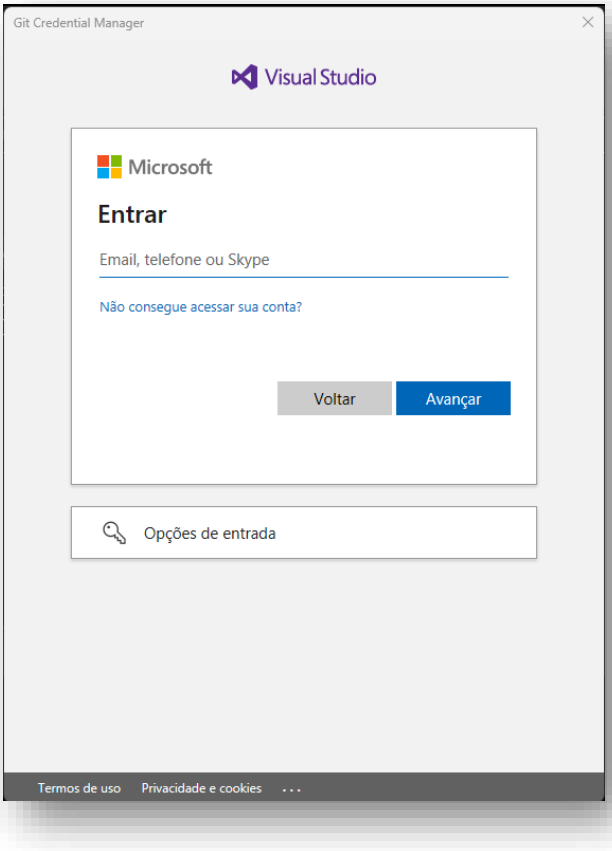


Figura 47 - INCLUINDO CREDENCIAIS DO REPOSITÓRIO NO GIT CREDENTIAL MANAGER

Após o procedimento acima, o projeto será clonado na pasta selecionada.

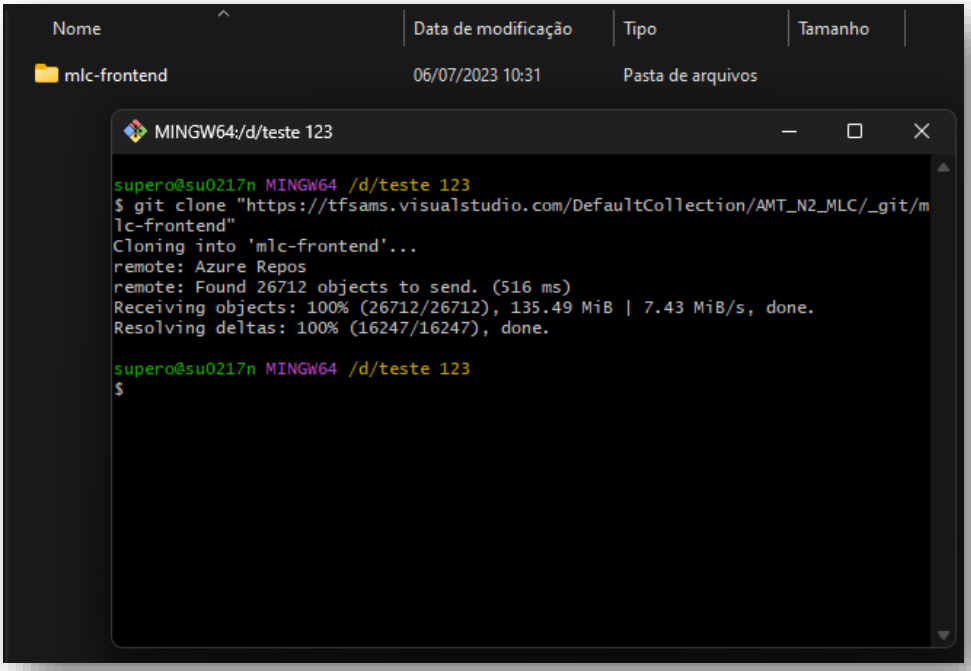


Figura 48 - MENSAGEM DE CLONE REALIZADO COM SUCESSO

Agora, verifique a versão do **Node** instalada no computador. A versão pode ser verificada atráves do CMD, ou “prompt”, utilizando o comando “node -v”, onde a versão esperada deve ser a “**14.20.0**”.

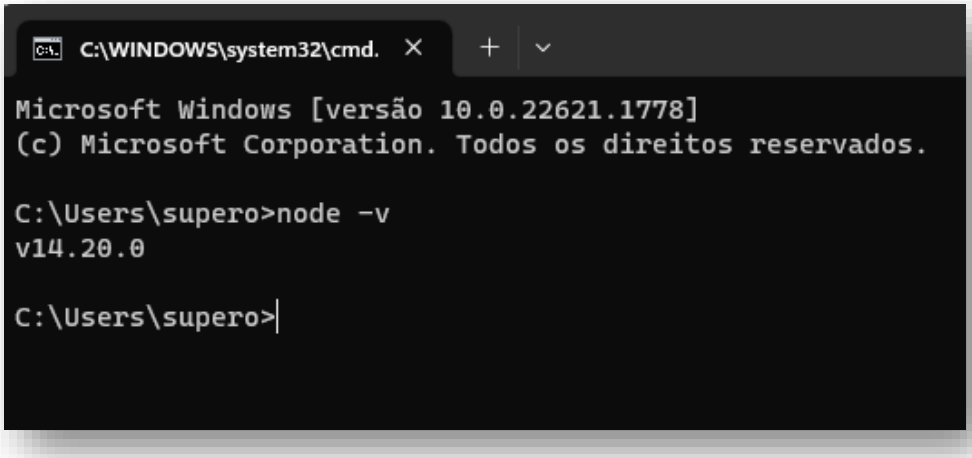


Figura 49 - VERIFICANDO A VERSÃO DO NODE

A versão do **Npm** instalada no computador também deve ser verifica, e para isso pode ser utilizado, através do CMD ou “promp”, o comando “npm -v”, onde a versão esperada deve ser a “**6.14.17**”.

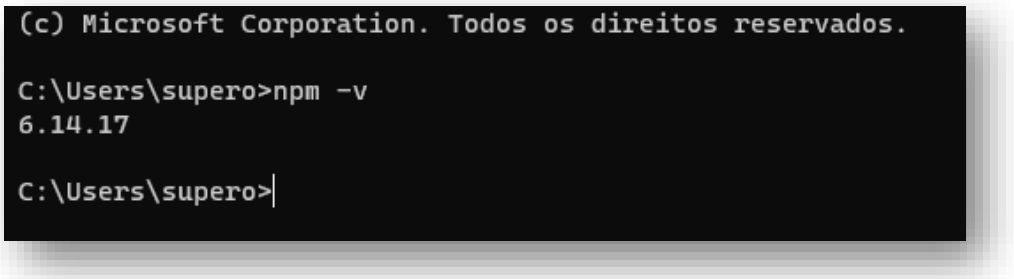


Figura 50 - VERIFICANDO A VERSÃO DO NPM

Agora, para usuários da plataforma Visual Studio Code, clique em “File” e depois em “Open Folder”, conforme imagem abaixo:

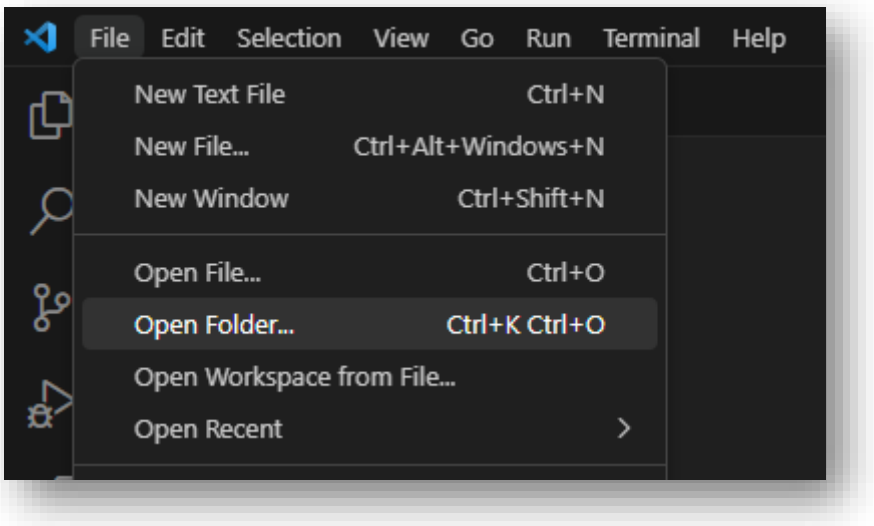


Figura 51 - ABRINDO O DIRETÓRIO DO PROJETO NO VISUAL STUDIO CODE

Navegue até a pasta do projeto e clique em “Selecionar Pasta”. Marque a caixa de seleção e clique em “Yes, i trust the authors”.

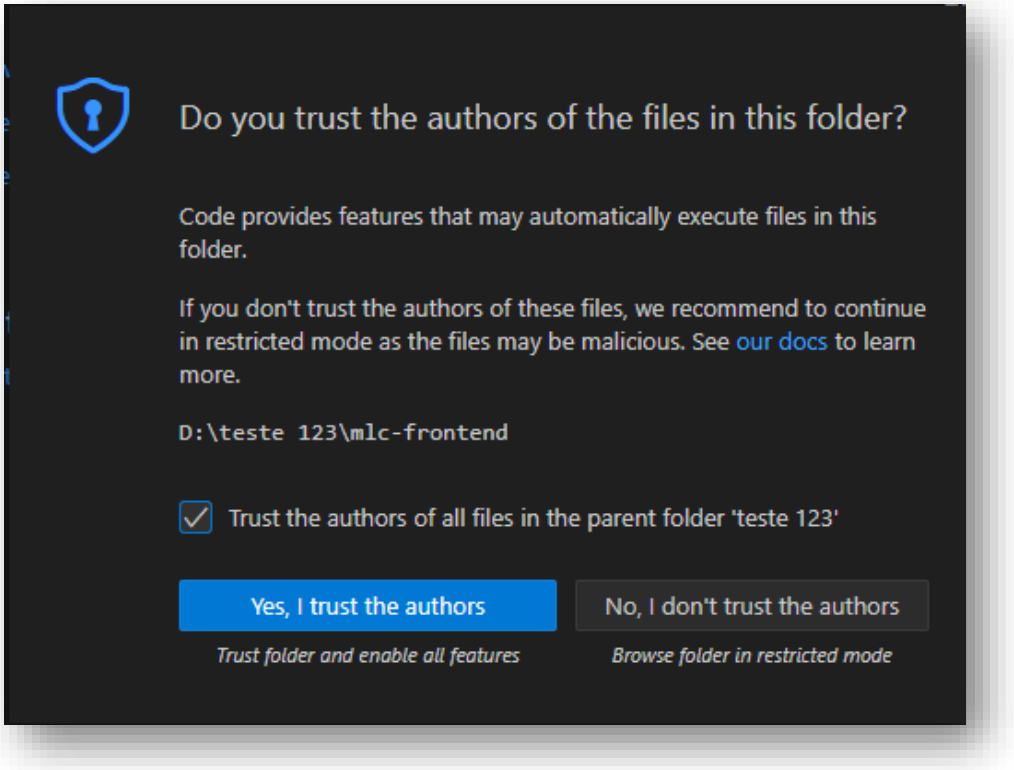


Figura 52 - CONFIRMANDO DIRETÓRIO COMO CONFIÁVEL NO VISUAL STUDIO CODE

Agora o próximo passo é a instalação de pacotes e dependências do projeto. Clique no menu “Terminal” e em seguida na opção “New Terminal”.

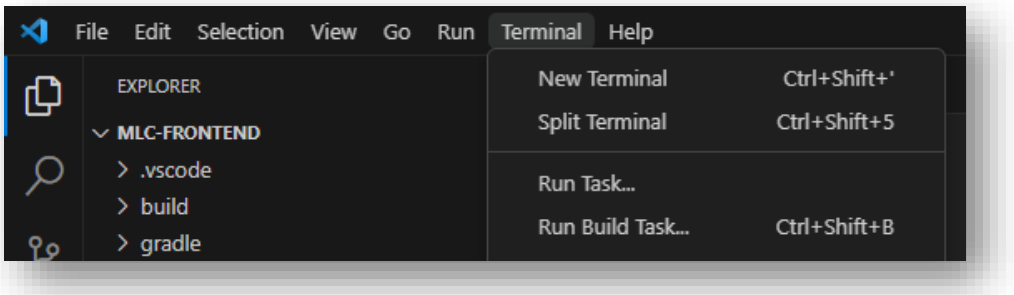


Figura 53 - ABRINDO UM TERMINAL NO VISUAL STUDIO CODE

Em seguida, na parte inferior, clique na seta para baixo e abra 2 terminais do Git Bash.

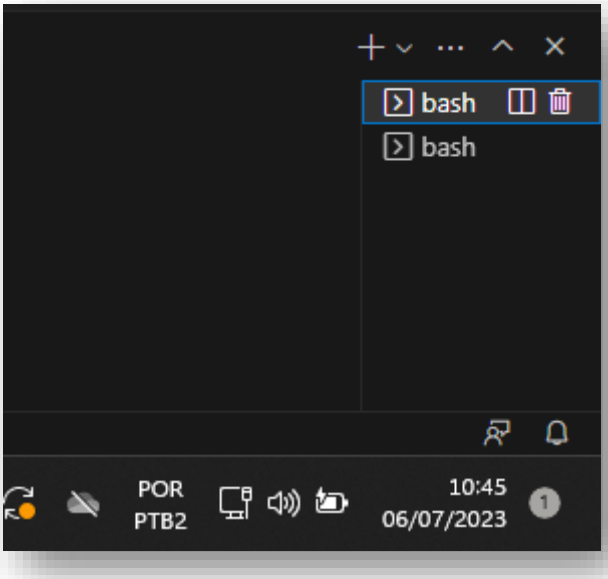


Figura 54 - ABRINDO TERMINAIS DO GIT BASH NO VISUAL STUDIO CODE

No primeiro terminal, digite “cd shell”, para entrar na pasta do Framework.

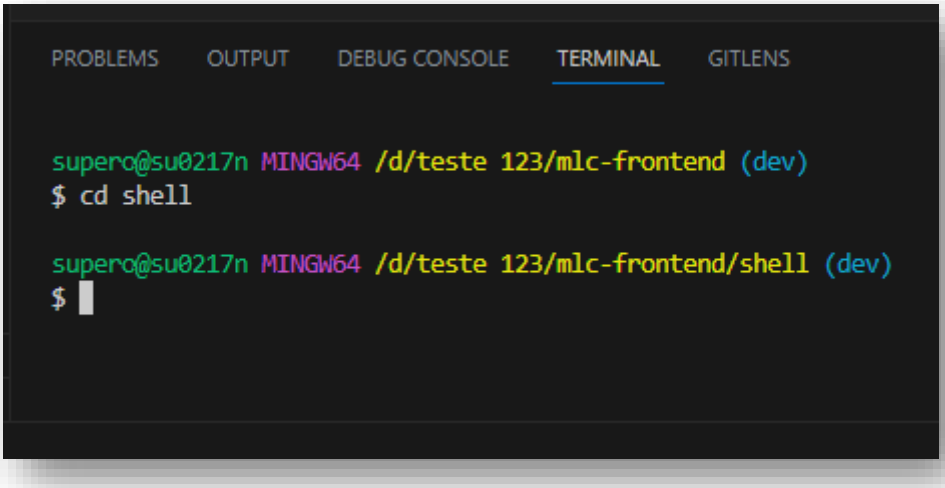


Figura 55 - ENTRANDO NA PASTA DO FRAMEWORK ATRAVÉS DO TERMINAL NO VISUAL STUDIO CODE

No segundo terminal, digite “cd mlc”, para entrar na pasta da Aplicação

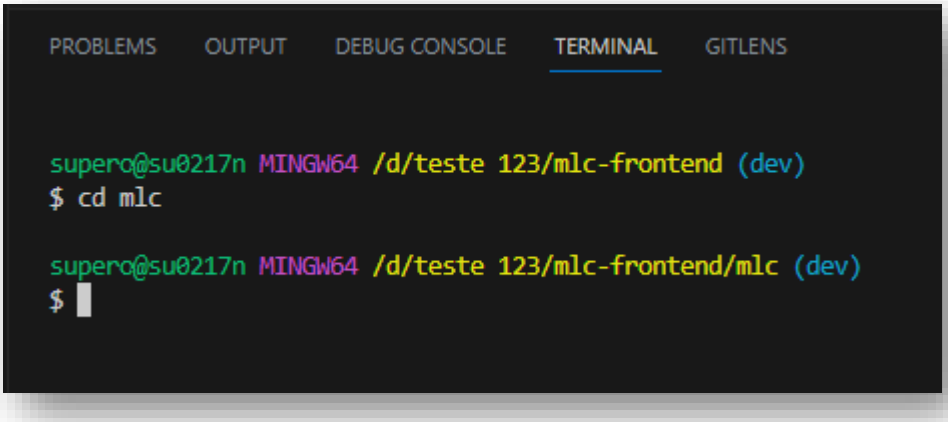


Figura 56 - ENTRANDO NA PASTA DA APLICAÇÃO ATRAVÉS DO TERMINAL NO VISUAL STUDIO CODE

Em ambos os terminais, utilize o comando “npm install” ou “yarn” (caso o tenha instalado) para instalar todas as dependências do projeto.

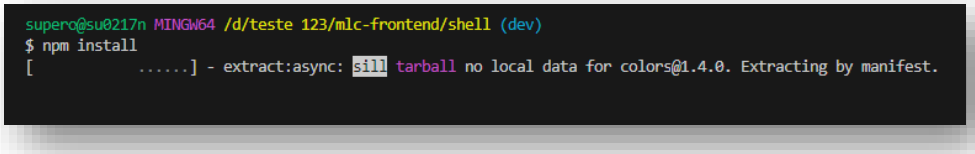


Figura 57 - INSTALANDO AS DEPENDÊNCIAS DO FRAMEWORK COM “NPM INSTALL”

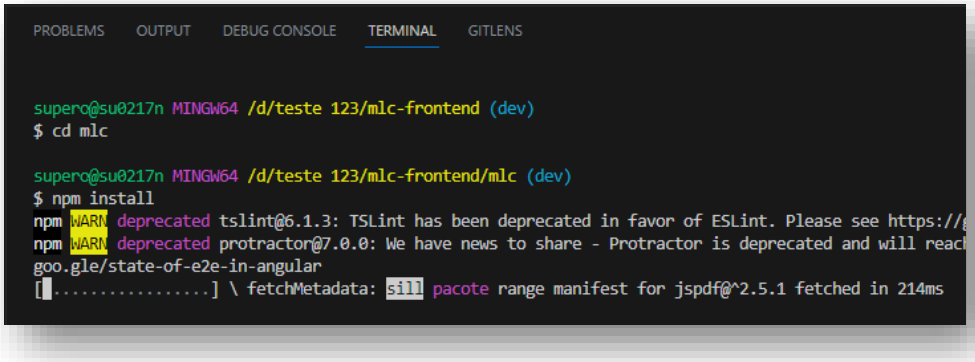


Figura 58 - INSTALANDO AS DEPENDÊNCIAS DA APLICAÇÃO COM “NPM INSTALL”

O próximo passo é configurar o apontamento do ambiente. O projeto é o mesmo para todas as MLC’s, logo para rodá-lo localmente para desenvolvimento é preciso informar no comando no

terminal da “shell” para qual MLC apontar e qual ambiente será o alvo. Exemplo: “ng serve --project shell -o --port 5207 --host 127.0.0.1 -c local-mlc1” ou “yarn start –c local-mlc1”.

O Procedimento é o mesmo para as MLC’s 2 ou 3, basta trocar no final da linha de comando qual MLC deseja estar rodando.

Ao final, você pode também criar um atalho do Google Chrome em sua área de trabalho, colocando na caixa “Destino” os comandos abaixo:

"C:\Program Files\Google\Chrome\Application\chrome.exe" --user-data-dir="%USERPROFILE%" --test-type --disable-web-security --start-maximized --app=http://127.0.0.1:5207/#/MLC1/login”

Para as outras MLC’s, basta alterar o “MLC1” para a MLC desejada.

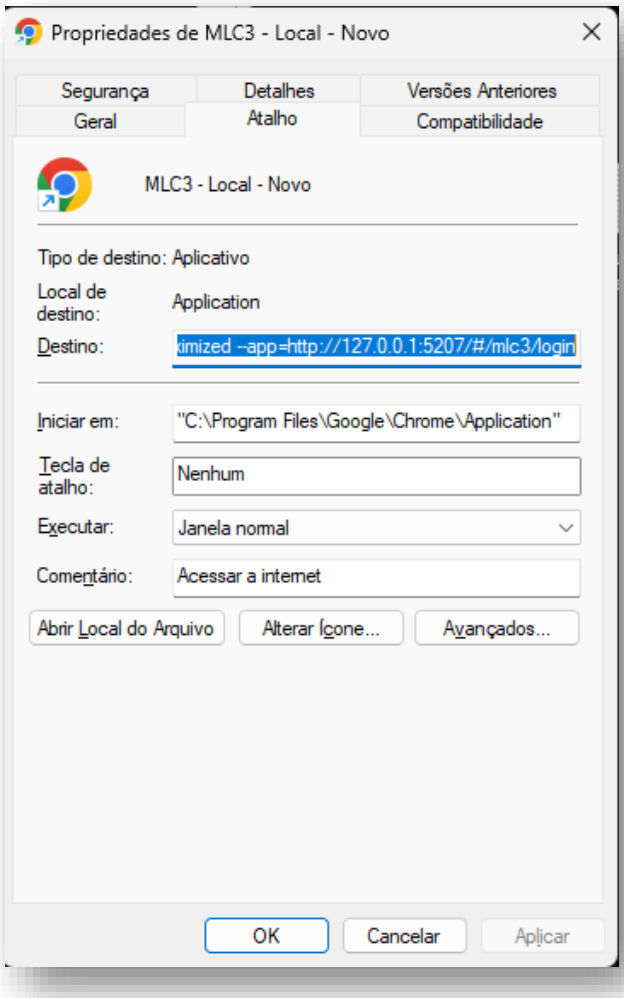


Figura 59 - CRIANDO ATALHO PARA ACESSAR O SISTEMA ATRAVÉS DO GOOGLE CHROME

Ao abrir o atalho criado no passo acima, a tela de Login da MLC desejada será exibida, conforme imagem abaixo:

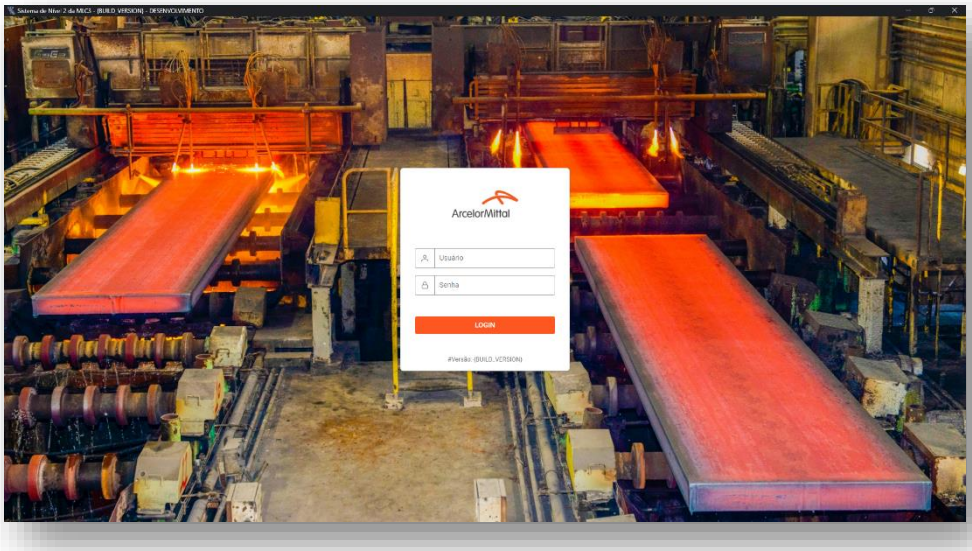


Figura 60 - ACESSANDO A TELA DE LOGIN DO SISTEMA

Informe o login e senha e clique em “Login”, e a tela inicial da aplicação será exibida.

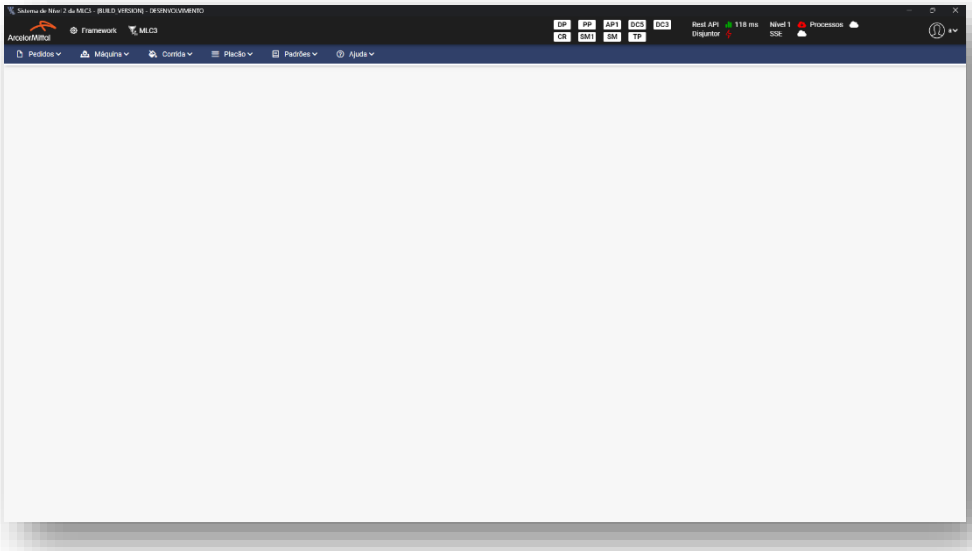


Figura 61 - ACESSANDO A TELA INICIAL DO SISTEMA

Sempre que for feita alguma alteração no codigo da MLC, é necessário compilar a aplicação, e para isso, basta utilizar o comando abaixo no terminal onde está aberta a pasta “mlc”: “npm run el-build-task && npm run el-package-task” ou “yarn el-build”.

Após finalizar a compilação, basta voltar a aplicação e pressionar “ctrl + F5” para verificar o resultado das alterações.

6 Aplicação do Zero

Esse capítulo mostrará como criar uma aplicação, no modelo do MLC, tanto do *front-end* como *back-end*.

6.1 *Front-end*

6.1.1 Pre-requisitos

O *front-end* do projeto é feito em Angular 2, por isso, antes de qualquer coisa, você precisa ter instalado na sua máquina:

- Node **14.20.0**;
- Npm **6.14.17**.

Tendo instalado os itens acima, verifique se eles foram instalados adequadamente executando os comandos abaixo num prompt de comando:

- node –v
- npm –v

6.1.2 Criando sua aplicação

6.1.1.1 Configurando seu Ambiente de Desenvolvimento

O passo a passo de criação e instalação do *front-end* está descrito no item **5.2 *Front-end***.

6.2 *Back-end*

Neste tópico, será explicado como criar o *back-end* da sua aplicação a partir do zero, começando pela criação do projeto raiz (root project). O projeto raiz será responsável por conter os módulos estendidos do framework da AMT, o módulo principal e as importações comuns a todos os módulos, localizados no common-gradle.

6.2.1 Criando o root Project da sua aplicação

Para iniciar um projeto seguindo os padrões, é necessário criar um projeto usando o Gradle. Isso implica na criação de um projeto raiz. O projeto raiz pode ser entendido como uma caixa que abrigará tanto os módulos do framework quanto a própria aplicação.

- No IntelliJ, acesse o menu Arquivo (File) e selecione a opção Novo projeto (New Project).
- Na tela que aparecerá, atribua um nome ao novo projeto, escolha uma localização (Location), marque as opções Language: JAVA, Build System: Gradle e selecione o JDK 1.8 Oracle Open JDK Version, conforme ilustrado na figura abaixo:

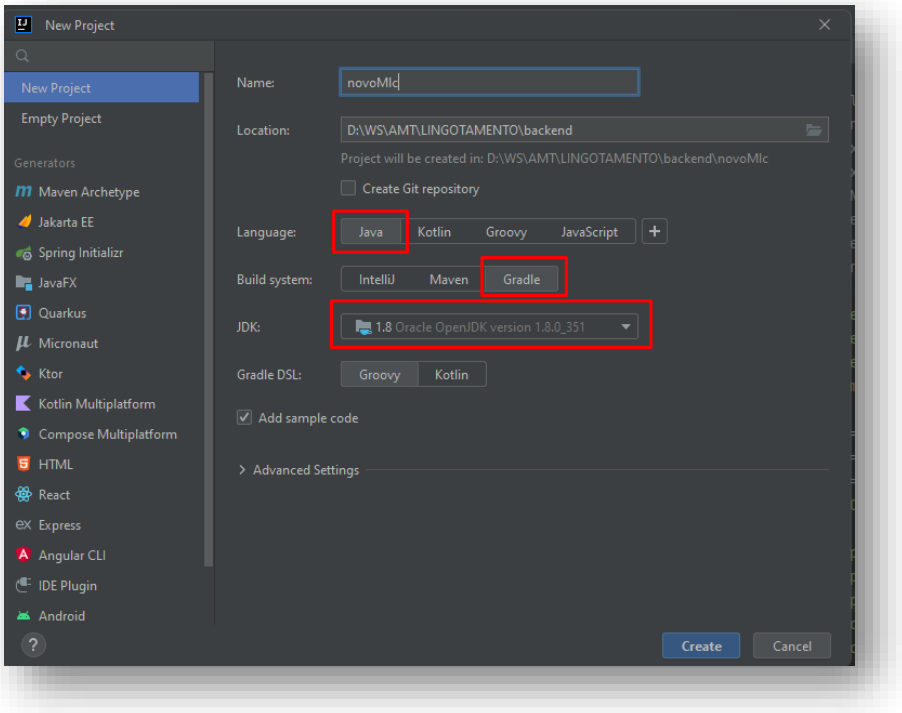


Figura 62 - GRADLE ROOT PROJECT

- Clicar em Criar (Create), a seguinte tela será exibida, fornecendo a estrutura básica para iniciar o novo projeto a partir do zero:

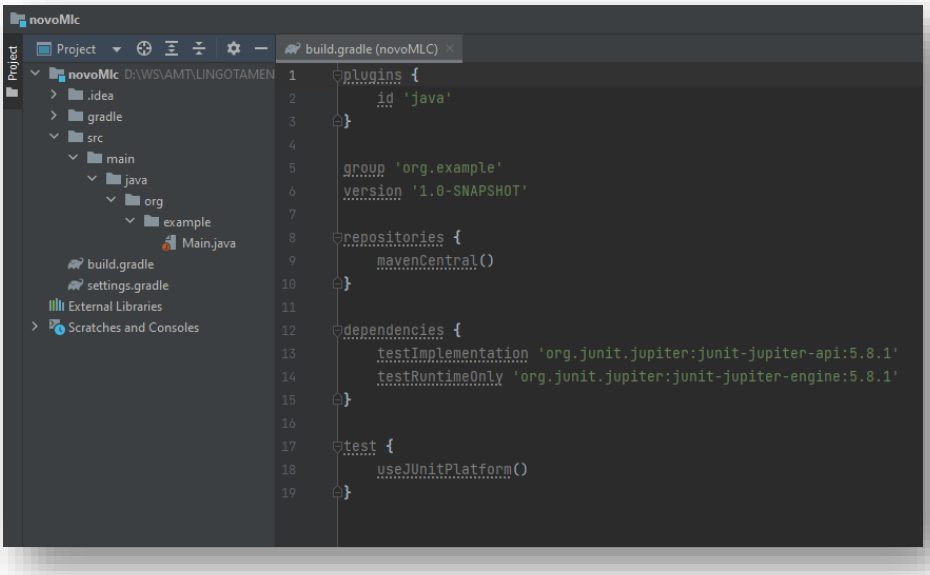


Figura 63 - NOME E LOCALIZAÇÃO GRADLE ROOT PROJECT

- Agora você tem a tela acima aberta e o seu projeto [root] foi criado, conforme mostra a figura abaixo:

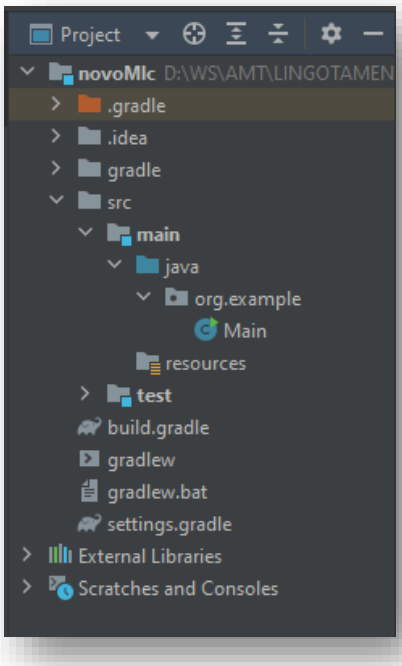


Figura 64 - ROOT PROJECT CRIADO

- **settings.gradle**: Este arquivo define quais subprojetos fazem parte da construção do seu projeto. Até a data de criação deste manual, o conteúdo deste arquivo é padrão e não precisa ser alterado.
- **build.gradle**: Arquivo este, explicado em detalhes na seção Build.Gradle. Este arquivo é executado durante a compilação do projeto raiz e permite a inclusão de scripts a serem executados nesse processo.
    - Você pode copiar integralmente este arquivo do projeto raiz nos repositórios de DevOps da AMT (https://spsprodeus22.vssps.visualstudio.com/_signedin), desde que tenha as devidas permissões para acesso.
- **common.gradle**: Também é executado durante o processo de compilação, mas, ao contrário do build.gradle, não permite a execução de scripts. Neste arquivo, são definidas as importações comuns a todos os subprojetos. Além disso, é nele que a variável "environment" é definida, indicando para qual ambiente a aplicação está sendo compilada para implantação (*WildFly* local, *WebSphere* HMG, *WebSphere* Produção, etc).
    - Se o seu novo projeto for configurado em um novo computador por exemplo, é possível copiar esse arquivo do projeto raiz de outra máquina já existente, observando apenas as variáveis "mavenGroupId" e "mavenVersion" dentro do common.gradle.

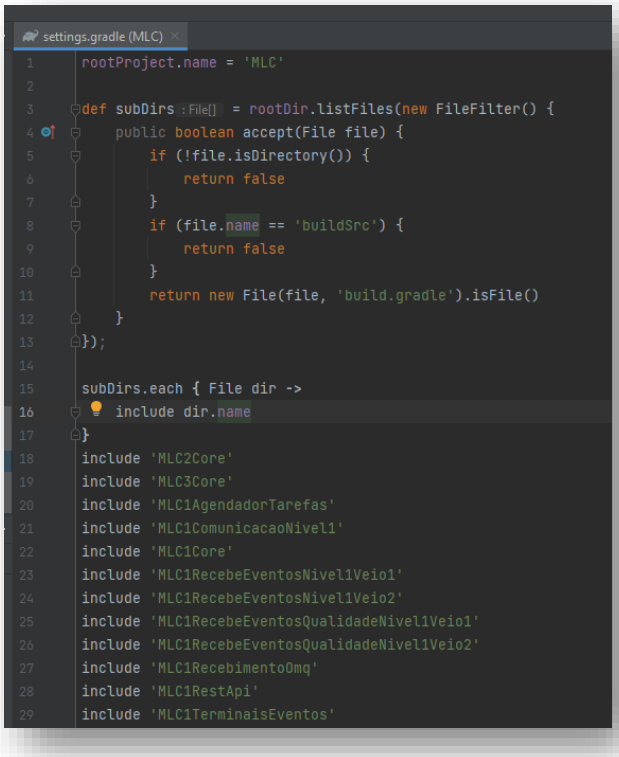


Figura 65 - ARQUIVO SETTINGS-GRADLE

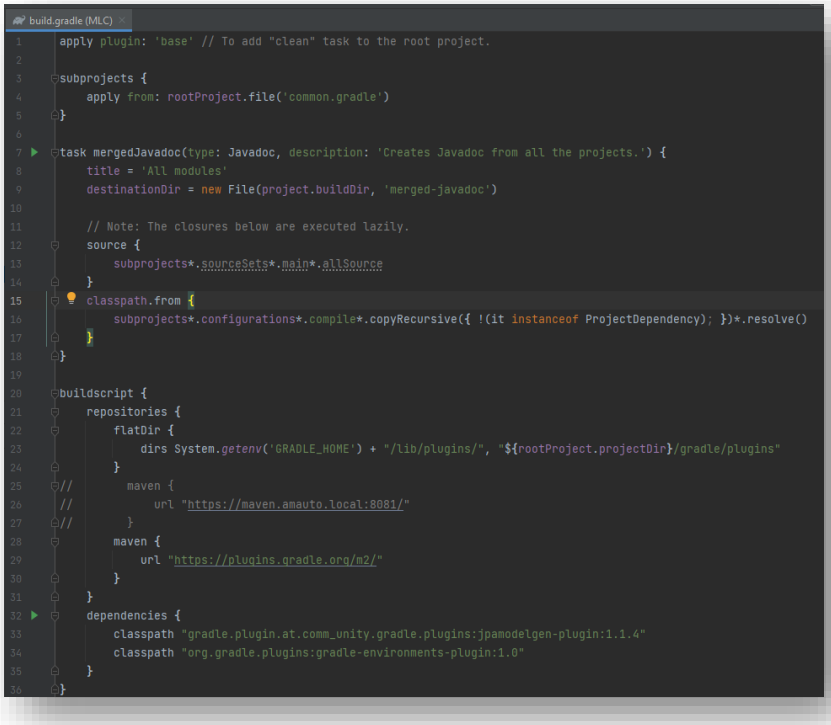


Figura 66 - ARQUIVO BUILD.GRADLE

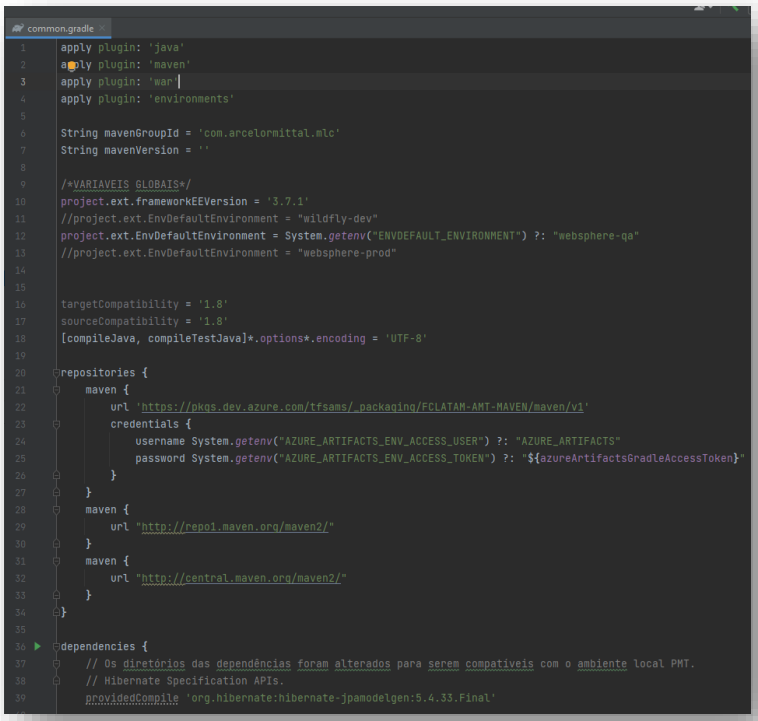


Figura 67 - ARQUIVO COMMON-GRADLE

6.2.2 Criando um subprojeto que estenda do Framework

Neste ponto, supõe-se que o seu projeto raiz já esteja configurado corretamente. Agora é necessário adicionar os módulos do framework à sua aplicação, o que é feito através da criação de subprojetos para o projeto raiz.

É importante adicionar apenas os módulos do framework que são necessários para a sua aplicação. Por exemplo, se a sua aplicação não utiliza OPC, não é necessário criar um subprojeto que estenda o FrameworkOPC.

Para criar um subprojeto para a sua aplicação, siga os seguintes passos:

- Acesse novamente o menu Arquivo (File) e selecione a opção Abrir Projeto (Open Project).
- Selecione o projeto Gradle que deseja abrir, conforme ilustrado na figura abaixo:

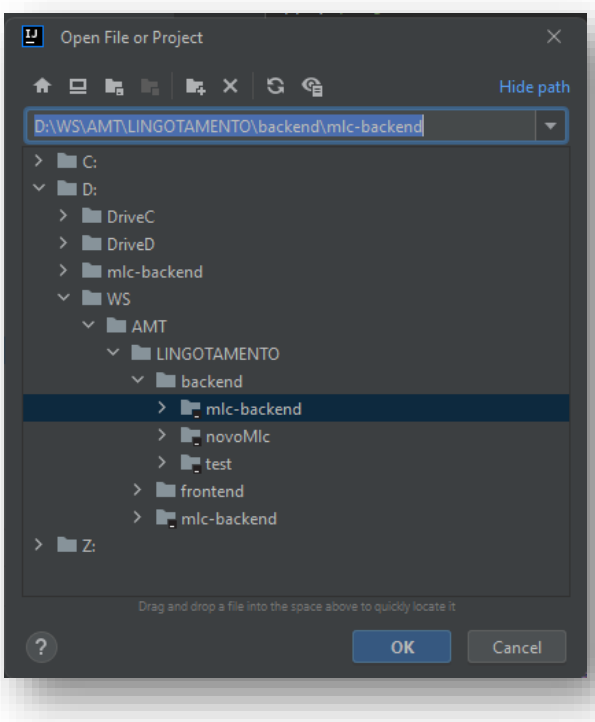


Figura 68 - GRADLE SUBPROJECT

Ao clicar em OK, o IntelliJ exibirá a seguinte tela. Aguarde até que o IntelliJ carregue suas configurações e construa o projeto:

- O nome do seu projeto (por exemplo, MLC1, MLC2) será concatenado com o nome do módulo correspondente (por exemplo, RestApiCore, RecebimentoOMQCore).
- Você também deverá especificar a pasta onde o subprojeto será colocado.
- Essa pasta será a mesma pasta do seu projeto raiz.

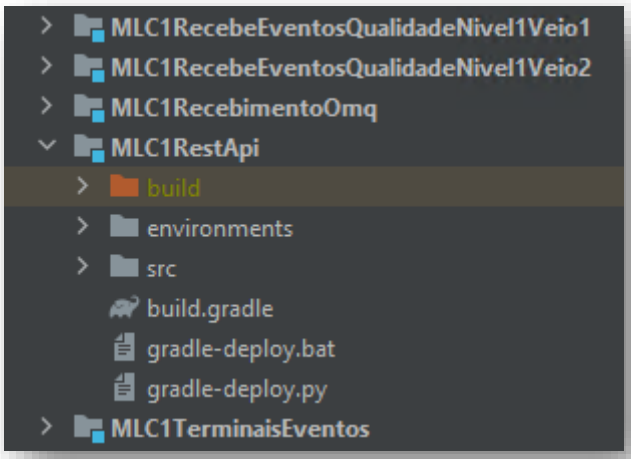


Figura 69 - NOME E LOCALIZAÇÃO GRADLE SUBROJECT

Conforme pode ser observado na imagem acima, o subprojeto foi criado com sucesso dentro do projeto raiz.

- É importante destacar que os arquivos settings.gradle, build.gradle e common.gradle do projeto raiz foram herdados corretamente. Agora, o próximo passo é ajustar o build.gradle do próprio projeto.
- O build.gradle do projeto pode ser copiado integralmente de outro projeto que utilize o mesmo framework equivalente. Por exemplo, na imagem, em que um subprojeto RestApi foi criado, é possível ir para o projeto MLCCore e copiar o seu build.gradle.
- Esse build.gradle do projeto inclui essencialmente o FrameworkCore e o módulo correspondente do framework (RestApiCore, RecebimentoOMQCore, etc.), além de versionar os arquivos necessários para a compilação.

O conteúdo da pasta Source Packages deve ser copiado integralmente de outro projeto que utilize o módulo equivalente do framework. Por exemplo, na imagem em que um subprojeto RestApi foi criado, é possível ir para o projeto MLCRestApiCore e copiar todo o conteúdo da pasta Source Packages.

No *WildFly*, a configuração JNDI pode ser encontrada no arquivo standalone.xml, localizado na pasta configuration dentro da pasta de instalação do *WildFly* (por exemplo, C:*WildFly*-26.1.0.Final\standalone\configuration). Dentro desse arquivo, deve existir uma tag "datasource" que contém a propriedade "jndi-name". É o conteúdo dessa propriedade que deve ser copiado.

No *WebSphere*, a configuração JNDI pode ser encontrada no menu lateral Recursos -> JDBC -> Origens de Dados. Nessa tela, há uma tabela que indica o JNDI criado para cada servidor.

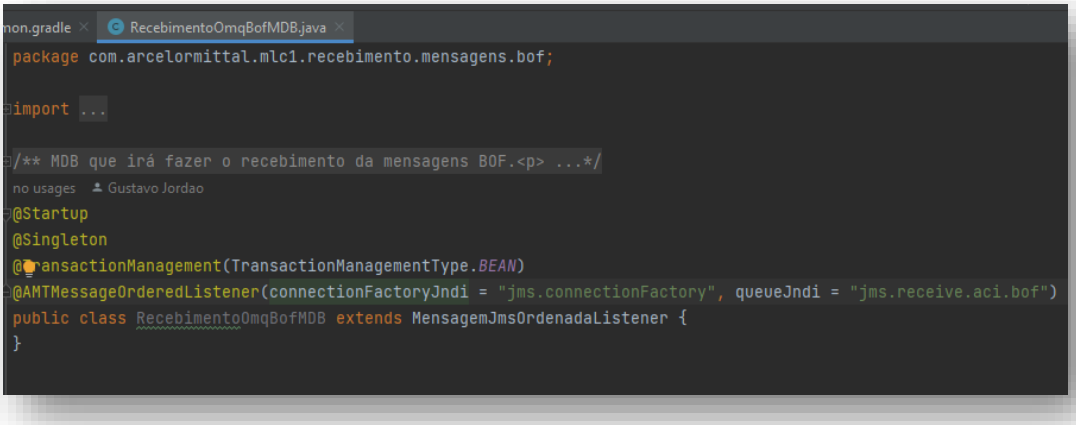


Figura 70 - CAMINHO DO LOCAL ONDE SE ENCONTRA O JNDI DO BANCO DE DADOS

6.2.3 Integração *front-end* e *back-end*

A integração entre o *front-end* e o *back-end* é uma parte fundamental do projeto MLC1, pois permite a comunicação entre o cliente (interface do usuário) e o servidor (lógica de negócio e banco de dados). Nesta seção, veremos como realizar essa integração de forma eficiente.

6.2.3.1 Criando um novo menu

Para inserir um novo menu na barra principal é necessário seguir os passos aqui descrito.

Inserir no arquivo “app.menu.component.ts” da MLC (mlc -> src -> app -> components -> ui -> menu) o novo menu, no exemplo adicionamos o menu “Teste” nas opções de Menu da MLC1.

A visibilidade do menu é alterada de acordo com as permissões de visualização do usuário logado, caso o menu a ser criado não exista dentro das permissões, é necessário adicionar a permissão ao Grupo em que o usuário alvo pertence.

Lembrando ainda, que a ordem dos menus é a ordem em que eles irão aparecer na barra. A fim do exemplo, vamos deixar o campo “visible” com o valor “true” assim o mesmo estará disponível para todos os usuários.

```
// Define as Telas da MLC3
modelMlc3 = [
    {...
    },
    {...
    },
    {...
    },
    {...
    },
    {...
    },
    {...
    },
    {
        label: 'Teste', icon: 'pi pi-question-circle', items: [

        ],
        visible: this.sessionService.hasItem(this.permissions.teste.menuTeste)
    }
];
```

Figura 71 - CONFIGURANDO UM NOVO MENU PARA O SISTEMA

Após compilar a aplicação, já é possível ver o novo menu, conforme a imagem abaixo:

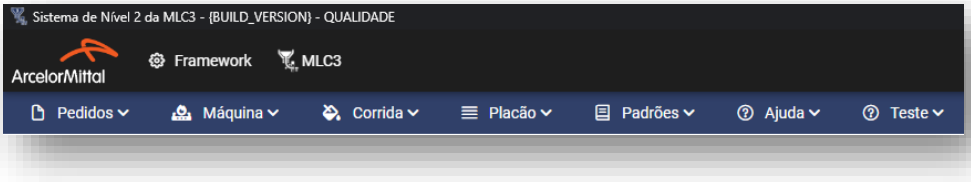


Figura 72 - VISUALIZAÇÃO DE UM NOVO MENU CRIADO

As permissões de visualização de menus e telas devem ser inseridas no arquivo “app.config.permissionsts” no caminho “mlc -> src -> app -> configs”, o serviço “SessionService” verificará se o usuário tem está permissão e exibirá o menu ou tela no caso de positivo.

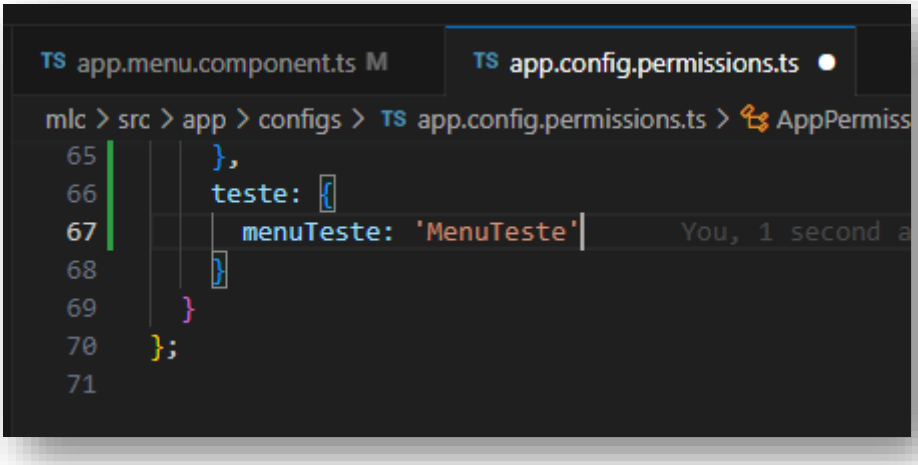


Figura 73 - CONFIGURANDO AS PERMISSÕES DE VISUALIZAÇÃO DE MENUS E TELAS PARA UM USUÁRIO

6.2.3.2 Criando uma nova tela

Para adicionar uma nova tela ao menu que foi criado no item anterior (6.2.4.1), basta adicionar a tela aos "items" do menu no arquivo "app.menu.components.ts", conforme a imagem abaixo:

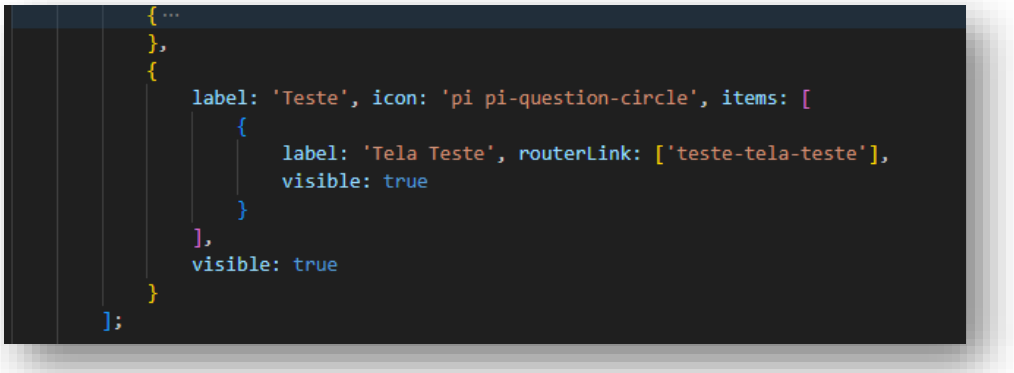


Figura 74 - ADICIONANDO UMA TELA AO MENU

O "label" é o nome que será exibido dentro do Menu, "routerLink" é a rota que será seguida até o component respectivo da Tela e "visible" deve seguir a regra de permissões exibido no item anterior (6.2.4.1).

O componente da "Tela Teste" deve ser criado dentro de sua respectiva pasta no caminho "mlc -> src -> components -> view". Caso a pasta do menu não exista, basta cria-la, assim como a pasta da respectiva tela como na imagem abaixo:

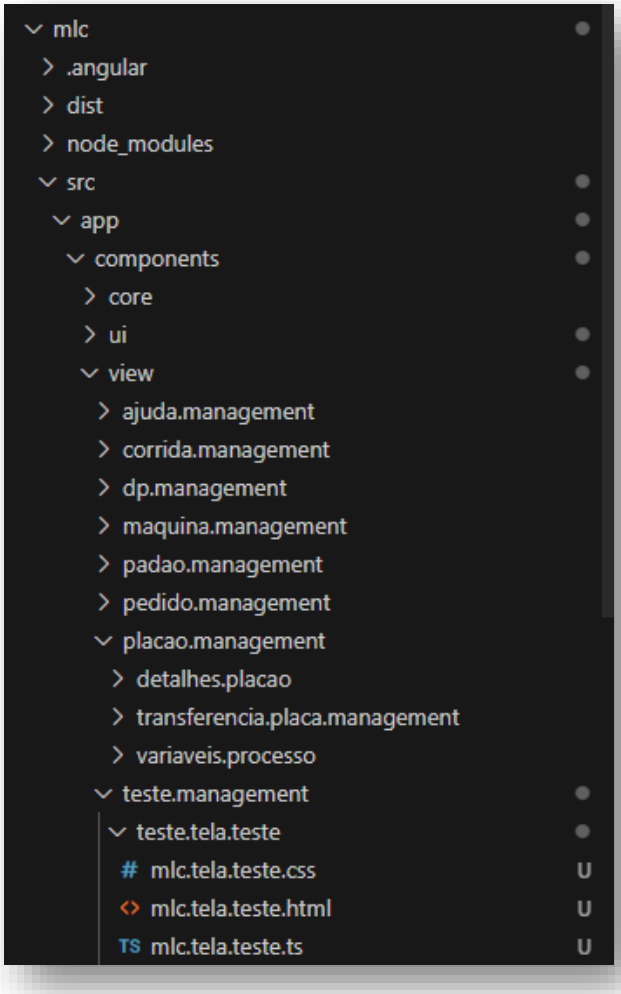


Figura 75 - CRIANDO A PASTA DE UMA NOVA TELA DENTRO DO PROJETO

O arquivo ".ts" da Tela deve conter no mínimo a anotação "@Component" com os respectivos "items":

- **selector**: Nome do component para ser utilizado nas tags html
- **templateUrl**: Nome do arquivo .html que vai conter os elementos da tela
- **styleUrls**: Array com os nomes dos arquivos .css de estilo
- **ngOnInit**: Método chamado assim que a tela é carregada, onde devem ser colocadas instruções a serem executadas sem a necessidade de comando do usuário.

Agora, no "constructor" da Tela, adicione o serviço "PageTitleService" e o chame no método "ngOnInit" para atribuir o título da Tela aberta na barra principal.

```
constructor(
    private pageTitleService: PageTitleService
) {

}

ngOnInit(): void {
    this.pageTitleService.setPageTitle('Tela de Teste');
}
```

Figura 76 - ADICIONANDO UM TÍTULO À TELA

O arquivo ".html" da Tela pode conter vários tipos de "items", os mais utilizados na aplicação são o "header" e o "body".

No "header" normalmente são inseridos os componentes de busca ou elementos próprios da tela, no exemplo foi colocado apenas o título e uma barra de separação.

No body foi adicionado um botão que chamará a função "buscar" no arquivo .ts, conforme imagem abaixo:

```
TS app.menu.component.ts M | TS mlc.tela.teste.ts U | <> mlc.tela.teste.html U ● | <> app.main.component.html | TS app.config.permissions.ts
mlc > src > app > components > view > teste.management > teste.tela.teste > <> mlc.tela.teste.html > amt-mlc-base-view
1  <amt-mlc-base-view>
2      <amt-mlc-base-view-header>
3          <h5>Tela de Testes</h5>
4          <p-divider></p-divider>
5      </amt-mlc-base-view-header>
6      <amt-mlc-base-view-body>
7          <button (click)="buscar()" pButton pRipple type="button" label="Buscar" icon="pi pi-check" class="p-button-success mr-2"></button>
8      </amt-mlc-base-view-body>
9  </amt-mlc-base-view>
```

Figura 77 - ADICIONANDO UM BOTÃO À TELA

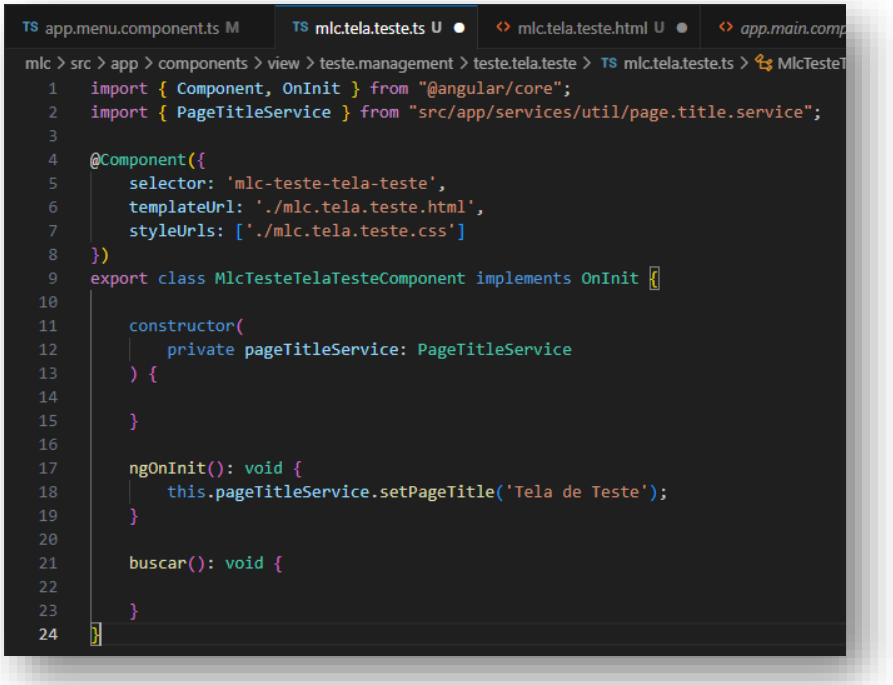


Figura 78 - ADICIONANDO UMA FUNÇÃO A UM COMPONENTE DE UMA TELA

No arquivo “app.routing.module.ts” no caminho “shell -> src -> app -> router” adicione a rota referente a tela criada às MLC’s onde devem estar disponíveis, conforme a imagem abaixo:

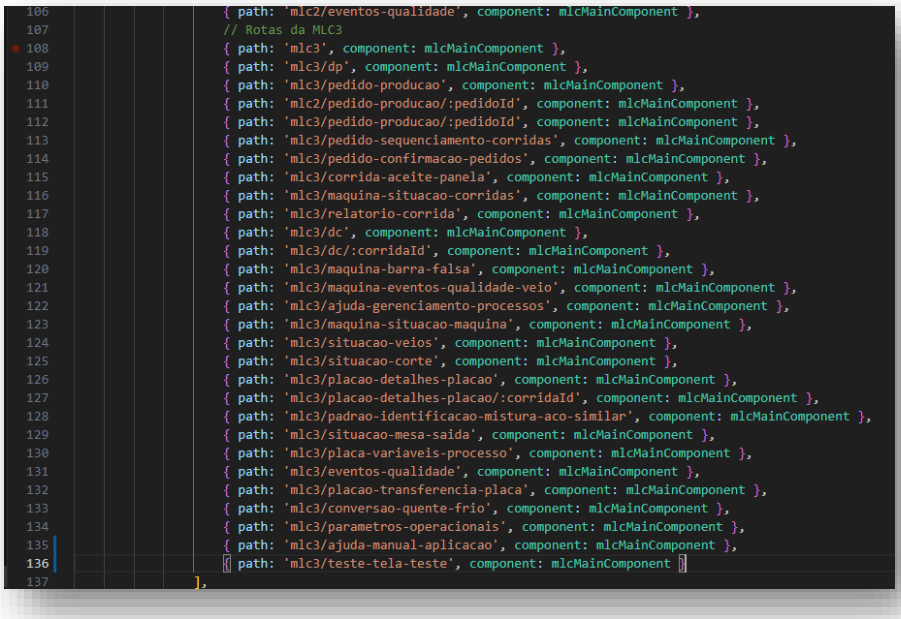


Figura 79 - ADICIONANDO A ROTA REFERENTE À UMA TELA

No arquivo “app.routing.module.ts” no caminho “mlc -> src -> app -> router” adicione a rota, componente e permissão da tela criada às MLC’s onde devem estar disponíveis, conforme a imagem abaixo:

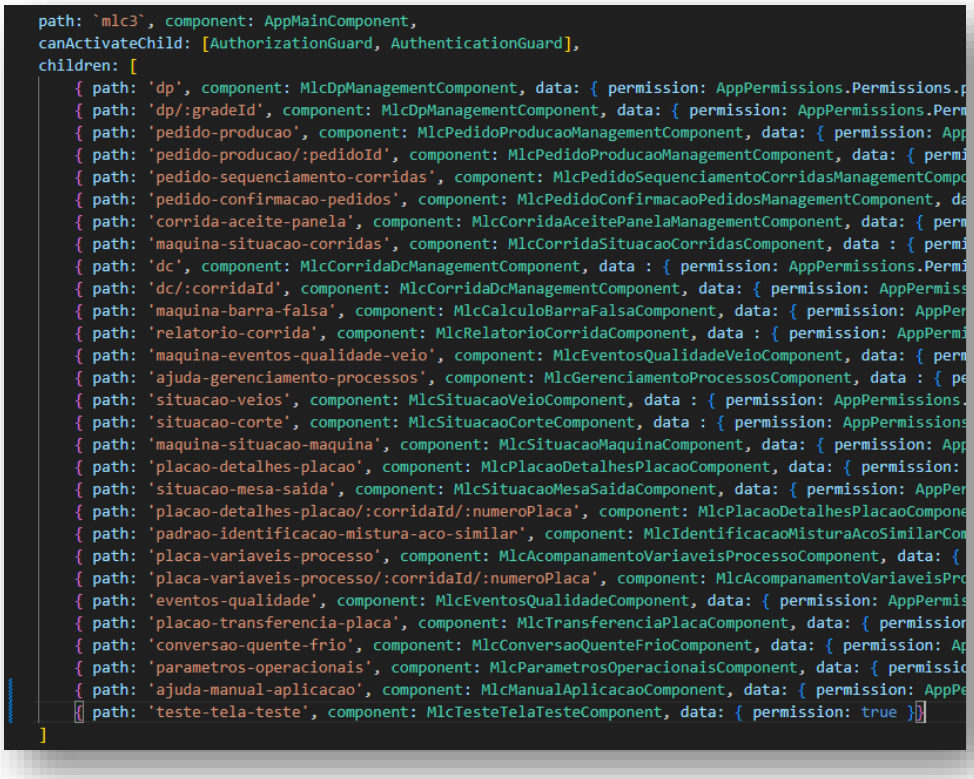


Figura 80 - ADICIONANDO ROTA, COMPONENTE E PERMISSÃO REFERENTES À UMA TELA

Adicione também a tela à lista de componentes da aplicação no arquivo “app.mlc.module.ts” no caminho “mlc -> src -> app -> modules”.

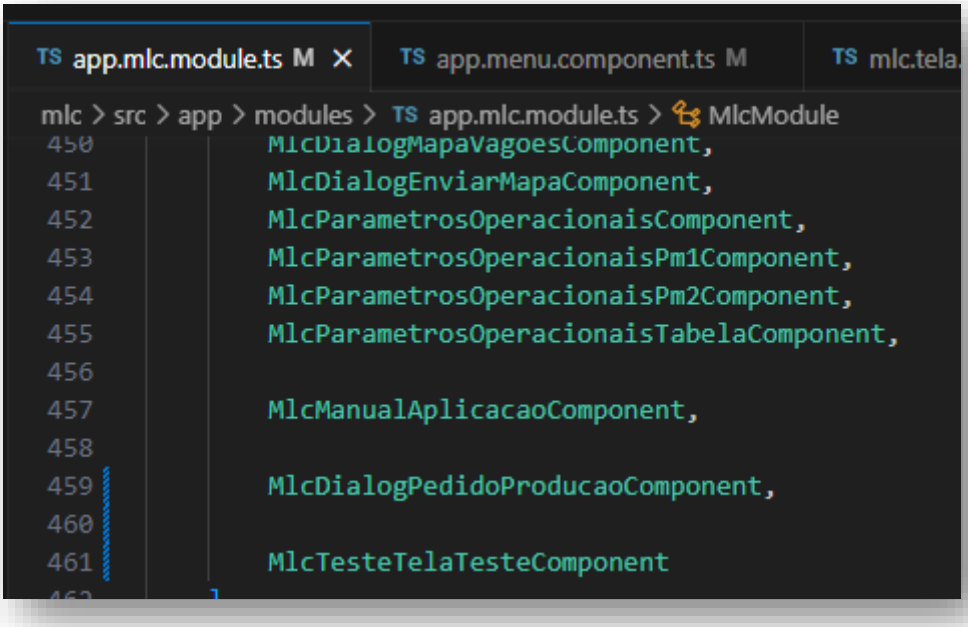


Figura 81 - ADICIONANDO UMA TELA À LISTA DE COMPONENTES DA APLICAÇÃO

Após compilar a aplicação, a tela estará disponível no menu indicado.

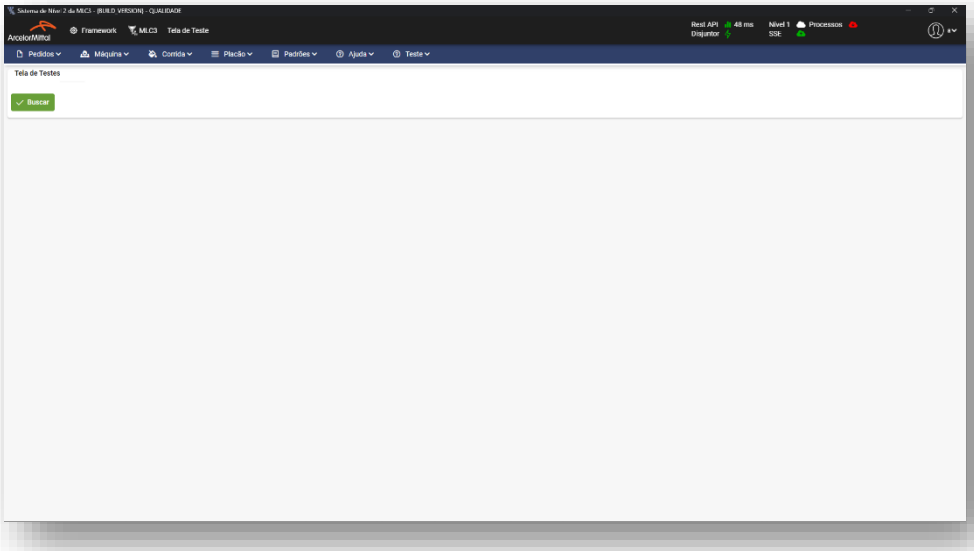


Figura 82 - TELA DISPONIBILIZADA NO MENU INDICADO

Crie um pacote para colocar os seus controllers (com.arcelormittal.mlc.rest.api.services) em nosso projeto, como mencionado anteriormente, as controllers são chamadas aqui, de services, ou

utilize o pacote controllers já existente em MLCCore. Dentro deste pacote, crie sua classe controller (Ex.: CorridaService.java).

Deve ser adicionado ao cabeçalho da sua classe 2 anotações:

- @Path("/"): Indica que todos os caminhos de serviço que apontam para sua classe começam na raiz da URI.
  - Ex.: http://localhost:8080/MLC1/corridas/;
- @Stateless: torna sua classe Stateless (Isso te permite, por exemplo, injetar recursos na sua classe);

Figura 83 - ADICIONANDO CABEÇALHO NO CONTROLLER

Nosso controller deve interagir com o banco através de consultas, updates, commits. Então é necessário injetar nele um DomainStore, que nos permite fazer justamente isso.

6.2.3.2 Utilizando um serviço e exibindo dados na Tela

Os serviços existentes na aplicação são encontrados na pasta "services" no caminho "mlc -> src -> services", onde estão divididos nas seguintes categorias: "handlers", "http", "mlc", "security" e "útil".

O serviço que utilizaremos no exemplo é o "corrida.service.ts" no caminho "mlc -> src -> services -> mlc -> corrida".

Para utilizar o serviço na tela, é necessário declará-lo no "constructor" da tela.

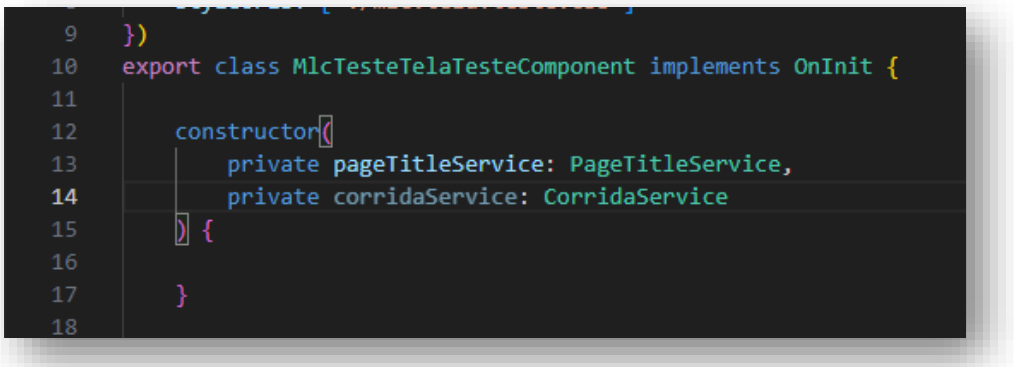


Figura 84 - DECLARANDO UM SERVIÇO NO "CONSTRUCTOR" DE UMA TELA PARA UTILIZAÇÃO

No exemplo, ao clicar no botão "Buscar" na tela de exemplo, vamos chamar o método "getCorrenteAnteriorInboardOutboard" e exibir na tela os números dessas corridas.

No método "buscar" que é o método chamado ao clicar no botão, realizamos a chamada, nos inscrevemos na requisição e aguardamos uma resposta do tipo "CorridaModel[]".

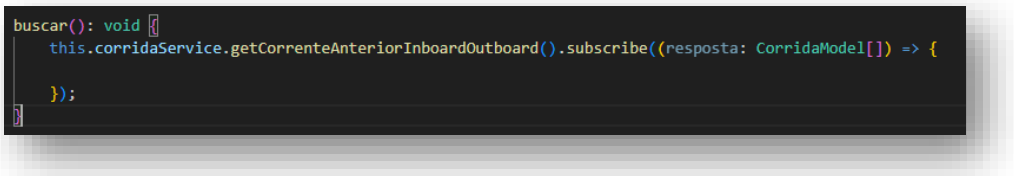


Figura 85 - CHAMANDO UM MÉTODO DE UM SERVIÇO

Adicionamos uma variável que irá "segurar" os dados recebidos no corpo da classe, conforme a imagem abaixo:

Figura 86 - DEFININDO UMA VARIÁVEL PARA RECEBER OS DADOS DE UMA REQUISIÇÃO

E atribuímos o valor recebido a está variável:

```
    buscar(): void {
        this.corridaService.getCorrenteAnteriorInboardOutboard().subscribe((resposta: CorridaModel[]) => {
            this.corridas = resposta;
        });
    }
}
```

Figura 87 - ATRIBUINDO VALOR À VARIÁVEL COM DADOS RECEBIDOS DE UMA REQUISIÇÃO

Existem diversas formas de exibir os dados recebidos na tela, e para o exemplo, iremos utilizar o ***ngFor** do Angular, como na imagem abaixo:

```
<amt-mlc-base-view-body>
    <button (click)="buscar()" pButton pRipple type="button"
    <div *ngFor="let corrida of corridas">
        <span>{{corrida.id}}</span>
    </div>
</amt-mlc-base-view-body>
```

*Figura 88 - UTILIZANDO O *NGFOR PARA EXIBIÇÃO DE DADOS NA TELA*

Agora basta compilar a aplicação, e podemos ver que após realizar a consulta clicando no botão “Buscar” os “Ids” das corridas são exibidos na tela.

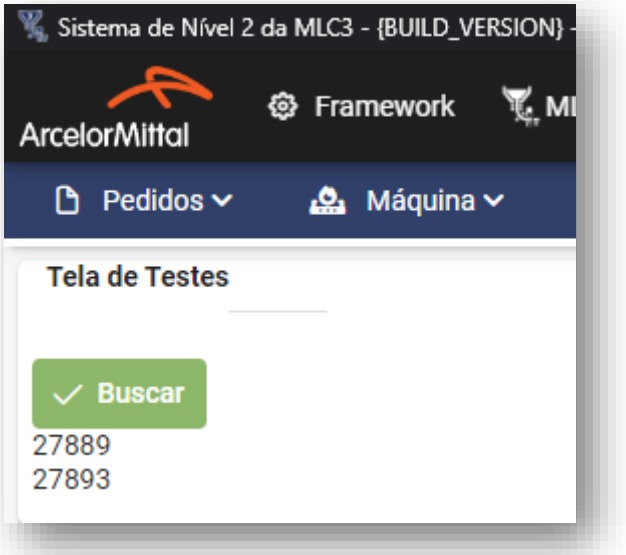


Figura 89 - DADOS EXIBIDOS NA TELA APÓS CONSULTA

6.2.3.3 Adicionando um novo Contexto

Na aplicação, Contextos são o "caminho" base para realizar uma requisição no *back-end*, e são utilizados na criação dos serviços, no exemplo anterior, utilizamos o "CorridaService" que utiliza o "CorridaContext" para montar as requisições, conforme imagem abaixo:

```
getCorridasDefaultAceite(): Subscribable<CorridaModel[]> {
    return this.http.get(`${EnvironmentConfig.corridaContext}buscarcorridapadraoaceite`);
}

getUltimaAceite(disableLoading?: boolean): Observable<CorridaModel> {
    return this.http.get(`${EnvironmentConfig.corridaContext}ultimaaceite`, disableLoading) as Observable<CorridaModel>;
}

getCorrenteAnteriorInboardOutboard(disableLoading?: boolean): Observable<CorridaModel[]> {
    return this.http.get(`${EnvironmentConfig.corridaContext}correnteanteriorinboardoutboardnomolde`, disableLoading) as Observable<CorridaModel[]>;
}

getInboardOutboard(disableLoading?: boolean): Observable<CorridaModel[]> {
    return this.http.get(`${EnvironmentConfig.corridaContext}inboardoutboard`, disableLoading) as Observable<CorridaModel[]>;
}

getCorrenteAnterior(disableLoading?: boolean): Observable<CorridaModel[]> {
    return this.http.get(`${EnvironmentConfig.corridaContext}correnteanterior`, disableLoading) as Observable<CorridaModel[]>;
}
```

Figura 90 - UTILIZANDO UM CONTEXTO

Para adicionar um novo Contexto, é necessário seguir os seguintes passos:

- No Arquivo "app.config.environment.ts" no caminho "shell -> src -> app -> configs" adicione uma variável do tipo "string" com o nome do novo contexto acrescido de "_mlc" no começo do nome, e atribua seu valor.

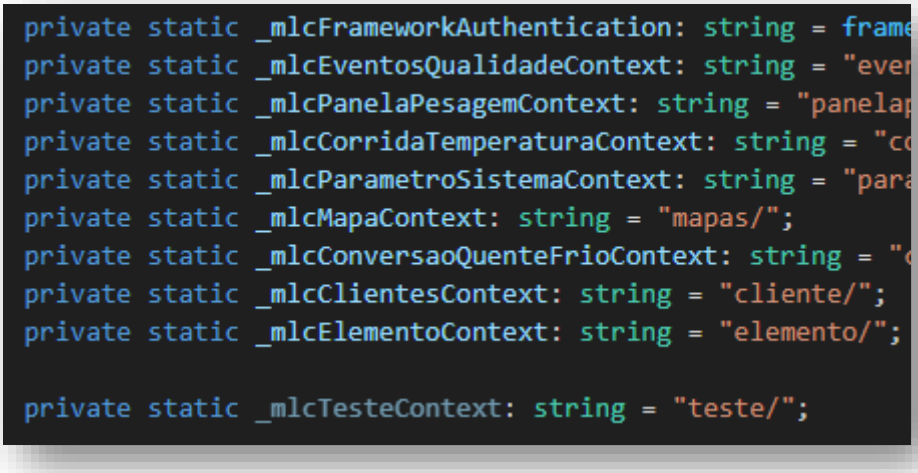


Figura 91 - – CRIANDO UM NOVO CONTEXTO

- No mesmo arquivo, crie dois métodos, um “get” e um “set”, para o novo Contexto.

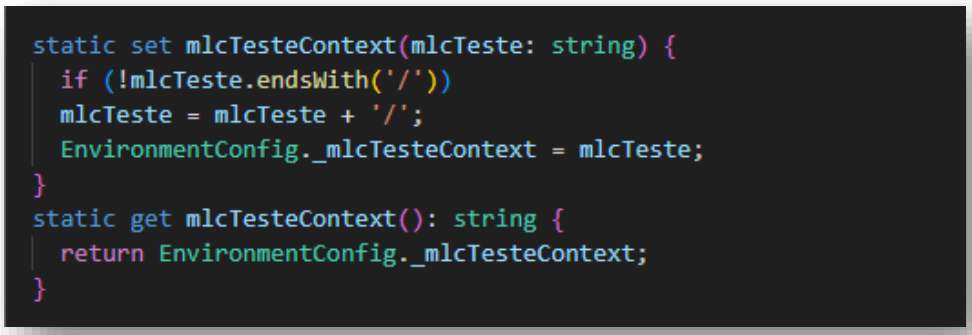


Figura 92 - CRIANDO MÉTODOS PARA UM CONTEXTO

- No arquivo “main.component.ts” no caminho “shell -> src -> app -> components -> view -> apps -> mlc” adicione um “element” com o novo contexto e o atribua com a variável que criamos no arquivo anterior.

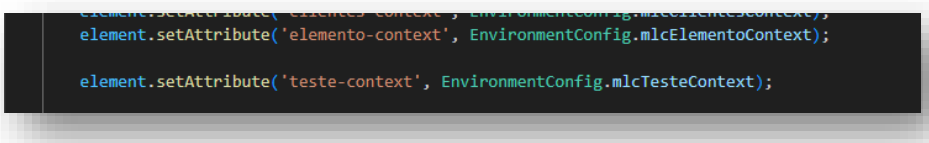


Figura 93 - ADICIONANDO UM “ELEMENT” COM O NOVO CONTEXTO

- No arquivo “app.config.environment.ts” no caminho “mlc -> src -> app -> configs” adicione o contexto, assim como os métodos “get” e “set”.

```
private static _portaRestApi: string = "";

private static _testeContext: string = "";

static set testeContext(testeContext: string) {
  if (!testeContext.endsWith('/'))
  testeContext = testeContext + '/';
  EnvironmentConfig._testeContext = testeContext;
}
static get testeContext(): string {
  return EnvironmentConfig.serverPath + EnvironmentConfig._testeContext;
}
```

Figura 94 - ADICIONANDO O NOVO CONTEXTO NO “ENVIRONMENT”

- Por fim, no arquivo “app.component.ts” no caminho “mlc -> src -> app” adicione um “@Input” do novo Contexto.

```
@Input()
portaRestOmq = "";
@Input()
portaRestApi = "";

@Input()
testeContext = "";
```

Figura 95 - ADICIONANDO O NOVO CONTEXTO NO “COMPONENT”

- No mesmo arquivo, no método “ngOnInit”, atribua o valor do novo Contexto como na imagem abaixo:

```
ngOnInit() {
    this.translateService.setDefaultLang('pt-br');
    this.primengConfig.ripple = true;
    this.router.initialNavigation();
    this.routerCorrection.listemCorrection();
    EnvironmentConfig.testeContext = this.testeContext;
```

Figura 96 - ATRIBUINDO O VALOR DO NOVO CONTEXTO NO "NGONINIT"

Após compilar a aplicação, o novo contexto pode ser utilizado.

6.2.3.3 Criando um novo Serviço

Serviços são utilizados na aplicação para enviar, receber ou processar dados. Neste exemplo, iremos criar um serviço utilitário simples.

Crie um arquivo chamado "teste.service.ts" no caminho "mlc -> src -> app -> services -> util".

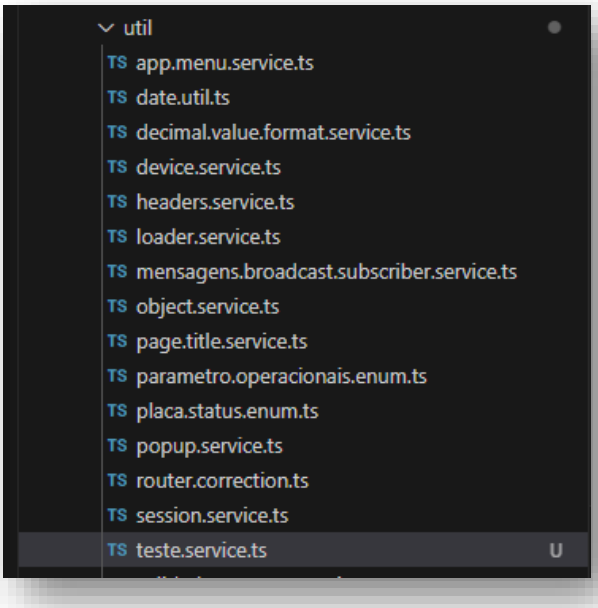


Figura 97 - CRIANDO UM NOVO SERVIÇO

Os serviços utilizam a injeção de dependência do Angular para serem utilizados nas telas ou componentes. O serviço se torna um “Singleton” fazendo que seja o mesmo a mesma instância em qualquer lugar da aplicação.

Figura 98 - INJEÇÃO DE DEPENDÊNCIA DO ANGULAR PARA UM NOVO SERVIÇO

Adicione no serviço suas funções, no exemplo, foi adicionado um Método para exibir uma mensagem de alerta para o usuário, que para exemplo, vai exibir o valor do Contexto que criamos no exemplo anterior.

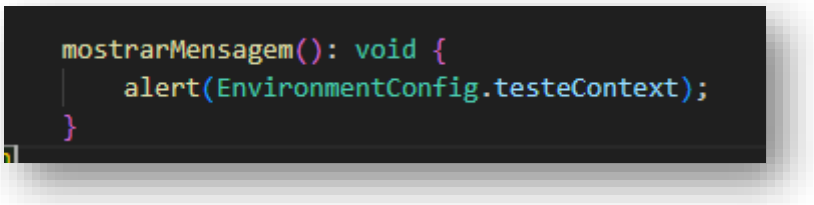


Figura 99 - ADICIONANDO FUNÇÕES A UM SERVIÇO

Agora é preciso registrar o serviço criado antes de poder ser utilizado na aplicação, para isso abra o arquivo “app.mlc.module.ts” no caminho “mlc -> src -> app -> modules” e registre o serviço na variável “provider”, não se esqueça de adicionar sua importação, como nas imagens abaixo:

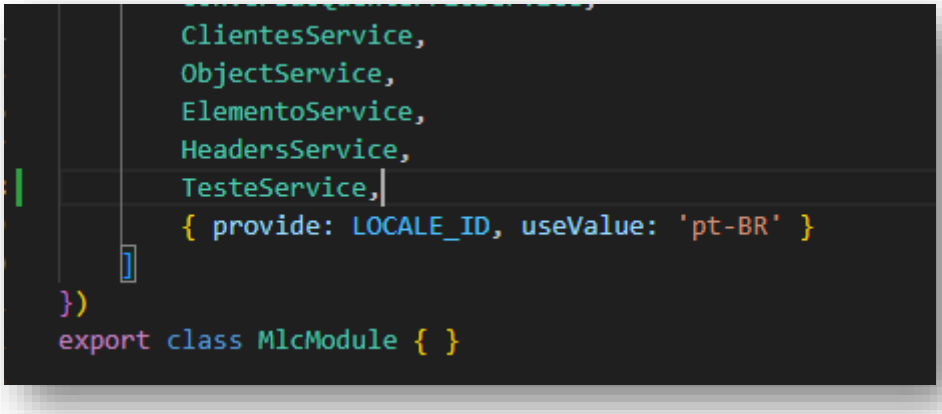


Figura 100 - REGISTRANDO UM SERVIÇO CRIADO

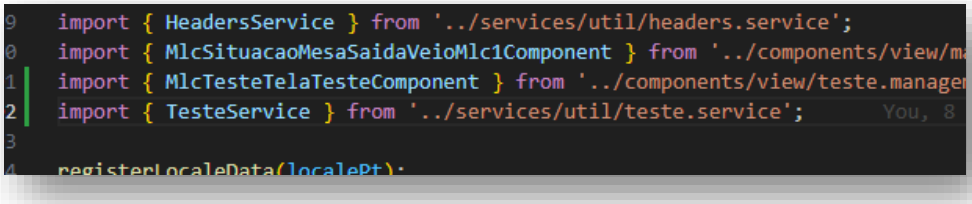


Figura 101 - ADICIONANDO A IMPORTAÇÃO DO NOVO SERVIÇO

Na Tela de teste que criamos em um exemplo anterior, adicione o serviço que foi criado no “constructor” da tela, e ao clicar no botão “Buscar”.

```typescript
import { Component, OnInit } from "@angular/core";
import { CorridaModel } from "src/app/models/mlc/corrida/corrida.model";
import { CorridaService } from "src/app/services/mlc/corrida/corrida.service";
import { PageTitleService } from "src/app/services/util/page.title.service";
import { TesteService } from "src/app/services/util/teste.service";

@Component({
    selector: 'mlc-teste-tela-teste',
    templateUrl: './mlc.tela.teste.html',
    styleUrls: ['./mlc.tela.teste.css']
})
export class MlcTesteTelaTesteComponent implements OnInit {

    corridas: CorridaModel[] = [];

    constructor(
        private pageTitleService: PageTitleService,
        private corridaService: CorridaService,
        private testeService: TesteService
    ) {

    }

    ngOnInit(): void {
        this.pageTitleService.setPageTitle('Tela de Teste');
    }

    buscar(): void {
        this.testeService.mostrarMensagem();
        this.corridaService.getCorrenteAnteriorInboardOutboard().subscribe((resposta: CorridaModel[]) => {
            this.corridas = resposta;
        });
    }
}
```

Figura 102 - ADICIONANDO UM SERVIÇO NO "CONSTRUCTOR" DE UMA TELA

Após compilar a aplicação, é possível ver que o Serviço e o Contexto estão funcionando corretamente.

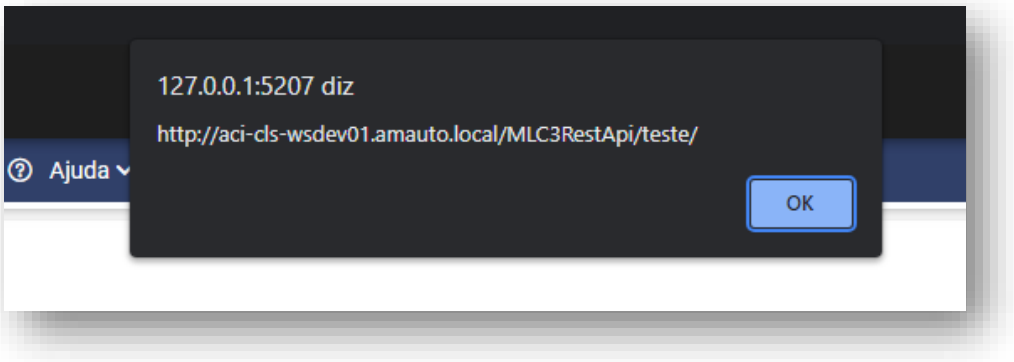


Figura 103 - VERIFICAÇÃO DO SERVIÇO E CONTEXTO CRIADOS

6.2.3.4 Adicionando uma variável de ambiente

As principais variáveis de ambiente são as URL's utilizadas nas comunicações REST, assim como as portas de determinados serviços.

Para modificar uma variável basta acessar o arquivo “mlcX.environment.json” no caminho “shell -> src -> assets -> config -> AMBIENTE* -> MLC**” da MLC desejada e alterar seu valor, onde:

- **AMBIENTE*** é onde a aplicação será utilizada, que podem ser: “dev”, “qa”, “local” ou “prod”
- **MLC**** é a respectiva MLC onde a variável será alterada.

Para adicionar uma nova variável, abra o arquivo “mlcX.environment.json” no caminho “shell > src > assets > config > AMBIENTE* > MLC**”, no caso da adição de nova variável, ela precisa ser adicionada aos arquivos de configuração de **TODAS** as MLCs.

Para o exemplo, vamos adicionar uma variável “teste” que terá o valor “teste123”, conforme a imagem abaixo:

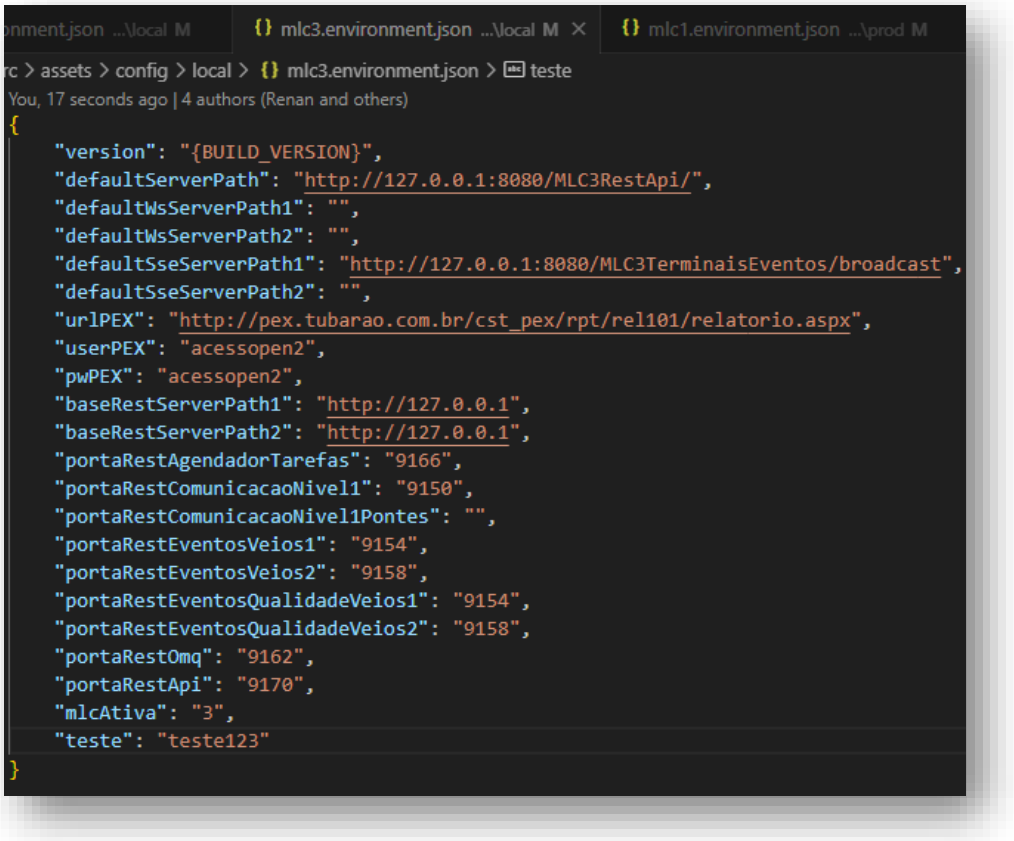


Figura 104 - ADICIONANDO UMA NOVA VARIÁVEL DE AMBIENTE

O arquivo “environment.ts” no caminho “shell -> src -> environments” também deve ter a nova variável adicionada, assim como todos os environments de todas as MLCs, neste caso, não é para ser atribuído nenhum valor.

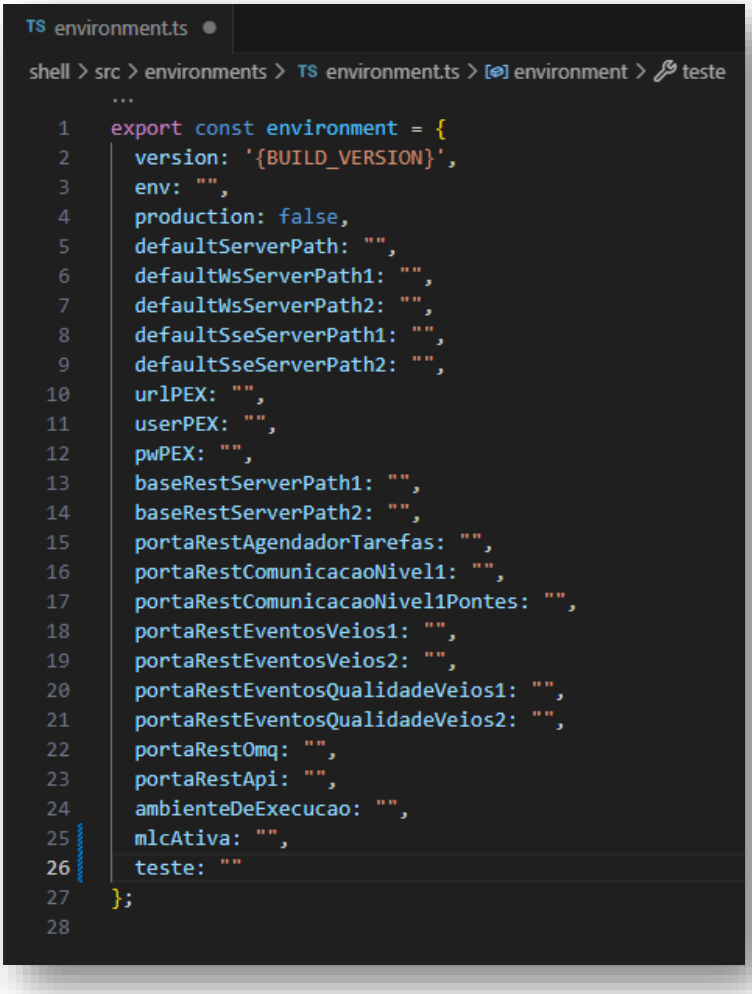


Figura 105 - ADICIONANDO NOVA VARIÁVEL NO "ENVIRONMENT"

No mesmo caminho, as pastas "MLC1", "MLC2" e "MLC3" contém os arquivos de Environment das respectivas MLCs, e os mesmas também devem ter a nova variável.

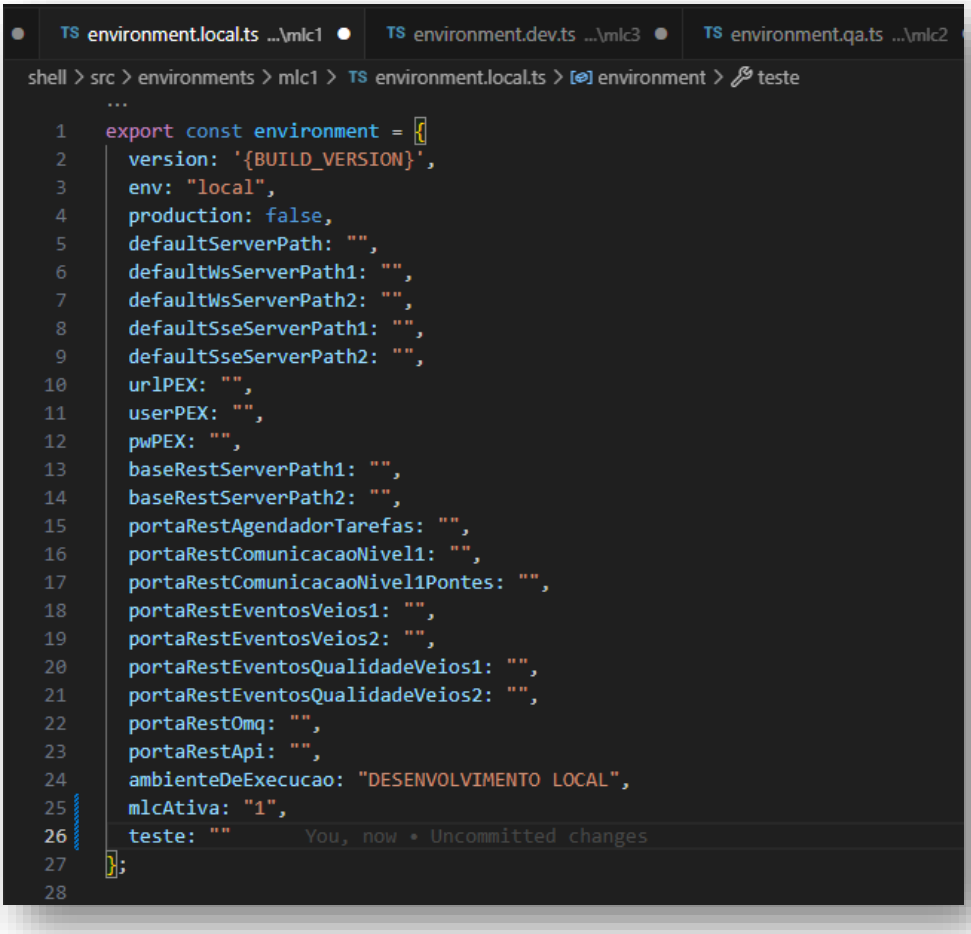


Figura 106 - ADICIONANDO NOVA VARIÁVEL NO "ENVIRONMENT" DE TODAS AS MLC'S

No Arquivo "app.config.environment.ts" no caminho "shell -> src -> app -> configs" adicione a nova variável assim como os métodos "get" e "set".

```
private static _pwPEX: string = environment.pwPEX;

private static _teste: string = environment.teste;

static set teste(teste: string) {
  EnvironmentConfig._teste = teste;
}
static get teste(): string {
  return EnvironmentConfig._teste;
}
```

Figura 107 - ADICIONANDO OS MÉTODOS GET E SET DA NOVA VARIÁVEL

No Arquivo “config.environment.ts” no caminho “shell -> src -> app -> models” adicione a nova variável.

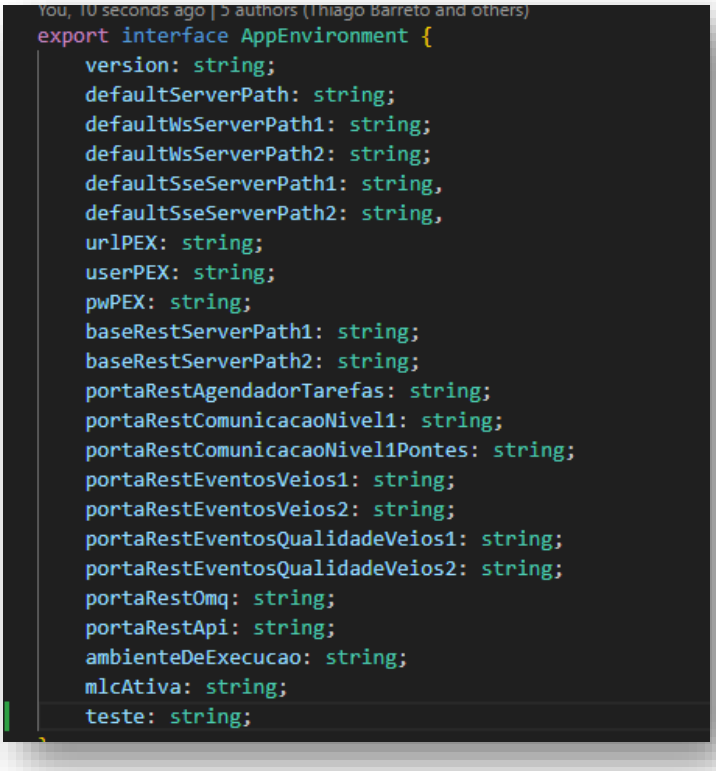


Figura 108 - ADICIONANDO NOVA VARIÁVEL EM “MODELS”

Por fim, no Arquivo “app.environment.service.ts” no caminho “shell > src > app > services > config” adicone a nova variável.

```
public init(): Promise<boolean> {
  return new Promise<boolean>((resolve, reject) => {
    this.http.get(`assets/config/${environment.env}/mlc${environment.mlcAtiva}.environment.json`).subscribe(
      EnvironmentConfig.version = response.version;
      EnvironmentConfig.urlPEX = response.urlPEX;
      EnvironmentConfig.userPEX = response.userPEX;
      EnvironmentConfig.pwPEX = response.pwPEX;
      EnvironmentConfig.serverPath = response.defaultServerPath;
      EnvironmentConfig.wsServerPath1 = response.defaultWsServerPath1;
      EnvironmentConfig.wsServerPath2 = response.defaultWsServerPath2;
      EnvironmentConfig.restSsePath1 = response.defaultSseServerPath1;
      EnvironmentConfig.restSsePath2 = response.defaultSseServerPath2;
      EnvironmentConfig.baseRestServerPath1 = response.baseRestServerPath1;
      EnvironmentConfig.baseRestServerPath2 = response.baseRestServerPath2;
      EnvironmentConfig.portaRestAgendadorTarefas = response.portaRestAgendadorTarefas;
      EnvironmentConfig.portaRestComunicacaoNivel1 = response.portaRestComunicacaoNivel1;
      EnvironmentConfig.portaRestComunicacaoNivel1Pontes = response.portaRestComunicacaoNivel1Pontes;
      EnvironmentConfig.portaRestEventosVeios1 = response.portaRestEventosVeios1;
      EnvironmentConfig.portaRestEventosVeios2 = response.portaRestEventosVeios2;
      EnvironmentConfig.portaRestEventosQualidadeVeios1 = response.portaRestEventosQualidadeVeios1;
      EnvironmentConfig.portaRestEventosQualidadeVeios2 = response.portaRestEventosQualidadeVeios2;
      EnvironmentConfig.portaRestOmq = response.portaRestOmq;
      EnvironmentConfig.portaRestApi = response.portaRestApi;
      EnvironmentConfig.mlcAtiva = response.mlcAtiva;
      EnvironmentConfig.teste = response.teste;
      resolve(true);
    });
  })
```

Figura 109 - ADICIONANDO NOVA VARIÁVEL NO “SERVICES”

Após compilar a aplicação, pressione a tecla F12 no navegador, mude para a aba “network” e pressione “ctrl + F5”, selecione na lista o arquivo “mlcX.environment.json” onde já é possível verificar a existência da nova variável, já podendo ser utilizada pela aplicação.

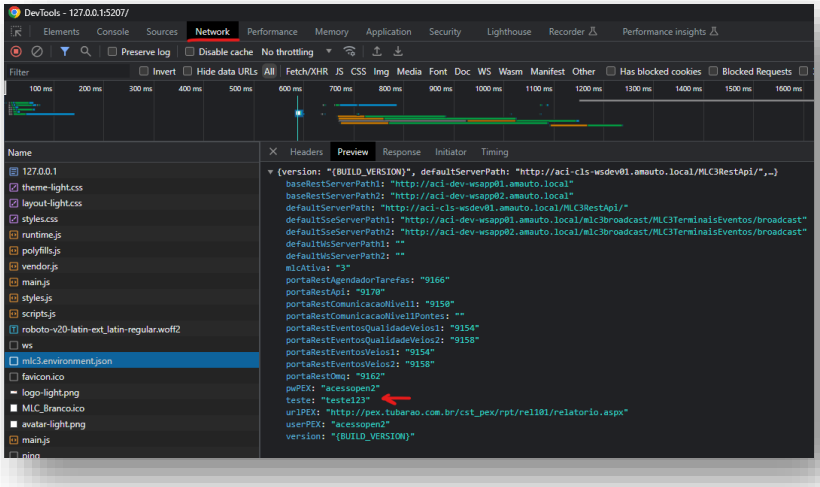


Figura 110 - VERIFICANDO A EXISTÊNCIA DA NOVA VARIÁVEL ATRAVÉS DA FERRAMENTA DO DESENVOLVEDOR DO GOOGLE CHROME

7 Descritivo Funcional

O Descritivo Funcional do Sistema de Lingotamento Contínuo da Máquina 1 (MLC1) é um documento detalhado que descreve as principais funcionalidades e interações do sistema, desde o aço líquido até o lingote final. Baseado em todas as informações descritas até aqui e no conhecimento geral do processo, apresentamos abaixo uma versão resumida do Descritivo Funcional. O Sistema *Back-end* MLC1 é responsável por processar e gerenciar todas as funcionalidades e lógicas de negócio do sistema. Ele é dividido em várias partes, cada uma com suas respectivas responsabilidades, relacionadas e descritas abaixo, conforme seu contexto.

Introdução e Visão Geral

O sistema de Lingotamento Contínuo da Máquina 1 (MLC1) é uma tecnologia avançada utilizada na produção de placas de aço. Ele permite a solidificação contínua do aço líquido, eliminando etapas intermediárias e proporcionando vantagens como menor custo operacional, menor consumo de energia e melhores condições de trabalho.

Escopo do Sistema

O sistema MLC1 é composto, entre outros, pelos seguintes módulos:

- Grades (*Back-end* e *Front-end*)
- Pedidos (*Back-end* e *Front-end*)
- Corrida (*Back-end* e *Front-end*)
- Panela (*Back-end*)
- Otimização (*Back-end*)
- Distribuidor (*Back-end*)
- Molde (*Back-end* e *Front-end*)
- Veios (*Back-end* e *Front-end*)
- Placas (*Back-end* e *Front-end*)
- Mesa de Saída (*Back-end* e *Front-end*)
- Qualidade (*Back-end* e *Front-end*)
- Parâmetros (*Back-end* e *Front-end*)

Requisitos Funcionais

Grades (*Back-end* e *Front-end*):

1. Recebem e gerenciam os pedidos dos clientes.
2. Realizam o acompanhamento da produção e fornecem informações em tempo real.

Pedidos (*Back-end* e *Front-end*):

1. Gerenciam os pedidos recebidos dos clientes.
2. Permitem a criação, edição e exclusão de pedidos.
3. Associam os pedidos aos lingotes produzidos.

Corrida (*Back-end* e *Front-end*):

1. Realiza o planejamento e programação da produção.
2. Define a sequência de lingotamento e a programação de trocas de panelas.

Panela (*Back-end*):

1. Recebe a panela de aço líquido do convertedor.
2. Purifica o aço, removendo inclusões e adicionando ligas.

Otimização (*Back-end*):

1. Realiza a otimização da produção, buscando maximizar a eficiência e qualidade.

Distribuidor (*Back-end*):

1. Recebe o aço líquido da panela e o distribui aos moldes e veios.
2. Evita a contaminação do aço e adiciona elementos de liga.

Molde (*Back-end* e *Front-end*):

1. Promove a solidificação inicial do aço, formando a placa.
2. Controla a largura e espessura da placa.
3. Utiliza oscilação para evitar o colamento do molde à placa.

Veios (*Back-end* e *Front-end*):

1. Recebem o aço líquido do distribuidor e o direcionam para a formação de placas.

Placas (*Back-end* e *Front-end*):

1. Recebem o aço líquido solidificado dos moldes e veios.
2. Realizam o resfriamento primário e secundário.
3. São cortadas na máquina de corte de acordo com as especificações do pedido.

Mesa de Saída (*Back-end* e *Front-end*):

1. Recebe as placas cortadas da máquina de corte.
2. Realiza a inspeção final e o controle de qualidade das placas.
3. Encaminha as placas para o armazenamento ou transporte.

Qualidade (*Back-end* e *Front-end*):

1. Realiza o controle de qualidade durante todo o processo de lingotamento.
2. Coleta e analisa dados sobre a composição e qualidade das placas.

Parâmetros (*Back-end* e *Front-end*):

1. Gerencia os parâmetros operacionais do sistema de lingotamento.
2. Permite a configuração e ajuste de variáveis como velocidade de lingotamento, temperatura, entre outros.

Descrição Detalhada das Funcionalidades

APIs e Comportamentos Esperados:

As APIs do Sistema *Back-end* MLC1 são projetadas para fornecer um conjunto de funcionalidades bem definidas e documentadas. Elas possuem métodos e endpoints específicos para realizar operações como criação de pedidos, programação das mesas de saída, controle de qualidade, gerenciamento de placas, entre outras. As APIs também são projetadas para serem seguras, exigindo autenticação e autorização adequadas para acessar determinadas funcionalidades.

Fluxo de Dados e Eventos:

O Sistema *Back-end* MLC1 possui um fluxo de dados bem definido, onde as informações sobre pedidos, placas, corridas, panelas, veios e demais componentes são armazenadas em bancos de dados. Os eventos, como a alteração do estado de uma placa, a conclusão de uma corrida ou a disponibilidade de uma mesa de saída, são comunicados por meio do MQ para que as partes relevantes do sistema sejam notificadas e tomem as ações apropriadas.

Escalabilidade e Performance:

O Sistema *Back-end* MLC1 é projetado para ser escalável, capaz de lidar com um grande volume de dados e processamento simultâneo de várias operações. Ele é otimizado para garantir uma alta performance, permitindo que as funcionalidades do sistema sejam executadas de forma eficiente e rápida.

Grades (*Back-end* e *Front-end*):

- Funcionalidade 1: Receber os pedidos de produção enviados pelo módulo de Pedidos *Front-end*.
- Funcionalidade 2: Validar e verificar a viabilidade dos pedidos em relação à capacidade de produção.
- Funcionalidade 3: Encaminhar os dados dos pedidos ao módulo de Otimização para otimizar o processo de lingotamento.
- Funcionalidade 4: Monitorar o status da produção em tempo real e atualizar o módulo de Pedidos *Front-end* conforme necessário.

Pedidos (*Back-end* e *Front-end*):

- Funcionalidade 1: Receber e processar as solicitações de produção de lingotes do *Front-end*.
- Funcionalidade 2: Registrar e armazenar os detalhes de cada pedido, como dimensões e especificações do lingote.
- Funcionalidade 3: Enviar os pedidos aos módulos correspondentes para iniciar a produção.
- Funcionalidade 4: Permitir a consulta do status de produção e detalhes dos pedidos em andamento.

Corrida (*Back-end* e *Front-end*):

- Funcionalidade 1: Controlar o fluxo contínuo do aço líquido proveniente do módulo de Distribuidor.
- Funcionalidade 2: Monitorar a velocidade de lingotamento e a distribuição adequada do aço líquido no molde.
- Funcionalidade 3: Gerenciar o resfriamento primário e secundário das placas para garantir a qualidade do lingote.
- Funcionalidade 4: Alertar e registrar anomalias no processo de lingotamento para correção imediata.

Panela (*Back-end*):

- Funcionalidade 1: Receber o aço líquido do convertedor e prepará-lo para o processo de lingotamento.
- Funcionalidade 2: Realizar a purga do distribuidor com argônio para evitar contaminação.
- Funcionalidade 3: Adicionar ligas e cálcio conforme necessário para obter a composição desejada do aço.
- Funcionalidade 4: Monitorar e controlar a temperatura da panela e o nível de aço líquido.

Otimização (*Back-end*):

- Funcionalidade 1: Utilizar os dados dos pedidos recebidos do módulo de Grades para otimizar o processo de lingotamento.
- Funcionalidade 2: Definir os parâmetros ideais para a produção de lingotes com base nas especificações dos pedidos.
- Funcionalidade 3: Enviar os parâmetros otimizados aos módulos correspondentes para implementação no processo de lingotamento.
- Funcionalidade 4: Monitorar o desempenho do processo de lingotamento e fazer ajustes contínuos para maximizar a eficiência.

Distribuidor (*Back-end*):

- Funcionalidade 1: Receber o aço líquido da panela e direcioná-lo para os moldes por meio de válvulas submersas.
- Funcionalidade 2: Controlar a vazão do aço líquido para garantir uma distribuição uniforme nos moldes.
- Funcionalidade 3: Realizar a purga do distribuidor com argônio para evitar contaminação e oxidação do aço.
- Funcionalidade 4: Monitorar e ajustar a temperatura do distribuidor para manter o aço líquido na faixa adequada.

Molde (*Back-end* e *Front-end*):

- Funcionalidade 1: Promover a primeira solidificação do aço líquido e direcioná-lo para formar placas sólidas.
- Funcionalidade 2: Controlar a oscilação do molde para evitar adesão do aço sólido à parede do molde.
- Funcionalidade 3: Monitorar a largura da placa em formação e ajustar o molde conforme necessário.
- Funcionalidade 4: Instalar termopares para coletar e transmitir dados de temperatura do molde ao sistema.

Veios (*Back-end* e *Front-end*):

- Funcionalidade 1: Extrair as placas sólidas formadas nos moldes e conduzi-las ao sistema de resfriamento secundário.
- Funcionalidade 2: Controlar a velocidade de extração dos veios para evitar defeitos superficiais nas placas.
- Funcionalidade 3: Monitorar a qualidade das placas durante a extração e acionar alertas em caso de falhas.

Resfriamento Secundário (*Back-end* e *Front-end*):

- Funcionalidade 1: Realizar a remoção controlada de calor das placas para completar a solidificação.

- Funcionalidade 2: Utilizar zonas de resfriamento com sistemas de spray de água ou air mist para controlar a temperatura.
- Funcionalidade 3: Monitorar a velocidade das placas durante o resfriamento e ajustar os parâmetros conforme necessário.
- Funcionalidade 4: Verificar a qualidade final das placas após o resfriamento e registrar os resultados.

Máquina de Corte (*Back-end* e *Front-end*):

- Funcionalidade 1: Receber as placas sólidas resfriadas e realizar o corte de acordo com as dimensões especificadas nos pedidos.
- Funcionalidade 2: Utilizar maçaricos de oxigênio e gás natural controlados por válvulas para efetuar o corte.
- Funcionalidade 3: Monitorar o processo de corte e garantir a precisão das dimensões das placas resultantes.

Requisitos Não Funcionais

Desempenho:

- O sistema MLC1 é capaz de processar pedidos em tempo real e realizar o lingotamento contínuo em alta velocidade.
- O tempo de resposta do sistema MLC1 é rápido, permitindo o controle imediato das operações.

Segurança:

- O acesso ao sistema MLC1 é controlado por autenticação e autorização, garantindo que apenas usuários autorizados possam operá-lo.
- Os dados do sistema MLC1 são armazenados e transmitidos de forma segura, utilizando criptografia para proteção.

Confiabilidade:

- O sistema MLC1 é altamente confiável, evitando falhas e quedas de desempenho que possam impactar a produção.

Escalabilidade:

- O sistema MLC1 é projetado para lidar com aumento na demanda e expansão da produção, até certo nível, garantindo que novos recursos e capacidades possam ser adicionados conforme necessário.

Usabilidade:

- A interface do sistema MLC1 é intuitiva e de fácil utilização, permitindo que os operadores interajam de forma eficiente e sem a necessidade de treinamentos extensos.

Manutenção:

- O sistema MLC1 é projetado com uma arquitetura modular e de fácil manutenção, permitindo que atualizações e correções possam ser implementadas de forma ágil e sem impactos negativos.

Disponibilidade:

- O Sistema MLC1 foi projetado para estar disponível 24 horas por dia, 7 dias por semana, garantindo a continuidade da produção.

Tolerância a Falhas:

- O Sistema MLC1 foi projetado para lidar com possíveis falhas e interrupções, minimizando o impacto na produção e possibilitando a recuperação rápida em caso de problemas.

Tratamento de Erros e Resiliência:

- O Sistema MLC1 inclui mecanismos para o tratamento de erros, garantindo que falhas inesperadas sejam detectadas, registradas e tratadas adequadamente. Ele é projetado para ser resiliente, sendo capaz de se recuperar de falhas e manter a disponibilidade dos serviços essenciais.

Integrações e Interfaces

O Sistema *Back-end* MLC1 pode se integrar com outros sistemas externos, como sistemas de gestão de produção (MES), sistemas de controle de qualidade, sistemas de estoque e outros. Essas integrações são realizadas por meio de APIs, serviços web ou outros protocolos de comunicação aceitos pelas aplicações externas. A integração com outros sistemas permite que o MLC1 obtenha informações atualizadas sobre pedidos de produção, estoque de materiais, dados de qualidade, entre outros, garantindo a sincronização adequada entre os diversos sistemas da planta industrial.

Comunicação com Front-end:

- O sistema *Back-end* deve receber e processar os pedidos de lingotamento enviados pelo Front.
- Deve ser estabelecida uma API ou protocolo de comunicação para troca de informações entre o Front e o *Back-end*.

Integração com OPC (Open Platform Communications):

- O sistema *Back-end* é capaz de se integrar com dispositivos e equipamentos industriais que suportam o padrão OPC para troca de dados em tempo real.

Integração com MQ (Message Queue):

- Utilização de filas de mensagens para comunicação assíncrona entre os diferentes módulos do sistema, garantindo o processamento confiável e escalável das informações.

Regras de Negócio

Controle de Velocidade de Lingotamento:

- O Sistema MLC1 calcula e ajusta automaticamente a velocidade de lingotamento com base em dados de temperatura, demanda de produção e condições do processo.

Gestão de Qualidade das Placas:

- Devem ser aplicadas regras de negócio para verificar a qualidade das placas produzidas, identificando e tratando desvios e defeitos.

Considerações de Segurança

Controle de Acesso:

- O sistema possuí diferentes níveis de acesso e permissões, garantindo que apenas usuários autorizados possam realizar determinadas ações.

Criptografia de Dados:

- A segurança é uma prioridade no Sistema MLC1. Ele inclui medidas para proteger os dados, evitar acesso não autorizado e garantir a integridade das informações. A autenticação e autorização são aplicadas para controlar o acesso às

funcionalidades do sistema, e as informações sensíveis são criptografadas para garantir a confidencialidade dos dados.

Exemplos de Uso

Início do Lingotamento:

- Um operador inicia o processo de lingotamento através do Front, enviando os parâmetros do pedido e as especificações da placa.

Monitoramento do Lingotamento:

- Durante o processo de lingotamento, o sistema monitora constantemente as temperaturas, velocidades e qualidade das placas produzidas.

Controle de Velocidade:

- Com base nas informações coletadas, o sistema *Back-end* realiza cálculos para ajustar a velocidade de lingotamento de acordo com as necessidades do processo e a demanda de produção.

Interação com Veios e Distribuidor:

- O sistema coordena a atuação dos veios e distribuidor, garantindo a distribuição uniforme do aço líquido e evitando falhas e obstruções no processo.

Resfriamento Secundário:

- O resfriamento secundário é controlado pelo sistema, que regula a temperatura e o fluxo de água para garantir a solidificação adequada do aço dentro do molde.

Extração e Corte das Placas:

- Após a solidificação da placa, a máquina de corte é acionada pelo sistema para cortar a placa na medida especificada no pedido do cliente.

Análise de Baumann:

- O sistema realiza a análise de Baumann para avaliar a qualidade das placas produzidas, verificando se estão dentro das especificações de segregação central, trincas, porosidade, inclusões, entre outros.

Diagrama de Arquitetura

O sistema de lingotamento contínuo MLC1 é composto por uma arquitetura distribuída, com os seguintes módulos principais:

a) *Front-end*: Interface de usuário que permite aos operadores iniciar o processo de lingotamento, configurar os pedidos e monitorar o status da produção.
b) *Back-end*: Módulo responsável por processar os pedidos de lingotamento, controlar a velocidade, interagir com os veios, distribuidor e moldes, e realizar análise de qualidade.
c) Veios e Distribuidor: Módulos físicos que atuam no processo de distribuição do aço líquido para os moldes.
d) Moldes: Módulos responsáveis pela solidificação do aço líquido e formação das placas.
e) Máquina de Corte: Módulo responsável por cortar as placas de acordo com as especificações do pedido.

Nota:

O Descritivo Funcional do sistema de lingotamento contínuo MLC1 foi elaborado com base nos documentos, trechos de código, arquivos de sistema, manuais e explicações fornecidas anteriormente neste documento. O objetivo é fornecer uma visão completa das funcionalidades, comportamentos e interações do sistema, com foco no processo de lingotamento do aço líquido até a obtenção do lingote. O documento serve como uma referência essencial para todos os envolvidos no desenvolvimento, operação e manutenção do sistema, garantindo a compreensão clara de sua operação e requisitos.

Em resumo, o Sistema *Back-end* MLC1 é responsável por todas as funcionalidades de controle e gerenciamento do processo de produção, envolvendo as diversas partes do sistema, desde a criação de pedidos até a programação das mesas de saída e o controle de qualidade. A comunicação entre o *Back-end* e o *Front-end* é realizada por meio de APIs, e a integração com outros sistemas pode ocorrer através de diferentes protocolos de comunicação. A utilização de OPC e MQ permite a comunicação eficiente com dispositivos de automação e a troca de mensagens confiável entre os componentes do sistema.

8 Conclusão

O projeto de Lingotamento Contínuo da Máquina 1 (MLC1) apresentado é uma aplicação complexa e abrangente que envolve diversas etapas e componentes para garantir seu pleno funcionamento. Ao longo deste documento, foram abordados aspectos cruciais do projeto, desde a configuração inicial até a integração entre *front-end* e *back-end*.

No início, foram discutidos os requisitos e o escopo do projeto, delineando as metas e funcionalidades esperadas. Em seguida, foram apresentadas as etapas de instalação e configuração do ambiente de desenvolvimento, incluindo a configuração de dependências e frameworks.

Posteriormente, foram abordados aspectos cruciais do desenvolvimento, como a estrutura do projeto, a definição dos módulos e a configuração do arquivo build.gradle para versionar e compilar corretamente os arquivos. Além disso, foram apresentadas as etapas para copiar o conteúdo da pasta Source Packages de outros projetos e a localização das configurações JNDI no *WildFly* e *WebSphere*.

Na seção de integração *front-end* e *back-end*, foram apresentados os passos para a criação de menus e telas, incluindo a definição de permissões e a utilização de serviços para o envio e recebimento de dados. Também foram abordados aspectos relacionados à criação de novos contextos e variáveis de ambiente, essenciais para a adaptação da aplicação a diferentes ambientes de implantação.

Ao final deste documento, podemos concluir que o projeto de Lingotamento Contínuo da Máquina 1 (MLC1) é um empreendimento complexo que demanda conhecimento técnico e habilidades avançadas em desenvolvimento de software. A integração entre *front-end* e *back-end*, juntamente com a configuração correta do ambiente e a utilização adequada de frameworks e serviços, são elementos cruciais para o sucesso da aplicação.

Através da compreensão e aplicação dos conceitos e passos apresentados neste documento, os desenvolvedores terão uma base sólida para dar continuidade ao desenvolvimento e aprimoramento da aplicação MLC1, garantindo sua funcionalidade, eficiência e adaptabilidade aos requisitos específicos de cada ambiente de implantação.

Em suma, o projeto MLC1 é um exemplo de uma aplicação robusta e sofisticada, que integra diversos componentes e tecnologias para oferecer uma solução completa para o processo de Lingotamento Contínuo. O sucesso desse projeto depende da compreensão profunda dos requisitos, do uso adequado de ferramentas e frameworks, e da aplicação de boas práticas de desenvolvimento de software. Com o conhecimento adquirido neste documento, os desenvolvedores estarão preparados para enfrentar os desafios do projeto e alcançar os objetivos propostos.