CENTRO UNIVERSITÁRIO FEI

ANTONIO GUSTAVO MUNIZ DA SILVA - 22.119.001 - 0
JOÃO VITOR DIAS DOS SANTOS - 22.119.006 - 9

PROJETO SISTEMAS DISTRIBUÍDOS
PLANTA DE BIODIESEL

São Bernardo do Campo
2022

1 CÓDIGO FONTE

Nesse capítulo, explicaremos o código descrevendo tanto as entidades criadas para representar os tanques presentes no fluxo de geração de biodiesel como a integração entre os diferentes tanques.

1.1 ENTIDADES

Primeiramente, nós criamos uma classe *Tanque* que representa os tanques, assim como exibimos abaixo. Na qual além dos atributos básicos, também adicionamos o método "transferir_baseado_vazao()", que é necessário para a comunicação entre os outros componentes da planta de biodiesel.

```
class Tanque:
   def __init__(self, vazao=1):
      self.vazao = vazao
      self.quantidade_armazenada = 0

   def depositar_insumo(self, quantidade):
      self.quantidade_armazenada += quantidade
      return self.quantidade_armazenada

   def transferir_baseado_vazao(self):
      quantidade_transferencia = 0

      if self.quantidade_armazenada >= self.vazao:
         self.quantidade_armazenada -= self.vazao
         quantidade_transferencia = self.vazao
      else:
         quantidade_transferencia = self.quantidade_armazenada
         self.quantidade_armazenada = 0
      return quantidade_transferencia

   def to_output(self):
      return {
         "quantidade_armazenada": self.quantidade_armazenada
      }
```

Também construimos a entidade *Decantador* que representa o decantador. Ela possui dois métodos principais: o primeiro é para armazenar a solução recebida do reator, respeitando sempre a capacidade máxima definida; já o segundo refere-se ao processo principal do decantador de geração de solução, que, além de retornar a quantidade de solução gerada, também ativa o modo de repouso do decantador.

```
class Decantador:
   def __init__(self):
      self.capacidade_maxima = 10
      self.tempo_repouso = 5
      self.quantidade_armazenada = 0
      self.esta_em_repouso = False
      self.tempo_inicio_processamento = None
      self.ciclos_processamento = 0

   def armazena_solucao(self, quantidade):
      quantidade_nao_armazenada = quantidade

      if quantidade + self.quantidade_armazenada <= self.capacidade_maxima:
         self.quantidade_armazenada += quantidade
         quantidade_nao_armazenada = 0
      return quantidade_nao_armazenada

   def processa_saida(self):

      #checa se passou 5 segundos e tira do modo repouso
      if self.esta_em_repouso and (time.time() > (self.tempo_inicio_processamento +
          self.tempo_repouso)):
         print("Decantador saiu do modo repouso")
         self.esta_em_repouso = False

      if not self.esta_em_repouso and self.quantidade_armazenada > 1:
         # quantidade solucao
         payload = {
            "quantidade_glicerina": self.quantidade_armazenada * 0.01,
            "quantidade_et_oh": self.quantidade_armazenada * 0.03,
            "quantidade_lavagem": self.quantidade_armazenada * 0.96,
            "total": self.quantidade_armazenada
         }
         self.tempo_inicio_processamento = time.time()
         self.esta_em_repouso = True
         print("Decantador iniciou o modo repouso")
         self.ciclos_processamento+=1
```

```
            self.quantidade_armazenada = 0
            return payload

        return {
                "quantidade_glicerina": 0,
                "quantidade_et_oh": 0,
                "quantidade_lavagem": 0,
                "total": 0
            }

    def to_output(self):
        return {
            "quantidade_armazenada": self.quantidade_armazenada,
            "esta_em_repouso": self.esta_em_repouso,
            "ciclos_processamento": self.ciclos_processamento
        }
```

Além do decantador, também construímos a classe referente à *Lavagem*, que possui uma lógica de métodos semelhantes ao que foi feito nos tanques. Além de receber o insumo, por meio do método "depositar()", a lavagem também realiza a transferência de insumos baseada na sua vazão de 1,5 segundos.

```
class Lavagem:
    def __init__(self):
        self.vazao = 1.5
        self.porcentagem_perda = 0.075 # 3 lavagem em sequencia
        self.perda_total = 0
        self.quantidade_armazenada = 0
        self.quantidade_transferida = 0

    def depositar(self, quantidade):
        self.quantidade_armazenada += quantidade * (1 - self.porcentagem_perda)
        self.perda_total += quantidade * self.porcentagem_perda
        return self.quantidade_armazenada

    def transferir_baseado_vazao(self):
        quantidade_transferencia = 0
        if self.quantidade_armazenada >= self.vazao:
            self.quantidade_armazenada -= self.vazao
            quantidade_transferencia = self.vazao
        else:
            quantidade_transferencia = self.quantidade_armazenada
            self.quantidade_armazenada = 0
```

```
        self.quantidade_transferida+=quantidade_transferencia
        return quantidade_transferencia

    def to_output(self):
        return {
            "perda_total": self.perda_total,
            "quantidade_armazenada": self.quantidade_armazenada
        }
```

O *Secador* é bem semelhante à lavagem e possui métodos de mesmo nome, porém a lógica da entidade difere no que diz respeito à perda, atributo que não existe na lavagem. A perda é somada no atributo de "perda_total"e o que não é perdido é transferido para o próximo componente.

```
class Secador:
    def __init__(self):
        self.vazao = 0.2 # processam 0.2 litros por segundo -> rever
        self.tempo_por_litro = 5
        self.perda_total = 0
        self.porcentagem_perda = 0.05
        self.quantidade_armazenada = 0

    def depositar(self, quantidade):
        self.quantidade_armazenada += quantidade * (1 - self.porcentagem_perda)
        self.perda_total += quantidade * self.porcentagem_perda

        return self.quantidade_armazenada

    def transferir_baseado_vazao(self):
        quantidade_transferencia = 0
        if self.quantidade_armazenada >= self.vazao:
            self.quantidade_armazenada -= self.vazao
            quantidade_transferencia = self.vazao
        else:
            quantidade_transferencia = self.quantidade_armazenada
            self.quantidade_armazenada = 0
        return quantidade_transferencia

    def to_output(self):
        return {
            "perda_total": self.perda_total,
            "quantidade_armazenada": self.quantidade_armazenada
```

```
    }
```

Por fim, construímos o *Reator*. Ele foi a entidade mais longa, a nível de código, possuindo 8 métodos principais. Três deles referem-se apenas ao depósito de insumos provenientes dos tanques, sendo que nesses definimos uma lógica de limite. O reator quando atinge a porcentagem da proporção de um dos insumos, automaticamente para de recebê-lo. Além dessa lógica, temos um método para a checagem da proporção e outro para processar a solução que chegará ao decantador. Dentro do reator, também adicionamos o atributo de "estado", pois precisamos saber quando o reator está transferindo a solução pra não realizar duas transferências de conteúdo ao mesmo tempo.

```
class Reator:
    def __init__(self):
        self.litros_processamento_por_segundo = 5
        self.quantidade_armazenada_oleo = 0
        self.proporcao_armazenada_oleo = 0
        self.quantidade_armazenada_na_oh = 0
        self.proporcao_armazenada_na_oh = 0
        self.quantidade_armazenada_et_oh = 0
        self.proporcao_armazenada_et_oh = 0
        self.quantidade_total = 0
        self.vazao = 1
        self.ciclos_processamento = 0
        self.estado = "Ligado"

    def to_output(self):
        return {
            "quantidade_armazenada_oleo": self.quantidade_armazenada_oleo,
            "quantidade_armazenada_na_oh": self.quantidade_armazenada_na_oh,
            "quantidade_armazenada_et_oh": self.quantidade_armazenada_et_oh,
            "quantidade_total": self.quantidade_total,
            "estado": self.estado,
            "ciclos_processamento": self.ciclos_processamento
        }

    def finalizar_transferencia(self):
        self.estado = "Ligado"
        return self.estado

    def depositar_insumo_recebido(self, payload):
        quantidade_a_ser_retornada = 0
```

```
    if "quantidade_et_oh" in payload:
        quantidade_a_ser_retornada =
            self.depositar_et_oh(float(payload["quantidade_et_oh"]))
    elif "quantidade_na_oh" in payload:
        quantidade_a_ser_retornada =
            self.depositar_na_oh(float(payload["quantidade_na_oh"]))
    elif "quantidade_oleo" in payload:
        quantidade_a_ser_retornada =
            self.depositar_oleo(float(payload["quantidade_oleo"]))

    return quantidade_a_ser_retornada

def depositar_oleo(self, quantidade):
    quantidade_nao_armazenada = quantidade

    if self.quantidade_armazenada_oleo < 2.5:
        if self.quantidade_armazenada_oleo + quantidade > 2.5:
            quantidade_nao_armazenada = quantidade - (2.5 -
                self.quantidade_armazenada_oleo)
            self.quantidade_armazenada_oleo = 2.5
        else:
            self.quantidade_armazenada_oleo += quantidade
            quantidade_nao_armazenada = 0
        self.quantidade_total += quantidade - quantidade_nao_armazenada
        self.proporcao_armazenada_oleo = (self.quantidade_armazenada_oleo /
            self.litros_processamento_por_segundo) * 100
    return quantidade_nao_armazenada

def depositar_na_oh(self, quantidade):
    quantidade_nao_armazenada = quantidade

    if self.quantidade_armazenada_na_oh < 1.25:
        if self.quantidade_armazenada_na_oh + quantidade > 1.25:
            quantidade_nao_armazenada = quantidade - (1.25 -
                self.quantidade_armazenada_na_oh)
            self.quantidade_armazenada_na_oh = 1.25
        else:
            self.quantidade_armazenada_na_oh += quantidade
            quantidade_nao_armazenada = 0
        self.quantidade_total += quantidade - quantidade_nao_armazenada
        self.proporcao_armazenada_na_oh = (self.quantidade_armazenada_na_oh /
            self.litros_processamento_por_segundo) * 100
    return quantidade_nao_armazenada
```

```
def depositar_et_oh(self, quantidade):
    quantidade_nao_armazenada = quantidade

    if self.quantidade_armazenada_et_oh < 1.25:
        if self.quantidade_armazenada_et_oh + quantidade > 1.25:
            quantidade_nao_armazenada = quantidade - (1.25 -
                self.quantidade_armazenada_et_oh)
            self.quantidade_armazenada_et_oh = 1.25
        else:
            self.quantidade_armazenada_et_oh += quantidade
            quantidade_nao_armazenada = 0
        self.quantidade_total += quantidade - quantidade_nao_armazenada
        self.proporcao_armazenada_et_oh = (self.quantidade_armazenada_et_oh /
            self.litros_processamento_por_segundo) * 100
    return quantidade_nao_armazenada

def checar_proporcao_acionamento(self):
    if self.quantidade_total != 0:
        porcentagem_na_oh = self.proporcao_armazenada_na_oh == 25
        porcentagem_et_oh = self.proporcao_armazenada_et_oh == 25
        porcentagem_oleo = self.proporcao_armazenada_oleo == 50

        if porcentagem_na_oh and porcentagem_et_oh and porcentagem_oleo:
            print("Porcentagem atingida...")
            print("Acionando processamento reator")

            if self.estado != "Em Transferencia":
                return self.processar()

    return {
        "quantidade_solucao": 0
    }

def processar(self):
    # 5 litros
    quantidade_processada = 5

    self.quantidade_total = 0
    self.quantidade_armazenada_oleo = 0
    self.quantidade_armazenada_et_oh = 0
    self.quantidade_armazenada_na_oh = 0

    self.estado = "Em Transferencia"
    self.ciclos_processamento += 1
```

```
    return {
        "quantidade_solucao": int(quantidade_processada)
    }
```

1.2 COMUNICAÇÃO

A comunicação entre os componentes de sistema é feito por meio de métodos definidos na classe *HandlerSystem*:

```
class HandlerSystem:
    def __init__(self):

    def depositar_tanque(self, quantidade, tanque, payload_parameter):

    def depositar_oleo_residual(self):

    def transferir_oleo_residual(self):

    def depositar_et_oh(self):

    def transferir_et_oh(self):

    def depositar_na_oh(self):

    def transferir_na_oh(self):

    def depositar_tanque_reator(self, payload={}):

    def processo_reator(self):

    def retornar_solucao_ao_reator(self, quantidade):

    def depositar_decantador(self, payload={}):

    def depositar_secador_etoh(self, payload={}):

    def transferir_secador_etoh(self):

    def depositar_tanque_glicerina(self, payload={}):

    def depositar_lavagem(self, payload={}):
```

```
def transferir_lavagem(self):

def depositar_secador_biodiesel(self, result_post):

def depositar_tanque_biodiesel(self, payload={}):

def processo_decantador(self):

def transferir_secador_biodiesel(self):
```

Adicionamos apenas à declaração dos métodos para não poluir o relatório. Porém, é possível checar diretamente o código e ver o funcionamento da interação entre esses métodos. Em linhas gerais o orquestrador, que será descrito mais à frente, coordena todas as funções do *HandlerSystem* e permite com que a planta execute o que suas funções.

Utilizamos a arquitetura de microsserviços, na qual cada componente do sistema da planta de Biodiesel é um API Rest. A interação entre os componentes é feita pela classe *HandlerSystem*. Todos os métodos apresentados acima são utilizados para realizar a comunicação entre os componentes da planta o *Handler*.

Cada método possui uma lógica definida, que pode ser conferida no repositório do código.

O método *POST* serve apenas para depositar insumos dentro de um tanque ou componente, por exemplo, no reator, o *POST* serve para receber os insumos provenientes dos tanques de óleo, de NaOH e de EtOh. Enquanto para o decantador, o mesmo método serve pra receber o insumo de solução gerado no reator. Já o método *PUT* serve para iniciar a execução do processo que esse componente realiza, por exemplo, nos tanques, serve pra realizar a transferência baseado na vazão, enquanto, no reator, serve para gerar a solução de NaOh, EtOh e óleo. No decantador esse método foi utilizado para a criação da solução da lavagem, glicerina e EtOh, bem como também realizar o processo de repouso estipulado. Por fim, o método *GET* serve apenas para retornar o estado do componente, foi esse método que foi utilizado para a exibição dos resultados.

Todas essas APIs, são expostas por intermédio da classe *HandlerRequests*, ela expõe o *endpoint* de cada API sempre respeitando as regras da planta.

1.2.1 Exposição da API do Tanque de óleo

```
# metodos tanque oleo residual

@app.route("/tanque_oleo", methods=['POST'])
def post_incomes_oleo():
    return {
        "quantidade_oleo":
            tanque_oleo.depositar_insumo(float(request.json["quantidade_oleo"]))
    }


@app.route("/tanque_oleo", methods=['PUT'])
def put_incomes_oleo():
    return {
        "quantidade_oleo": tanque_oleo.transferir_baseado_vazao()
    }


@app.route("/tanque_oleo", methods=['GET'])
def get_oleo():
    return tanque_oleo.__dict__
```

1.2.2 Exposição da API do Tanque de EtOh

```
# metodos tanque et oh

@app.route("/tanque_etoh", methods=['POST'])
def post_incomes_etoh():
    return {
        "quantidade_et_oh":
            tanque_etoh.depositar_insumo(float(request.json["quantidade_et_oh"]))
    }


@app.route("/tanque_etoh", methods=['PUT'])
def put_incomes_etoh():
    return {
        "quantidade_et_oh": tanque_oleo.transferir_baseado_vazao()
    }


@app.route("/tanque_etoh", methods=['GET'])
```

```
def get_et_oh():
    return tanque_etoh.__dict__
```

1.2.3 Exposição da API do Tanque de NaOh

```
# metodos tanque na oh

@app.route("/tanque_naoh", methods=['POST'])
def post_incomes_naoh():
    return {
        "quantidade_na_oh":
            tanque_naoh.depositar_insumo(float(request.json["quantidade_na_oh"]))
    }


@app.route("/tanque_naoh", methods=['PUT'])
def put_incomes_naoh():
    return {
        "quantidade_na_oh": tanque_naoh.transferir_baseado_vazao()
    }


@app.route("/tanque_naoh", methods=['GET'])
def get_na_oh():
    return tanque_naoh.__dict__
```

1.2.4 Exposição da API do Reator

O reator possui uma operação a mais, o *PATCH*, que serve para alterar o estado de reator. Como explicamos anteriormente, quando uma transferência está sendo feita, o reator fica num estado de "Em Transferência", que é alterado quando realizamos um *PATCH*.

```
# metodos reator

@app.route("/reator", methods=['POST'])
def post_incomes_reator():
    return {
        "quantidade_retorno_reator": reator.depositar_insumo_recebido(request.json)
    }
```

```
@app.route("/reator", methods=['PUT'])
def put_incomes_reator():
   return reator.checar_proporcao_acionamento()


@app.route("/reator", methods=['PATCH'])
def patch_incomes_reator():
   return reator.finalizar_transferencia()


@app.route("/reator", methods=['GET'])
def get_reator():
   return reator.__dict__
```

1.2.5 Exposição da API do Decantador

```
# metodos decantador

@app.route("/decantador", methods=['POST'])
def post_incomes_decantador():
   return {
      "quantidade_retorno_decantador":
          decantador.armazena_solucao(float(request.json["quantidade_solucao"]))
   }


@app.route("/decantador", methods=['PUT'])
def put_incomes_decantador():
   return decantador.processa_saida()


@app.route("/decantador", methods=['GET'])
def get_decntador():
   return decantador.__dict__
```

1.2.6 Exposição da API do Tanque de glicerina

```
# metodos tanque glicerina

@app.route("/tanque_glicerina", methods=['POST'])
def get_incomes_glicerina():
    return {
        "quantidade_glicerina":
            tanque_glicerina.depositar_insumo(float(request.json["quantidade_glicerina"]))
    }


@app.route("/tanque_glicerina", methods=['GET'])
def get_glicerina():
    return tanque_glicerina.__dict__
```

1.2.7 Exposição da API do Secador de EtOh

```
# metodos secador

@app.route("/secador_etoh", methods=['POST'])
def get_incomes_secador_etoh():
    return {
        "quantidade_et_oh":
            secador_etoh.depositar(float(request.json["quantidade_et_oh"]))
    }


@app.route("/secador_etoh", methods=['PUT'])
def put_incomes_secador_etoh():
    return {
        "quantidade_et_oh": secador_etoh.transferir_baseado_vazao()
    }


@app.route("/secador_etoh", methods=['GET'])
def get_secador_etoh():
    return secador_etoh.__dict__
```

1.2.8 Exposição da API da Lavagem

```
# meotdos lavagem

@app.route("/lavagem", methods=['POST'])
def get_incomes_lavagem():
    return {
        "quantidade_lavagem":
            lavagem.depositar(float(request.json["quantidade_lavagem"]))
    }


@app.route("/lavagem", methods=['PUT'])
def put_incomes_lavagem():
    return {
        "quantidade_lavagem": lavagem.transferir_baseado_vazao()
    }


@app.route("/lavagem", methods=['GET'])
def get_lavagem():
    return lavagem.__dict__
```

1.2.9 Exposição da API de Secador de Biodiesel

```
@app.route("/secador_biodiesel", methods=['POST'])
def post_incomes_secador_biodiesel():
    return {
        "quantidade_biodiesel":
            secador_biodiesel.depositar(float(request.json["quantidade_lavagem"]))
    }

@app.route("/secador_biodiesel", methods=['PUT'])
def put_incomes_secador_biodiesel():
    return {
        "quantidade_biodiesel": secador_biodiesel.transferir_baseado_vazao()
    }


@app.route("/secador_biodiesel", methods=['GET'])
def get_secador_biodiesel():
    return secador_biodiesel.__dict__
```

1.2.10 Exposição da API do Tanque de Biodiesel

```
# metodos tanque biodiesel

@app.route("/tanque_biodiesel", methods=['POST'])
def get_incomes_tanque_biodiesel():
   return {
      "quantidade_biodiesel":
          tanque_biodiesel.depositar_insumo(float(request.json["quantidade_biodiesel"]))
   }


@app.route("/tanque_biodiesel", methods=['GET'])
def get_tanque_biodiesel():
   return tanque_biodiesel.__dict__
```

1.3 ORQUESTRADOR

Além das APIs e do *HandlerSystem*, temos também a classe principal, o *main*, que é responsável por criar as *threads* na qual cada componente é executado. Assim como mostra o código abaixo.

```
def start_fluxo():
   # periodicamente enviar os insumos
   depositar_oleo_residual()
   depositar_et_oh()
   depositar_na_oh()

   # periodicamente transferir os insumos entre os componentes
   transferir_oleo_residual()
   transferir_et_oh()
   transferir_na_oh()

   # periodicamente o reator processa
   processo_reator()
   processo_decantador()

   transfere_secador_etoh()
   transfere_lavagem()
   transfere_secador_biodiesel()
```

```
start_fluxo()
```

Cada um desses métodos contidos na função de *start_fluxo()* cria uma thread e executa o que lhes são propostos para que interação entre os componentes da planta ocorra com sucesso.

1.4 REPOSITÓRIO DO GITHUB

Além do código python, o repositório também contêm o *script bash* que utilizamos para a execução do projeto na máquina de cada um, bem como os resultados dos testes que realizamos. O link para o repositório pode ser encontrado na Seção 3.

2 RESULTADOS

Realizamos dois testes da nossa planta, ambos com duração de uma hora. No primeiro teste, obtivemos o seguinte resultado ao interromper o fluxo.


```
{
"decantador": {
   "ciclos_processamento": 264,
   "esta_em_repouso": false,
   "quantidade_armazenada": 0
},
"lavagem": {
   "perda_total": 47.51999999999994,
   "quantidade_armazenada": 0
},
"reator": {
   "ciclos_processamento": 132,
   "estado": "Ligado",
   "quantidade_armazenada_et_oh": 1.0,
   "quantidade_armazenada_na_oh": 1.25,
   "quantidade_armazenada_oleo": 2.0,
   "quantidade_total": 4.25
},
"secador_biodiesel": {
   "perda_total": 29.303999999999792,
   "quantidade_armazenada": 0.0
},
"secador_etoh": {
   "perda_total": 0.9899999999999972,
   "quantidade_armazenada": 0
},
"tanque_biodiesel": {
   "quantidade_armazenada": 556.775999999989
},
"tanque_etoh": {
   "quantidade_armazenada": 963.8100000000038
},
"tanque_glicerina": {
   "quantidade_armazenada": 6.599999999999985
},
"tanque_naoh": {
   "quantidade_armazenada": 1641.75
```

```
},
"tanque_oleo": {
   "quantidade_armazenada": 0
}
```


Nesse primeiro teste a planta gerou 556,78 litros de biodiesel e a quantidade restante nos outros tanques é 0, 1641,75, 6,6, 963,8 litros para os de óleo, NaOh, glicerina e EtOh, respectivamente. Além disso, é possível ver que o reator ainda possuía 4,25 litros de mistura em seu compartimento, o decantador, por sua vez, possuía 0 litros armazenado. Os componentes de secadores, tanto EtOh quanto biodiesel, possuíam 0 litros armazenados. Outro ponto interessante de destacar sobre o teste é referente à lavagem. A quantidade de emulsão é chamada de "perda_total"e, no teste, atingiu quase 48 litros.

Esse primeiro teste evidencia alguns pontos do nosso projeto. O primeiro é a quantidade armazenada nos tanques de óleo, NaOh e EtOh. Como já foi dito, definimos uma lógica de que o reator precisa atingir 5 litros, respeitando as proporções. Esse respeito as proporções é feito do seguinte modo: caso ele atinja a quantidade que precisa, não irá mais receber aquele insumo. Desse modo os tanques de NaOh e EtOh, que possuem um depósito num intervalo de tempo menor, acabam armazenando muitos insumos dentro de seus próprios tanques, assim como mostra a referência acima. Enquanto o de óleo, que além de ter um intervalo de tempo muito maior entre os recebimento, é um que é mais demandado pelo reator. Por essa razão, em ambos os testes o tanque de óleo ficou com 0 litros armazenado.

Outro ponto que é possível observar nesse teste refere-se a relação entre o decantador e o reator. O reator, como já falamos, gerará 5 litros de insumo, porém, por sua vazão ser de 1 litro por segundo, ele irá depositar no decantador e só no próximo segundo enviará mais um litro. O efeito dessa geração de 5 litros é considerado como um ciclo. Após ele depositar no decantador, o mesmo irá logo gerar a solução para lavagem, glicerina e secador para, enfim, entrar no modo repouso e incrementar um ciclo de processamento. Nesses 5 segundos que o decantador está em repouso, o reator estará enviando os 4 litros restantes. Logo, após o fim do repouso o decantador processará os 4 litros que faltam em seu armazenamento e incrementará mais um ciclo de processamento. Esse comportamento é corroborado por ambos os testes, no qual o decantador sempre possui o dobro de ciclos que o reator.

O segundo teste possui resultados muito semelhantes, em termos quantitativos, ao primeiro teste feito. Além de ressaltar os pontos ditos anteriormente, sobre o que restou nos tanques.


```
{
```

```
    "decantador": {
        "ciclos_processamento": 262,
        "esta_em_repouso": false,
        "quantidade_armazenada": 0
    },
    "lavagem": {
        "perda_total": 47.15999999999994,
        "quantidade_armazenada": 0
    },
    "reator": {
        "ciclos_processamento": 131,
        "estado": "Ligado",
        "quantidade_armazenada_et_oh": 1.0,
        "quantidade_armazenada_na_oh": 1.25,
        "quantidade_armazenada_oleo": 2.0,
        "quantidade_total": 4.25
    },
    "secador_biodiesel": {
        "perda_total": 29.081999999999795,
        "quantidade_armazenada": 0
    },
    "secador_etoh": {
        "perda_total": 0.9824999999999973,
        "quantidade_armazenada": 0
    },
    "tanque_biodiesel": {
        "quantidade_armazenada": 552.557999999988
    },
    "tanque_etoh": {
        "quantidade_armazenada": 1007.6675000000037
    },
    "tanque_glicerina": {
        "quantidade_armazenada": 6.549999999999986
    },
    "tanque_naoh": {
        "quantidade_armazenada": 1691.5
    },
    "tanque_oleo": {
        "quantidade_armazenada": 0
    }
}
```

É importante destacar os campos de "estado"do reator e "repouso"do decantador, que foram explicados com mais detalhes na Seção 1.1 desse relatório. Ambom foram campos necessários para evitar a sobrecarga do fluxo, pois os componentes não são objetos e sim APIs. O estado de cada um desses campos é exibido nos *logs* acima.

Os resultados do projeto também estão em dois vídeos postados no Youtube. Caso deseje assisti-los, existe uma versão curta, de 2 minutos, com uma breve explicação dos resultados. Além dela, existe outra versão com 8 minutos em que explicamos com mais detalhes os resultados vistos em nossos testes. Ambas as versões podem ser acessadas pelos links que estão na Seção 3.

3 LINKS

Link do repositório no github:
https://github.com/antuniooh/sistemas-distribuidos-project/blob/main/TesteFinal.pdf

Link do video no youtube com duração de 2 minutos sobre os resultados do projeto:
https://youtu.be/j3W-CUmf7OA

Link do video no youtube com duração de 8 minutos sobre os resultados do projeto (versão estendida):
https://youtu.be/8Uvy1extHy0