UNIVERSIDADE DO ESTADO DA BAHIA - UNEB
DEPARTAMENTO DE CIÊNCIAS EXATAS E DA TERRA I

CONTROLE DE ALMOXARIFADO NA CONSTRUÇÃO CIVIL

Trabalho apresentado a UNEB para obtenção de nota na disciplina Computação Aplicada à Engenharia

Discente: Andreza Pereira; Jailmária Marques
Docente: Robson Marinho

Salvador - BA
2024

Sumário

1 RESUMO **1**

2 INTRODUÇÃO **2**

3 OBJETIVO **2**

4 METODOLOGIA DA PESQUISA **2**

5 PYTHON AUXILIANDO NO CONTROLE DE ALMOXARIFADO **3**

6 DESENVOLVIMENTO DO PROJETO (Quadros do Trello e Github) **8**

7 REDE DE PETRI **9**

8 CONCLUSÃO **10**

1 RESUMO

Este estudo tem como objetivo aprofundar a análise da gestão de almoxarifados na indústria da construção civil, com o propósito de identificar e propor melhorias eficientes que otimizem tanto o armazenamento quanto o controle de materiais. Reconhecendo a importância crucial do gerenciamento adequado do almoxarifado, busca-se assegurar não apenas a disponibilidade oportuna de materiais, mas também a minimização de desperdícios e o aumento da produtividade no cenário da construção civil. É notório que a falta de materiais pode acarretar atrasos significativos nas obras, sendo essencial sanar essa questão por meio de estratégias eficazes de gestão de almoxarifados.

Neste trabalho, serão cuidadosamente examinados os desafios enfrentados e as práticas comuns adotadas na organização de almoxarifados na indústria da construção. Com base nessa análise, serão apresentadas não apenas recomendações, mas também estratégias concretas que visam aprimorar o fluxo de materiais e fortalecer o controle de estoque. Tais melhorias propostas têm o potencial não apenas de resolver problemas imediatos, como a escassez de materiais, mas também de promover uma gestão mais eficiente e sustentável dos recursos, contribuindo assim para a otimização dos processos e para o sucesso geral dos projetos na construção civil.

Além dos aspectos relacionados à gestão de almoxarifados na construção civil, este estudo também explora o potencial da linguagem de programação Python para otimizar processos de controle de estoque e gestão de materiais.

Python que é uma linguagem aberta de programação que oferece uma ampla gama de bibliotecas e ferramentas que podem ser aplicadas no contexto da construção civil, desde a automação de tarefas rotineiras até a análise de dados para previsão de demanda e otimização de inventário.

Dessa forma o python consegue ser utilizado para análise de dados do estoque, que permite identificar padrões de consumo e tendências que podem nos auxiliar a prever demandas do almoxarifado. Além disso, conseguimos realizar dashboards para visualizar a operação

dos almoxarifado

Palavras-chave: Almoxarifado, Construção Civil, Organização, Gerenciamento de Materiais, Controle de Estoque.

2 INTRODUÇÃO

O setor da construção civil requer um eficiente sistema de organização de almoxarifados para garantir que os materiais necessários estejam disponíveis quando necessário, minimizar desperdícios, evitar atrasos e melhorar a eficiência geral do projeto. No entanto, muitas empresas enfrentam desafios na gestão de seus almoxarifados, como dificuldades de localização de materiais, falta de controle de estoque e perdas financeiras decorrentes de má gestão. Portanto, este estudo visa identificar e analisar as práticas comuns de organização de almoxarifados na construção civil, bem como propor estratégias de melhoria para otimizar o gerenciamento de materiais.

Uma das principais características do almoxarifado na construção civil é a diversidade de materiais e equipamentos armazenados. Desde materiais básicos, como cimento, areia e tijolos, até elementos mais complexos, como tubulações, ferragens e sistemas elétricos, é preciso gerenciar uma ampla variedade de itens. Além disso, a gestão de estoque também deve considerar as quantidades necessárias para cada etapa da obra, a fim de evitar excessos ou falta de materiais.

Outro aspecto importante no almoxarifado da construção civil é o controle rigoroso dos itens recebidos e utilizados. É necessário registrar e acompanhar a entrada e saída de materiais, verificando a conformidade com as especificações técnicas, a qualidade e a quantidade. A adoção de sistemas de inventário, etiquetagem e código de barras pode facilitar o controle e a rastreabilidade dos materiais, permitindo uma gestão mais precisa e eficiente.

Por meio de um planejamento eficiente, é possível otimizar a logística de recebimento, armazenamento e distribuição dos materiais, garantindo um fluxo contínuo e adequado de suprimentos. Isso contribui para reduzir desperdícios, evitar a falta de materiais em momentos críticos e assegurar a qualidade das construções.

3 OBJETIVO

O objetivo geral deste projeto é analisar as práticas atuais de controle de almoxarifado na construção civil e desenvolver um modelo de referência para a gestão desses dados na área. Os objetivos especícos são:

- Avaliar as estratégias de armazenamento de dados utilizadas atualmente no setor;
- Propor um modelo referência para catalogação dos itens presentes no almoxarifado na construção civil.

4 METODOLOGIA DA PESQUISA

A pesquisa será realizada por meio de uma abordagem mista, envolvendo revisão bibliográfica, observação direta em canteiros de obras e entrevistas com profissionais da área. A revisão bibliográfica abordará conceitos-chave relacionados à gestão de almoxarifados, boas

práticas de organização, métodos de controle de estoque e tecnologias utilizadas. A observação direta permitirá a coleta de dados sobre a atual organização do almoxarifado em diferentes canteiros de obras. As entrevistas com profissionais da área, como gestores de projetos e almoxarifes, fornecerão informações sobre desafios enfrentados e possíveis melhorias.

5 PYTHON AUXILIANDO NO CONTROLE DE ALMOXARIFADO

Python, uma linguagem de programação versátil e de alto nível, pode desempenhar um papel significativo no controle de almoxarifado na construção civil, oferecendo uma série de benefícios e recursos para otimizar a gestão dos materiais. Através de suas bibliotecas e recursos, o Python pode auxiliar em várias etapas do controle de estoque, desde o registro de entrada e saída de materiais até a análise de dados e a automação de processos.
Uma das formas pelas quais o Python pode ajudar é na criação de um sistema de inventário digitalizado para o almoxarifado. Sendo possivel desenvolver uma estrutura de dados capaz de registrar e acompanhar os materiais em estoque. Isso simplifica a tarefa de controlar as quantidades, especificações técnicas e datas de validade dos itens, facilitando a identificação de necessidades de reposição e evitando a falta ou o excesso de estoque.
Outra funcionalidade do Python é a integração com sistemas de gestão empresarial (ERP). Através de bibliotecas e APIs, é possível conectar o almoxarifado ao sistema central da empresa, garantindo uma troca de informações fluida e precisa. Isso possibilita o compartilhamento de dados em tempo real, agiliza o fluxo de informações entre diferentes setores e facilita a tomada de decisões estratégicas baseadas em informações atualizadas.

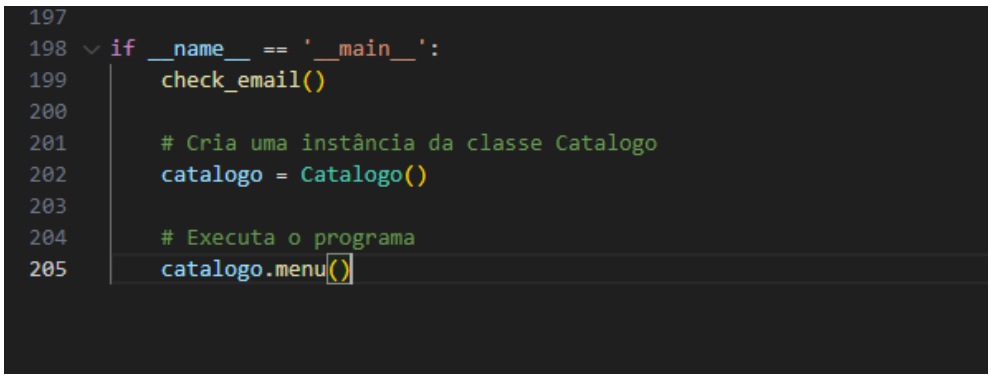
```python
import csv
import datetime
import imaplib
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.parser import BytesParser
from email.policy import default


class Catalogo:
    def __init__(self):
        self.arquivo_catalogo = "catalogo.csv"
        self.arquivo_historico = "historico.csv"

    def adicionar_item(self):
        codigo = input("Digite o código do item: ")
        descricao = input("Digite a descrição do item: ")
        quantidade = input("Digite a quantidade disponível: ")

        with open(self.arquivo_catalogo, "a", newline="") as arquivo:
            writer = csv.writer(arquivo)
            writer.writerow([codigo, descricao, quantidade])

        print("Item adicionado com sucesso!")

    def pesquisar_item(self):
        codigo = input("Digite o código do item: ")
```

```
    def pesquisar_item(self):
        codigo = input("Digite o código do item: ")

        try:
            with open(self.arquivo_catalogo, "r") as arquivo:
                reader = csv.reader(arquivo)
                for linha in reader:
                    if linha[0] == codigo:
                        print("Descrição:", linha[1])
                        print("Quantidade disponível:", linha[2])
                        return
            print("Item não encontrado.")
        except FileNotFoundError:
            print("Arquivo do catálogo não encontrado.")
        except Exception as e:
            print("Ocorreu um erro:", e)

    def atualizar_quantidade(self):
        codigo = input("Digite o código do item: ")
        nova_quantidade = input("Digite a nova quantidade disponível: ")

        try:
            linhas_atualizadas = []
            with open(self.arquivo_catalogo, "r") as arquivo:
                reader = csv.reader(arquivo)
                for linha in reader:
                    if linha[0] == codigo:
                        linha[2] = nova_quantidade
                    linhas_atualizadas.append(linha)

            with open(self.arquivo_catalogo, "w", newline="") as arquivo:
                writer = csv.writer(arquivo)
                writer.writerows(linhas_atualizadas)

            print("Quantidade atualizada com sucesso!")
        except FileNotFoundError:
            print("Arquivo do catálogo não encontrado.")
        except Exception as e:
            print("Ocorreu um erro:", e)

    def exibir_catalogo(self):
        try:
            with open(self.arquivo_catalogo, "r") as arquivo:
                reader = csv.reader(arquivo)
                for linha in reader:
                    print("Código:", linha[0])
                    print("Descrição:", linha[1])
                    print("Quantidade disponível:", linha[2])
                    print()
        except FileNotFoundError:
            print("Arquivo do catálogo não encontrado.")
        except Exception as e:
            print("Ocorreu um erro:", e)

    def exibir_historico(self):
        try:
            with open(self.arquivo_historico, "r") as arquivo:
                reader = csv.reader(arquivo)
                for linha in reader:
```

```
                    data = datetime.datetime.strptime(linha[0], "%Y-%m-%d %H:%M:%S")
                    descricao = linha[1]
                    quantidade = linha[2]
                    print("Data e hora:", data)
                    print("Descrição:", descricao)
                    print("Quantidade:", quantidade)
                    print()
            except FileNotFoundError:
                print("Arquivo do histórico não encontrado.")
            except Exception as e:
                print("Ocorreu um erro:", e)

    def remover_item(self):
        codigo = input("Digite o código do item a ser removido: ")
        item_encontrado = False

        try:
            linhas_atualizadas = []
            with open(self.arquivo_catalogo, "r") as arquivo:
                reader = csv.reader(arquivo)
                for linha in reader:
                    if linha[0] == codigo:
                        item_encontrado = True
                    else:
                        linhas_atualizadas.append(linha)

            if item_encontrado:
                with open(self.arquivo_catalogo, "w", newline="") as arquivo:
                    writer = csv.writer(arquivo)
                    writer.writerows(linhas_atualizadas)
                print("Item removido com sucesso!")
            else:
                print("Item não encontrado.")
        except FileNotFoundError:
            print("Arquivo do catálogo não encontrado.")
        except Exception as e:
            print("Ocorreu um erro:", e)

    def menu(self):
        while True:
            print("1. Adicionar item")
            print("2. Pesquisar item")
            print("3. Atualizar quantidade")
            print("4. Exibir catálogo")
            print("5. Exibir histórico")
            print("6. Remover item")
            print("7. Sair")

            opcao = input("Digite a opção desejada: ")

            if opcao == "1":
                self.adicionar_item()
            elif opcao == "2":
                self.pesquisar_item()
            elif opcao == "3":
                self.atualizar_quantidade()
            elif opcao == "4":
                self.exibir_catalogo()
            elif opcao == "5":
```

```
124         def menu(self):
143                     self.exibir_catalogo()
144                 elif opcao == "5":
145                     self.exibir_historico()
146                 elif opcao == "6":
147                     self.remover_item()
148                 elif opcao == "7":
149                     break
150                 else:
151                     print("Opção inválida. Tente novamente.")
152 
153                 print()
154 
155 
156     # Configurações do e-mail
157     IMAP_SERVER = 'imap.exemplo.com'
158     IMAP_PORT = 993
159     SMTP_SERVER = 'smtp.exemplo.com'
160     SMTP_PORT = 587
161     EMAIL_ACCOUNT = 'seu_email@exemplo.com'
162     PASSWORD = 'sua_senha'
163     FORWARD_TO_EMAIL = 'destinatario@exemplo.com'
164     AVISO = "Aviso: Almoxarifado com capacidade menor que 30%"
165 
166 
167     def check_email():
168         mail = imaplib.IMAP4_SSL(IMAP_SERVER, IMAP_PORT)
169         mail.login(EMAIL_ACCOUNT, PASSWORD)
170         mail.select('inbox')
171         status, messages = mail.search(None, '(UNSEEN)')
```

```
167 def check_email():
168     mail = imaplib.IMAP4_SSL(IMAP_SERVER, IMAP_PORT)
169     mail.login(EMAIL_ACCOUNT, PASSWORD)
170     mail.select('inbox')
171     status, messages = mail.search(None, '(UNSEEN)')
172     email_ids = messages[0].split()
173 
174     for email_id in email_ids:
175         status, msg_data = mail.fetch(email_id, '(RFC822)')
176         raw_email = msg_data[0][1]
177         msg = BytesParser(policy=default).parsebytes(raw_email)
178 
179         forward_email(msg)
180 
181     mail.logout()
182 
183 
184 def forward_email(original_msg):
185     msg = MIMEMultipart()
186     msg['From'] = EMAIL_ACCOUNT
187     msg['To'] = FORWARD_TO_EMAIL
188     msg['Subject'] = "Fwd: " + original_msg['Subject']
189     body = AVISO + "\n\n" + original_msg.get_body(preferencelist=('plain',)).get_content()
190 
191     msg.attach(MIMEText(body, 'plain'))
192     with smtplib.SMTP(SMTP_SERVER, SMTP_PORT) as server:
193         server.starttls()
194         server.login(EMAIL_ACCOUNT, PASSWORD)
195         server.sendmail(EMAIL_ACCOUNT, FORWARD_TO_EMAIL, msg.as_string())
196 
```



```
197
198 if __name__ == '__main__':
199     check_email()
200
201     # Cria uma instância da classe Catalogo
202     catalogo = Catalogo()
203
204     # Executa o programa
205     catalogo.menu()
```

DESCRIÇÃO DO CÓDIGO PYTHON: 1. Este código Python é uma aplicação que gerencia um catálogo de itens e manipula e-mails. Ele é dividido em duas partes principais: a classe Catalogo para gerenciar itens e funções relacionadas ao envio e encaminhamento de e-mails. Abaixo está uma descrição detalhada de cada parte.

Classe Catalogo
A classe Catalogo permite adicionar, pesquisar, atualizar, exibir e remover itens de um catálogo, além de exibir um histórico de transações.

Métodos:

$_{i}nit_{(self):Inicializaaclassecomdoisarquivos CSV:umparaocatálogo(catalogo.csv)eoutroparaohistórico(historico.csv).}$
adicionar$_{i}tem(self)$: $Adicionaumnovoitemaocatálogopedindoaousuárioparainserircódigo, descr$

pesquisar$_{i}tem(self)$: $Pesquisaumitempelocódigonoarquivocatalogo.csveexibeadescriçãoeaquan$
atualizar$_{q}uantidade(self)$: $Atualizaaquantidadedisponíveldeumitemespecíficonocatálogo.Onov$
exibir$_{c}atalogo(self)$: $Exibetodosositensdocatálogo, mostrandocódigo, descriçãoequantidadedisp$
exibir$_{h}istorico(self)$: $Exibeohistóricodetransações, lendoosdadosdoarquivohistorico.csv.$

remover$_{i}tem(self)$: $Removeumitemdocatálogopelocódigo.Seoitemforencontrado, eleéremovido$

menu(self): Exibe um menu para o usuário selecionar diferentes opções (adicionar item, pesquisar item, atualizar quantidade, exibir catálogo, exibir histórico, remover item, sair).

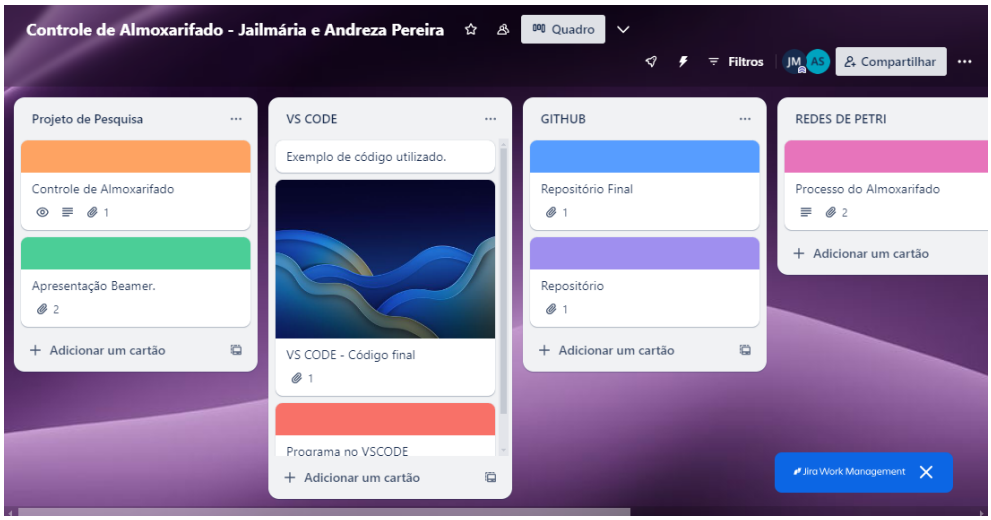
Manipulação de E-mails
Além da classe Catalogo, o código também configura a verificação de e-mails não lidos em uma conta de e-mail específica, e encaminha esses e-mails para outro endereço com um aviso adicional se a quantidade de algum item estiver abaixo de 30
Variáveis de Configuração de E-mail:
$IMAP_{S}ERVER : ServidorIMAP.$
$IMAP_{P}ORT : PortaIMAP.$
$SMTP_{S}ERVER : ServidorSMTP.$
$SMTP_{P}ORT : PortaSMTP.$
$EMAIL_{A}CCOUNT : Contadee - mailusadaparalogin.$
$PASSWORD : Senhadacontadee - mail.$
$FORWARD_{T}O_{E}MAIL : Endereçodee - mailparaencaminhamento.$
$AVISO : Mensagemdeavisoparaserincluídanocorpodoe - mailencaminhado.Funções : check_{e}mail$
forward$_{e}mail(original_{m}sg)$: $Criaumnovoe-mailqueincluioavisodefinidoeoconteúdodoe-mailoriginal, eoencaminhaparaoendereçodefinidoemFORWARD_{T}O_{E}MAIL.$

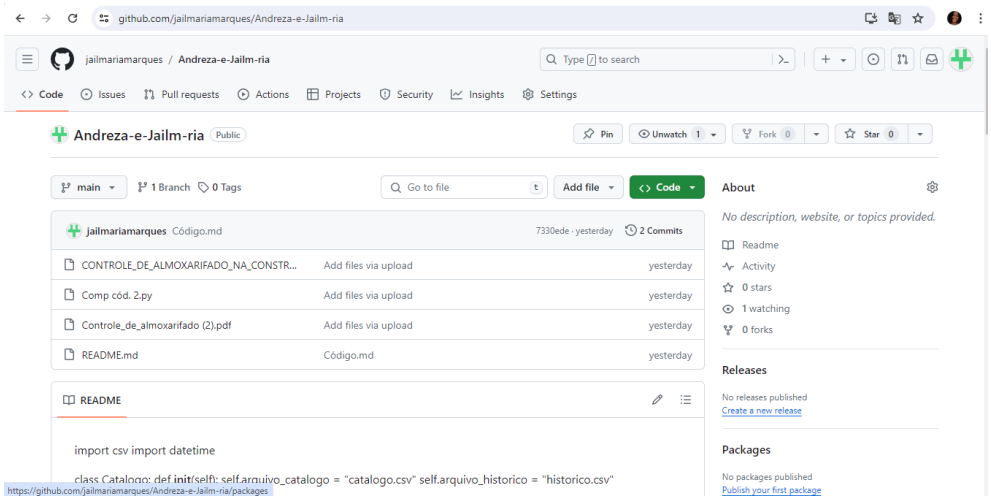
Execução do Programa No bloco if $_{name}{}_{==}{}'_{main}{}'_{:,ocódigo:}$

Verifica e-mails não lidos e os encaminha conforme necessário. Cria uma instância da classe Catalogo e executa o menu principal para permitir ao usuário gerenciar o catálogo de itens.

6 DESENVOLVIMENTO DO PROJETO (Quadros do Trello e Github)

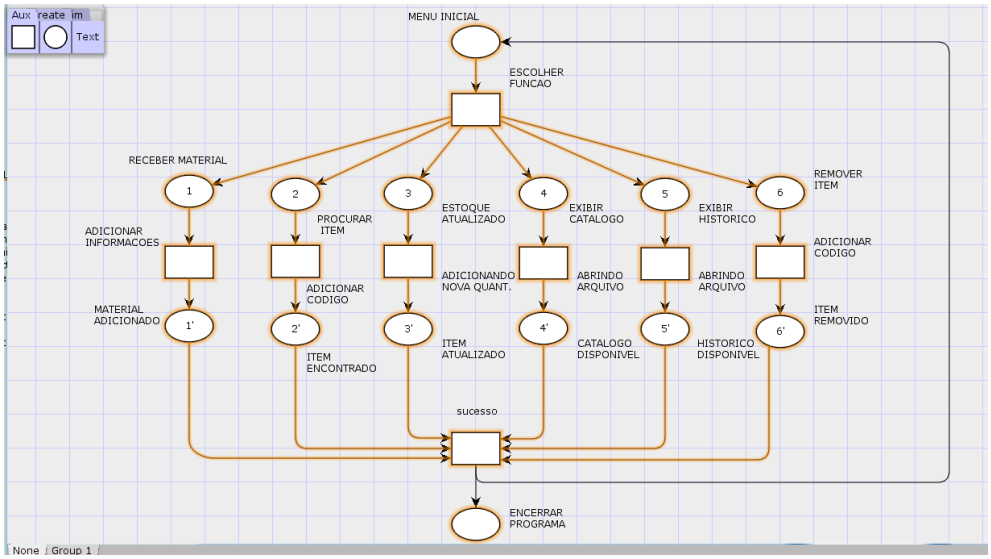
A ferramenta Trello foi utilizada para gerenciar o projeto de pesquisa de maneira eficiente e colaborativa. O projeto foi organizado em um quadro, dividido em listas que representavam as etapas do processo.Podiamos comentar nos cartões, anexar documentos relevantes e atualizar o status das tarefas, promovendo uma comunicação clara e acompanhamento em tempo real do progresso do projeto. Essa estrutura facilitou a visualização das atividades pendentes, a identificação de gargalos e a adaptação rápida às mudanças, garantindo um fluxo de trabalho mais eficiente e alinhado com os objetivos da pesquisa.
O GitHub foi fundamental para o gerenciamento e a colaboração no projeto de pesquisa, especialmente no controle de versões de documentos e códigos. Os membros da equipe criaram um repositório central onde todos os arquivos relacionados ao projeto eram armazenados e atualizados.Além disso, o uso de branches permitiu que diferentes partes do projeto fossem desenvolvidas em paralelo, sem interferência. Dessa forma, o GitHub não apenas organizou o fluxo de trabalho, mas também promoveu uma colaboração eficiente e estruturada, essencial para o sucesso do projeto de pesquisa.

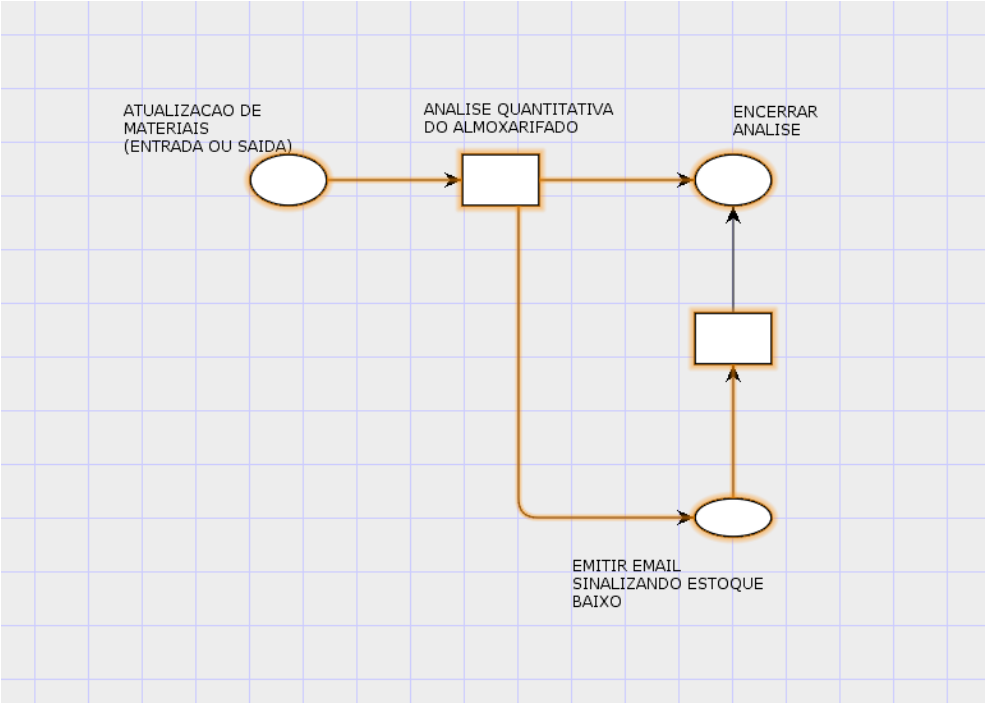




7 REDE DE PETRI

Além disso, a modelagem através de redes de Petri ofereceu uma representação clara e detalhada do fluxo de materiais, ajudando a identificar gargalos e a analisar a eficiência das operações. Com essas ferramentas e estratégias, é possível implementar um sistema de gestão de almoxarifados mais eficiente e sustentável, que não apenas resolve problemas imediatos, mas também contribui para o sucesso a longo prazo dos projetos na construção civil. Assim, este estudo apresenta uma base sólida para futuras melhorias e inovações na gestão de materiais nesse setor vital.





8 CONCLUSÃO

A gestão eficiente de almoxarifados na construção civil é um elemento crucial para assegurar a disponibilidade oportuna de materiais, minimizar desperdícios e aumentar a produtividade dos projetos. Este estudo aprofundou-se na análise das práticas atuais de gerenciamento de estoque, identificando desafios comuns e propondo melhorias que visam otimizar tanto o armazenamento quanto o controle de materiais. A adoção de tecnologias como a linguagem de programação Python e a rede de Petri mostrou-se promissora na automação e análise de processos, permitindo uma gestão mais precisa e eficiente.

Referências

[1] N. B. d. OLIVEIRA, B. S. d. SILVA, C. C. d. SOUZA, I. C. d. S. PEREIRA, and D. T. FRANCO, "Gestão de estoques: otimização no almoxarifado de empresas de materiais de construção," 2022.

[2] E. D. Sandrini, "Controle de almoxarifado através da otimização da gestão de estoques em uma construtora," B.S. thesis, Universidade Tecnológica Federal do Paraná, 2017.

[3]
(1) (2)
(3)