

Controle de Almoxarifado na Construção Civil

Jailmária Marques e Andreza Pereira.

Computação aplicada à engenharia.

1.Introdução

- ▶ O controle de almoxarifado desempenha um papel fundamental na eficiência e sucesso dos projetos na construção civil. No contexto da construção civil, o controle de almoxarifado refere-se ao processo de rastrear, organizar e controlar os materiais utilizados em uma obra.

2.Objetivos

- ▶ Objetivo Geral: analisar práticas de controle de almoxarifado e desenvolver um modelo referência com base nisso.
- ▶ Objetivos específicos: controle de entrada e saída de materiais, gestão de estoque e reposição facilitar a rastreabilidade e controle de validade, otimizar o tempo de obra.

3. Benefícios do Controle de Almoxarifado

- ▶ Redução de desperdícios;
- ▶ Melhor aproveitamento dos recursos;
- ▶ Evita paralisações de obra por falta de materiais;
- ▶ Planejamento mais eficiente;
- ▶ Redução de custos e aumento da lucratividade.

4. Metodologia de pesquisa

- ▶ A pesquisa foi realizada por meio de uma revisão bibliográfica que abordou tópicos como:
- ▶ Conceitos chave em relação à administração de almoxarifados;
- ▶ Práticas mais adequadas de organização e planejamento;
- ▶ Métodos recomendados de controle e estoque.

5.Evolução

- ▶ Nesse momento foi adicionado a possibilidade do nosso programa disparar emails automáticos para equipe de compras, informando um baixo estoque, dessa forma, aumentamos a integração entre os funcionários responsáveis pela gestão do almoxarifado e o time de engenharia montante.

6. Funcionamento do código Python

```
1  import csv
2  import datetime
3  import imaplib
4  import smtplib
5  from email.mime.multipart import MIMEMultipart
6  from email.mime.text import MIMEText
7  from email.parser import BytesParser
8  from email.policy import default
9
10
11 class Catalogo:
12     def __init__(self):
13         self.arquivo_catalogo = "catalogo.csv"
14         self.arquivo_historico = "historico.csv"
15
16     def adicionar_item(self):
17         codigo = input("Digite o código do item: ")
18         descricao = input("Digite a descrição do item: ")
19         quantidade = input("Digite a quantidade disponível: ")
20
21         with open(self.arquivo_catalogo, "a", newline="") as arquivo:
22             writer = csv.writer(arquivo)
23             writer.writerow([codigo, descricao, quantidade])
24
25         print("Item adicionado com sucesso!")
26
27     def pesquisar_item(self):
28         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)
```



```
56
57         with open(self.arquivo_catalogo, "w", newline="") as arquivo:
58             writer = csv.writer(arquivo)
59             writer.writerows(linhas_atualizadas)
60
61         print("Quantidade atualizada com sucesso!")
62     except FileNotFoundError:
63         print("Arquivo do catálogo não encontrado.")
64     except Exception as e:
65         print("Ocorreu um erro:", e)
66
67     def exibir_catalogo(self):
68         try:
69             with open(self.arquivo_catalogo, "r") as arquivo:
70                 reader = csv.reader(arquivo)
71                 for linha in reader:
72                     print("Código:", linha[0])
73                     print("Descrição:", linha[1])
74                     print("Quantidade disponível:", linha[2])
75                     print()
76             except FileNotFoundError:
77                 print("Arquivo do catálogo não encontrado.")
78             except Exception as e:
79                 print("Ocorreu um erro:", e)
80
81     def exibir_historico(self):
82         try:
83             with open(self.arquivo_historico, "r") as arquivo:
84                 reader = csv.reader(arquivo)
85                 for linha in reader:
```



```
115         writer.writerows(linhas_atualizadas)
116         print("Item removido com sucesso!")
117     else:
118         print("Item não encontrado.")
119 except FileNotFoundError:
120     print("Arquivo do catálogo não encontrado.")
121 except Exception as e:
122     print("Ocorreu um erro:", e)
123
124 def menu(self):
125     while True:
126         print("1. Adicionar item")
127         print("2. Pesquisar item")
128         print("3. Atualizar quantidade")
129         print("4. Exibir catálogo")
130         print("5. Exibir histórico")
131         print("6. Remover item")
132         print("7. Sair")
133
134         opcao = input("Digite a opção desejada: ")
135
136         if opcao == "1":
137             self.adicionar_item()
138         elif opcao == "2":
139             self.pesquisar_item()
140         elif opcao == "3":
141             self.atualizar_quantidade()
142         elif opcao == "4":
143             self.exibir_catálogo()
144         elif opcao == "5":
```

```

124     def menu(self):
125         while True:
126             self.exibir_catalogo()
127             elif opcao == "5":
128                 self.exibir_historico()
129             elif opcao == "6":
130                 self.remover_item()
131             elif opcao == "7":
132                 break
133             else:
134                 print("Opção inválida. Tente novamente.")
135
136         print()
137
138
139 # Configurações do e-mail
140 IMAP_SERVER = 'imap.exemplo.com'
141 IMAP_PORT = 993
142 SMTP_SERVER = 'smtp.exemplo.com'
143 SMTP_PORT = 587
144 EMAIL_ACCOUNT = 'seu_email@exemplo.com'
145 PASSWORD = 'sua_senha'
146 FORWARD_TO_EMAIL = 'destinatario@exemplo.com'
147 AVISO = "Aviso: Almoxarifado com capacidade menor que 30%"
148
149
150 def check_email():
151     mail = imaplib.IMAP4_SSL(IMAP_SERVER, IMAP_PORT)
152     mail.login(EMAIL_ACCOUNT, PASSWORD)
153     mail.select('inbox')
154     status, messages = mail.search(None, '(UNSEEN)')

```

```

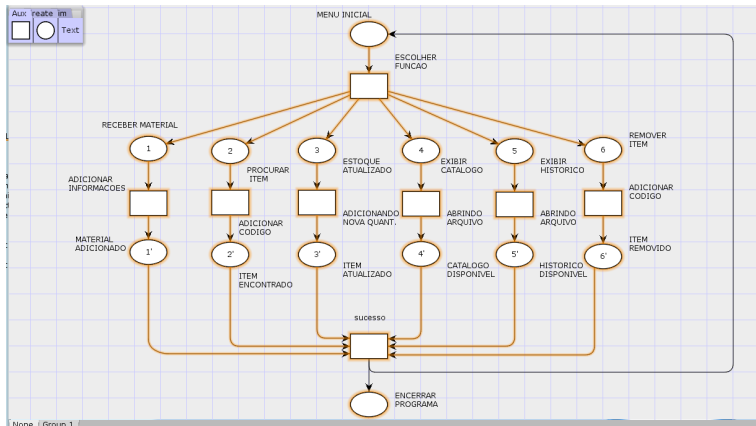
167 def check_email():
168     mail = imaplib.IMAP4_SSL(IMAP_SERVER, IMAP_PORT)
169     mail.login(EMAIL_ACCOUNT, PASSWORD)
170     mail.select('inbox')
171     status, messages = mail.search(None, '(UNSEEN)')
172     email_ids = messages[0].split()
173
174     for email_id in email_ids:
175         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

```

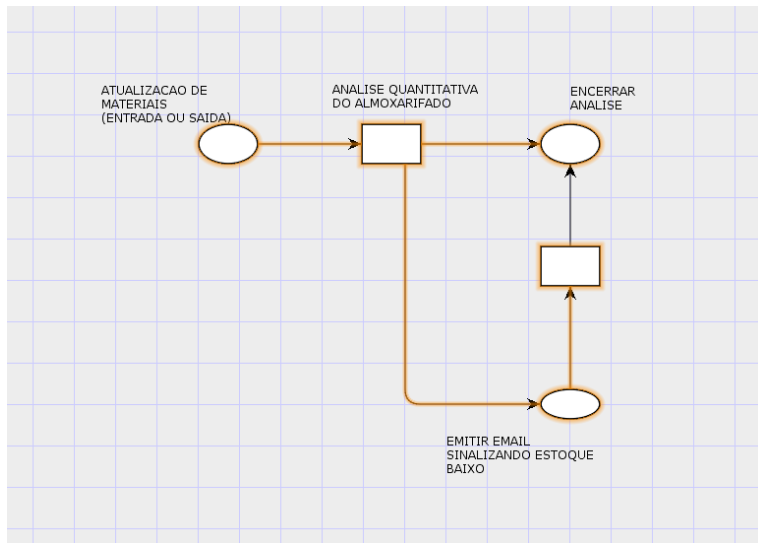
```
197
198 ∨ if __name__ == '__main__':
199     check_email()
200
201     # Cria uma instância da classe Catalogo
202     catalogo = Catalogo()
203
204     # Executa o programa
205     catalogo.menu()
```

7. Rede de Petri

- ▶ 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.



7. Rede de Petri



Conclusões Finais

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

- ▶ 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.
- ▶ 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