SYNEXIS
TECHNOLOGY

ESCOLA SENAI “AVAK BEDOUIAN”
TÉCNICO EM DESENVOLVIMENTO DE SISTEMAS

BIANCA LOQUETE
JOÃO PEDRO DE FREITAS MAFI
LETICIA MUNIZ FONTANETTI
MARIA EDUARDA SANCHES LIMA

FIXCLICK

SEUS PROBLEMAS RESOLVIDOS EM UM CLICK

BIRIGUI, 26 DE NOVEMBRO DE 2024

SYNEXIS
TECHNOLOGY

BIANCA LOQUETE
JOÃO PEDRO DE FREITAS MAFI
LETICIA MUNIZ FONTANETTI
MARIA EDUARDA SANCHES LIMA

FIXCLICK

SEUS PROBLEMAS RESOLVIDOS EM UM CLICK

Trabalho de conclusão de curso apresentado à Escola SENAI Avak Bedouian como requisito para obtenção do diploma de Técnico em Desenvolvimento de Sistemas.

BIRIGUI, 26 DE NOVEMBRO DE 2024

SYNEXIS
TECHNOLOGY

BIANCA LOQUETE
JOÃO PEDRO DE FREITAS MAFI
LETICIA MUNIZ FONTANETTI
MARIA EDUARDA SANCHES LIMA

FIXCLICK

SEUS PROBLEMAS RESOLVIDOS EM UM CLICK

Trabalho de conclusão de curso apresentado à Escola SENAI Avak Bedouian como requisito para obtenção do diploma de Técnico em Desenvolvimento de Sistemas.

Birigui, 26 de novembro de 2024

BANCA EXAMINADORA

Prof. Wellington - Coordenação - (SENAI)

Prof.

SYNEXIS
TECHNOLOGY

AGRADECIMENTOS

Dedico este trabalho a todas as pessoas que foram essenciais para a conclusão deste TCC.

Aos nossos queridos professores, Igor Cacerez e Lais Sinatra, cuja orientação e apoio foram fundamentais para o desenvolvimento deste projeto. A dedicação e o conhecimento de ambos foram verdadeiras fontes de inspiração.

À nossa família, pelo amor incondicional, paciência e encorajamento em cada passo desta nova jornada.

Aos amigos, que sempre estiveram prontos para oferecer suporte e conselhos valiosos. A amizade tornou o caminho mais leve e gratificante.

E, finalmente, ao ensino do SENAI Avak Bedouian, que proporcionou uma formação sólida e nos preparou para enfrentar os desafios da área de desenvolvimento de sistemas.

SYNEXIS
TECHNOLOGY

“A inovação é fomentada por informações coletadas de novas conexões, de insights obtidos por jornadas em outras disciplinas ou lugares, de redes ativas e colegiais e fronteiras fluidas e abertas. A inovação surge de círculos de troca contínuos, onde a informação não é apenas acumulada ou armazenada, mas criada. O conhecimento é gerado novamente a partir de conexões que não existiam antes.”

Margaret J. Wheatley

SYNEXIS
TECHNOLOGY

RESUMO

Este trabalho tem como objetivo apresentar e viabilizar uma plataforma online para a gestão de manutenções em escolas, com o intuito de otimizar o processo de identificação e correção de problemas nas instalações. O método de operação do aplicativo permite que funcionários identifiquem o local específico e registrem o problema no sistema. Assim, uma pessoa responsável pela manutenção recebe e se dirige ao local para resolver a questão. Este sistema visa melhorar a eficiência na gestão das manutenções escolares, assegurando um ambiente mais seguro e bem cuidado para todos.

PALAVRAS-CHAVES: Plataforma online, Gestão de manutenções, Escolas, Identificação de problemas.

ABSTRACT

The aim of this work is to present and make viable an online platform for managing maintenance in schools, with the aim of optimizing the process of identifying and correcting problems in facilities. The application's method of operation allows students and teachers to capture photos of the area in need of maintenance, identify the specific location and register the problem in the system. A maintenance person then receives the problem and goes to the site to resolve the issue. This system aims to improve efficiency in the management of school maintenance, ensuring a safer and more well-kept environment for everyone.

KEYWORDS: Online platform, Managing maintenance, Schools, Problem identification.

SYNEXIS
TECHNOLOGY

SUMÁRIO

INTRODUÇÃO

No Brasil, a gestão de manutenção dos aparelhos e da infraestrutura é uma questão de extrema importância. De acordo com a Confederação Nacional da Indústria (CNI), as instituições que investem regularmente em manutenção podem aumentar a vida útil de seus equipamentos em média de 20 a 30 anos, reduzindo os custos com trocas. Com o aumento do número de instituições educacionais, a demanda por soluções eficazes de manutenção se torna essencial.

A manutenção não só contribui para aumentar a vida útil dos equipamentos, mas também é importante para a segurança. O Anuário Estatístico de Acidentes de Trabalho, elaborado pelo Ministério da Economia, revela que cerca de 16% dos acidentes industriais estão relacionados a problemas de manutenção. Esses dados destacam a importância de um gerenciamento eficiente para garantir a segurança dos alunos e funcionários.

Os custos também são um fator crítico. Estudos mostram que práticas de manutenção reativa enfrentam custos de reparo significativamente mais altos do que aquelas que adotam abordagens preventivas ou preditivas. A manutenção preditiva, que utiliza dados e tecnologias avançadas para prever falhas antes que ocorram, pode reduzir custos. De acordo com a Associação Brasileira de Manutenção e Gestão de Ativos (ABRAMAN), a adoção de práticas preditivas e preventivas pode reduzir custos de manutenção em até 30% e aumentar a produtividade em até 25%.

Diante desse cenário, a FixClick se torna fundamental para a gestão dos recursos e a melhoria contínua das operações nas instituições educacionais. O projeto proposto visa desenvolver um sistema que permita aos professores registrarem e reportarem problemas de manutenção de forma organizada, e não mais através de imagens via WhatsApp. A solução buscará otimizar a comunicação entre os professores, o coordenador e a equipe de manutenção, garantindo uma resposta rápida e eficiente às necessidades de reparo, com a possibilidade de integrar dados preditivos para melhorar ainda mais a eficácia do sistema.

1.1 Objetivo

O objetivo deste projeto é desenvolver um sistema de manutenção que permita a gestão dos reparos necessários no SENAI. Ele permitirá que os professores registrem e selecionem a categoria de urgência do que necessita de manutenção, diretamente do seu celular, e essa informação irá para o chefe da manutenção, que designará o funcionário mais apropriado para realizar o reparo.

Isso facilitará a comunicação entre funcionários e coordenação, garantindo que a manutenção seja feita de forma mais ágil e organizada.

A FixClick tem como intuito otimizar a gestão e garantir que tudo seja consertado de forma rápida, facilitando o processo de comunicação da escola.

1.2 Justificativa

O interesse pela realização deste projeto surgiu ao vivenciar problemas diários na manutenção dos equipamentos, onde foi possível observar a importância de um software de gerenciamento de manutenção.

Com a realização de uma pesquisa de mercado, foi constatado que o mercado de manutenção tem se transformado, cada vez mais, em um setor importante para os resultados de uma empresa. Segundo um estudo da Plant Engineering Magazine de 2021, empresas que implementam programas de manutenção preventiva podem reduzir paradas não planejadas em até 30-50%. A manutenção preventiva garante que os equipamentos sejam inspecionados e ajustados regularmente, minimizando a probabilidade de falhas inesperadas.

Um software de gerenciamento de manutenção oferece uma plataforma para monitorar e controlar as atividades de manutenção, proporcionando vários benefícios, eficiência operacional, redução de custos e acesso em tempo real, contribuindo para o crescimento da empresa.

2. METODOLOGIA

Para o desenvolvimento da plataforma FixClick, foi adotada uma abordagem incremental, permitindo ajustes contínuos conforme as necessidades do sistema se tornavam mais claras. A seguir, descrevemos as ferramentas e processos utilizados durante o desenvolvimento da API, aplicação mobile e web, bem como a validação dos requisitos funcionais e não funcionais.

2.1 Ferramentas de Desenvolvimento

- Banco de Dados MySQL: o MySQL foi escolhido como sistema de gerenciamento de banco de dados relacional devido à sua robustez, capacidade de lidar com grandes volumes de dados e fácil integração com o Flask. O banco de dados armazena informações críticas sobre usuários, problemas de manutenção, locais, fotos e status dos reparos.
- Flask no PyCharm: para o desenvolvimento da API, foi utilizado o framework Flask no ambiente PyCharm. Flask é um framework leve e flexível, que nos permitiu construir uma API eficiente e escalável, facilitando a comunicação entre o front-end e o banco de dados. A API foi projetada para gerenciar o fluxo de dados entre as plataformas web e mobile, além de realizar operações de criação, leitura, atualização e exclusão (CRUD) de registros de manutenção.
- Expo Go: a parte mobile do projeto foi desenvolvida utilizando o Expo Go, um conjunto de ferramentas que agiliza o desenvolvimento de aplicativos React Native. Com essa tecnologia, foi possível garantir que os funcionários gerais e de manutenção pudessem usar o aplicativo de maneira eficiente para cadastrar e gerenciar problemas, respectivamente.
- WebStorm: para o desenvolvimento da interface web, utilizada exclusivamente pelo chefe de manutenção, o WebStorm foi adotado como ambiente de desenvolvimento. Esta ferramenta permitiu a criação de uma aplicação web responsiva, onde o supervisor pode designar problemas, monitorar status e gerenciar a equipe de manutenção.

2.2 Integração e Testes

Postman: todos os testes da API foram realizados utilizando o Postman, uma ferramenta amplamente usada para testar e desenvolver APIs. Cada funcionalidade da API foi exaustivamente testada para garantir que as requisições HTTP (GET, POST, PUT, DELETE) funcionassem de forma correta, assegurando que os dados fossem recebidos e armazenados no banco de dados MySQL de maneira precisa.

Exemplos de testes realizados:

- Teste de login e autenticação de usuários: testes para garantir que apenas usuários autorizados possam acessar o sistema.
- Designação de funcionários: testes para assegurar que o chefe de manutenção pode designar problemas a funcionários específicos.
- Atualização de status: validação do fluxo de atualização do status de problemas, de "pendente" para "em andamento" e "resolvido".

2.3 Desenvolvimento mobile

O desenvolvimento da versão mobile da FixClick foi dividido em duas partes principais:

- Aplicativo para funcionários gerais: permite que os usuários registrem problemas e insiram detalhes sobre a localização. Este aplicativo foi desenvolvido com React Native via Expo Go, aproveitando a facilidade de deployment em dispositivos móveis.
- Aplicativo para funcionários de manutenção: uma versão semelhante ao aplicativo geral, mas com funções adicionais para atualização de status e inserção de observações sobre os problemas designados.

2.4 Desenvolvimento web

A versão web da plataforma foi desenvolvida para uso exclusivo do chefe de manutenção, onde é possível:

- Visualizar todos os problemas cadastrados.
- Designar funcionários para tarefas específicas com base na urgência e tipo de problema.
- Acompanhar o progresso das manutenções e gerar relatórios de performance.
- A aplicação web foi construída utilizando React e integra-se diretamente com a API desenvolvida em Flask, permitindo uma comunicação eficiente com o banco de dados MySQL.

3. Requisitos funcionais

- Teste: O supervisor designar um profissional de manutenção para cada tipo de problema específico

  Passo: O supervisor receber uma queixa e conseguir direcionar um profissional especializado para o problema

  Verificação: o supervisor conseguir ver todas as queixas, ordenadas de maior grau de gravidade para o menor, com o tipo da queixa para que o profissional certo seja direcionado para o problema
- Teste: Enviar a queixa de um problema para que seja resolvido

  Passo: enviar a localização, marcar a seriedade e descrever o problema em uma queixa

  Verificação: verificar se a queixa foi direcionada corretamente para o supervisor de manutenção, para que o mesmo direcione profissionais específicos para cada tipo de problema.
- Teste: Marcar a seriedade do problema.

  Passo: identificar no aplicativo a seriedade do problema.

  Verificação: verificar se o problema vai ser identificado com uma cor específica para cada tipo de gravidade do problema.
- Teste: Fazer o login de usuário

  Passo: preencher os campos de email e login.

  Verificação: verificar se o usuário foi logado com sucesso no sistema.

4. Requisitos não funcionais

- Teste: Verificar a capacidade de resposta durante o envio de todas as informações.

  Passo: simular vários usuários enviando simultaneamente.

  Verificação: verificar se o aplicativo mantém um tempo de resposta aceitável e se o sistema não se torna instável ou lento durante picos de uso.
- Teste: Verificar a facilidade de uso do processo de envio.

  Passo: fazer o processo completo de informar a localização e enviar a solicitação.

  Verificação: verificar se o processo é intuitivo e se não há etapas confusas para o usuário.
- Teste: Avaliar a clareza das notificações recebidas pelo chefe de manutenção

  Passo: enviar uma solicitação de manutenção e observar a notificação recebida.

  Verificação:Verificar se a notificação é clara, contendo todas as informações necessárias, como localização e descrição do problema.
- Teste: Avaliar a proteção contra acesso não autorizado.

  Passo: acessar as informações de manutenção sem as permissões apropriadas.

  Verificação: verificar se o sistema impede o acesso não autorizado e que os dados estão protegidos adequadamente.
- Teste: Verificar a compatibilidade com diferentes dispositivos e sistemas operacionais.

  Passo: testar o aplicativo em diferentes dispositivos.

  Verificação: assegurar que o aplicativo funcione de maneira consistente e sem erros significativos em todas as plataformas.

5. DESCRIÇÃO DA API

5.1 App.py

```python
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from config import Config
from flask_cors import CORS


app = Flask(__name__)
app.config.from_object(Config)
db = SQLAlchemy(app)
CORS(app, resources={r"/*": {"origins": "*"}})

from view import *

if __name__ == '__main__':
    with app.app_context():
        app.run(debug=True)
    app.run(host='0.0.0.0', port=5000, debug=True)
```

- from flask import Flask: Importa a classe Flask, que é usada para criar a aplicação web.
- from flask_sqlalchemy import SQLAlchemy: Importa a classe SQLAlchemy que permite a integração com o banco de dados utilizando a ORM (Object-Relational Mapping), facilitando a interação com o banco de dados.
- from config import Config: Importa as configurações da aplicação.
- from flask_cors import CORS: Importa a funcionalidade CORS (Cross-Origin Resource Sharing) do Flask. Neste caso, o CORS está configurado para permitir acessos de qualquer origem.

5.2 Models.py

```
from app import db
from enum import Enum
from datetime import datetime
```

- from app import db: Importa a instância do banco de dados (db) que foi configurada no código anterior (com a variável db = SQLAlchemy(app)).
- from enum import Enum: Importa a classe Enum, que é usada para criar conjuntos de valores constantes. Isso é útil para campos onde você tem opções fixas, como cargos, prioridades e status.
- from datetime import datetime: Importa a classe datetime que é usada para trabalhar com datas e horários. Aqui, é usada para definir o valor padrão das colunas de data e hora.

```
class Cargo(Enum):
    funcionario_geral = "funcionario_geral"
    funcionario_manutencao = "funcionario_manutencao"
    chefe_manutencao = "chefe_manutencao"
```

- Cargo define os cargos dos funcionários. Cada cargo é um valor fixo:
  - funcionario_geral: Funcionário geral.
  - funcionario_manutencao: Funcionário de manutenção.
  - chefe_manutencao: Chefe de manutenção.

```
class Prioridade(Enum):
    Baixa = 'Baixa'
    Media = 'Media'
    Alta = 'Alta'
```

- Prioridade define os níveis de prioridade que um problema pode ter. Cada valor é uma string.
    - Baixa: Prioridade baixa.
    - Media: Prioridade média.
    - Alta: Prioridade alta.

```
class Status(Enum):
    Aberto = "Aberto"
    Em_andamento = "Em andamento"
    Fechado = "Fechado"
```

- Status define os possíveis status de um problema. Cada status é uma string:
    - Aberto: O problema ainda está aberto.
    - Em_andamento: O problema está sendo resolvido.
    - Fechado: O problema foi resolvido.

```python
class Funcionario(db.Model):

    id_funcionario = db.Column(db.Integer, primary_key=True)
    nome = db.Column(db.String(255), nullable=False)
    nif = db.Column(db.String(255), unique=True, nullable=False)
    senha = db.Column(db.String(255), nullable=False)
    cargo = db.Column(db.Enum(Cargo), nullable=False)

    problemas = db.relationship('Problema', backref='funcionario', lazy=True)
```

- Funcionario representa a tabela de funcionários.
    - id_funcionario: Identificador único do funcionário (chave primária).
    - nome: Nome do funcionário.
    - nif: Número de identificação fiscal (único e não nulo).
    - senha: Senha do funcionário.
    - cargo: Cargo do funcionário (utiliza o Enum Cargo para definir o valor).
    - problemas: Relacionamento com a tabela Problema. Isso significa que um funcionário pode estar associado a vários problemas (relacionamento um-para-muitos).

```python
class Materiais(db.Model):

    id_material = db.Column(db.Integer, primary_key=True)
    nome = db.Column(db.String(255), nullable=False)
    unidade = db.Column(db.String(255), nullable=False)
    quantidade_estoque = db.Column(db.Numeric(10, 2), nullable=False)
    valor = db.Column(db.Numeric(10, 2), nullable=False)
```

- Materiais representa a tabela de materiais..
    - nome: Nome do material.
    - unidade: Unidade de medida (como "kg", "litro", etc.).
    - quantidade_estoque: Quantidade disponível no estoque.
    - valor: Valor unitário do material.

```
class Problema(db.Model):
    caminho_imagem = db.Column(db.String(255), nullable=True)
    id_problema = db.Column(db.Integer, primary_key=True)
    descricao = db.Column(db.String(255), nullable=True)  # Agora é opcional
    local = db.Column(db.String(255), nullable=True)  # Agora é opcional
    prontuario = db.Column(db.Integer, nullable=True)  # Agora é opcional
    prioridade = db.Column(db.Enum(Prioridade), nullable=True)  # Agora é opcional
    status = db.Column(db.Enum(Status), default=Status.Aberto, nullable=False)  # Default é "Aberto"
    obs_funcionario = db.Column(db.String(255), nullable=True)  # Observações agora são opcionais
    datetime_inicio = db.Column(db.DateTime, default=datetime.now)
    datetime_finalizado = db.Column(db.DateTime, nullable=True)
    quanto_gastou = db.Column(db.Numeric(10, 2), nullable=True)
    id_funcionario = db.Column(db.Integer, db.ForeignKey('funcionario.id_funcionario'), nullable=True)  # Opcional
```

- Problema representa a tabela de problemas.
    - id_problema: Identificador único do problema (chave primária).
    - descricao: Descrição do problema.
    - local: Local onde o problema foi identificado.
    - prontuario: Número do prontuário (pode ser opcional).
    - prioridade: Prioridade do problema (utiliza o Enum Prioridade).
    - status: Status do problema (utiliza o Enum Status), com valor padrão "Aberto".
    - obs_funcionario: Observações feitas pelo funcionário.
    - datetime_inicio: Data e hora em que o problema foi registrado.
    - datetime_finalizado: Data e hora em que o problema foi finalizado (se aplicável).
    - quanto_gastou: Custo total associado ao problema (caso tenha gastos).
    - id_funcionario: Relacionamento com a tabela Funcionario (funcionário responsável). Esse campo é uma chave estrangeira (ForeignKey), que referencia o id_funcionario da tabela Funcionario.

```python
class ProblemaMaterial(db.Model):

    id_problema_material = db.Column(db.Integer, primary_key=True)
    id_funcionario = db.Column(db.Integer, db.ForeignKey('funcionario.id_funcionario'))
    id_problema = db.Column(db.Integer, db.ForeignKey('problema.id_problema'))
    id_material = db.Column(db.Integer, db.ForeignKey('materiais.id_material'))
    quantidade = db.Column(db.Integer)
    valor_total = db.Column(db.Numeric(10, 2))

    funcionario = db.relationship('Funcionario', backref='problema_material', lazy=True)
    problema = db.relationship('Problema', backref='problema_material', lazy=True)
    material = db.relationship('Materiais', backref='problema_material', lazy=True)
```

- ProblemaMaterial representa a tabela de relacionamento entre problemas e materiais.
    - id_funcionario: Chave estrangeira que referencia o id_funcionario da tabela Funcionario.
    - id_problema: Chave estrangeira que referencia o id_problema da tabela Problema.
    - id_material: Chave estrangeira que referencia o id_material da tabela Materiais.
    - quantidade: Quantidade do material utilizada no problema.
    - valor_total: Valor total gasto com o material no problema.
    - As relações de Funcionario, Problema e Materiais são representadas através do db.relationship, permitindo acessar facilmente os dados relacionados.

5.3 View.py

```
from flask import request, jsonify
from werkzeug.security import check_password_hash, generate_password_hash
from app import app, db
from models import Funcionario, Problema, Materiais, ProblemaMaterial, Cargo, Prioridade, Status
from datetime import datetime
import os
import uuid
```

- Flask: Utilizado para criar as rotas e manipular as requisições HTTP.
- werkzeug.security: Para criptografar e verificar senhas de usuários.
- app, db: A instância da aplicação Flask e a instância do SQLAlchemy para interagir com o banco de dados.
- models: Importação dos modelos definidos anteriormente (Funcionario, Problema, Materiais, etc.).
- datetime, os, uuid: Para lidar com datas e gerar identificadores únicos (UUID) e manipulação de arquivos.

```python
@app.route('/funcionario', methods=['POST'])
def cadastrar_funcionario():
    data = request.get_json()
    nome = data.get('nome')
    nif = data.get('nif')
    senha = data.get('senha')
    cargo = data.get('cargo')

    if Funcionario.query.filter_by(nif=nif).first():
        return jsonify(mensagem="Funcionário com este NIF já está cadastrado!"), 400

    try:
        cargo_enum = Cargo[cargo]
        novo_funcionario = Funcionario(
            nome=nome,
            nif=nif,
            senha=generate_password_hash(senha),
            cargo=cargo_enum
        )
        db.session.add(novo_funcionario)
        db.session.commit()
        return jsonify(mensagem="Funcionário cadastrado com sucesso!"), 201

    except KeyError:
        cargos_validos = [cargo.name for cargo in Cargo]
        return jsonify(mensagem=f"Cargo inválido! Cargos válidos são: {', '.join(cargos_validos)}"), 400
    except Exception as e:
        db.session.rollback()
        return jsonify(mensagem=f"Erro ao salvar: {str(e)}"), 500
```

- Cadastro de Funcionário (/funcionario, método POST)
- Validação de NIF: Verifica se o NIF fornecido já está cadastrado.
- Tratamento de erros: Verifica se o cargo informado é válido, com base no Enum Cargo. Caso contrário, retorna uma mensagem de erro.

```python
@app.route('/funcionario', methods=['GET'])
def listar_funcionarios():
    nif = request.args.get('nif')
    if nif:
        funcionario = Funcionario.query.filter_by(nif=nif).first()
        if funcionario:
            return jsonify({
                'id_funcionario': funcionario.id_funcionario,
                'nome': funcionario.nome,
                'nif': funcionario.nif,
                'cargo': funcionario.cargo.name
            }), 200
        else:
            return jsonify(mensagem="Funcionário não encontrado!"), 404
    else:
        funcionarios = Funcionario.query.all()
        return jsonify(funcionarios=[{
            'id_funcionario': f.id_funcionario,
            'nome': f.nome,
            'nif': f.nif,
            'cargo': f.cargo.name
        } for f in funcionarios]), 200
```

- Busca por NIF: Se um NIF for fornecido como parâmetro de consulta, busca um funcionário específico.
- Caso Não Fornecido NIF: Caso contrário, retorna todos os funcionários cadastrados.
- Tratamento de Erros: Se o funcionário não for encontrado, retorna um erro 404.
- Objetivo: Exibir a lista de todos os funcionários ou um funcionário específico.

```
@app.route('/funcionario/<int:id_funcionario>', methods=['PUT'])
def atualizar_funcionario(id_funcionario):
    data = request.get_json()
    funcionario = Funcionario.query.get(id_funcionario)

    if not funcionario:
        return jsonify(mensagem="Funcionário não encontrado!"), 404

    try:
        nome = data.get('nome', funcionario.nome)
        nif = data.get('nif', funcionario.nif)
        cargo = data.get('cargo', funcionario.cargo.name)

        # Verificar se o NIF está sendo alterado e se já existe outro funcionário com esse NIF
        if Funcionario.query.filter_by(nif=nif).first() and nif != funcionario.nif:
            return jsonify(mensagem="Já existe um funcionário com este NIF!"), 400

        cargo_enum = Cargo[cargo]

        funcionario.nome = nome
        funcionario.nif = nif
        funcionario.senha = generate_password_hash(data.get('senha', funcionario.senha))
        funcionario.cargo = cargo_enum


    db.session.commit()
    return jsonify(mensagem="Funcionário atualizado com sucesso!"), 200
except KeyError:
    cargos_validos = [cargo.name for cargo in Cargo]
    return jsonify(mensagem=f"Cargo inválido! Cargos válidos são: {', '.join(cargos_validos)}"), 400
except Exception as e:
    db.session.rollback()
    return jsonify(mensagem=f"Erro ao atualizar: {str(e)}"), 500
```

- Validação de NIF: Verifica se o NIF informado já está em uso por outro funcionário, garantindo que não haja duplicidade.
- Validação de Cargo: O cargo é validado com base no Enum Cargo. Se o cargo for inválido, é retornada uma mensagem de erro com os cargos válidos.
- Objetivo: Atualizar os dados de um funcionário existente, como nome, NIF, cargo e senha.

```python
@app.route('/funcionario/<int:id_funcionario>', methods=['DELETE'])
def deletar_funcionario(id_funcionario):
    funcionario = Funcionario.query.get(id_funcionario)

    if not funcionario:
        return jsonify(mensagem="Funcionário não encontrado!"), 404

    try:
        db.session.delete(funcionario)
        db.session.commit()
        return jsonify(mensagem="Funcionário deletado com sucesso!"), 200
    except Exception as e:
        db.session.rollback()
        return jsonify(mensagem=f"Erro ao deletar: {str(e)}"), 500
```

- Validação de Existência: Verifica se o id_funcionario informado corresponde a um funcionário existente no banco de dados.
- Tratamento de Erros: Se o funcionário não for encontrado, retorna erro 404. Se ocorrer um erro ao tentar deletar o funcionário, retorna erro 500.
- Objetivo: Excluir um funcionário do sistema.

```python
@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    nif = data.get('nif')
    senha = data.get('senha')

    user = Funcionario.query.filter_by(nif=nif).first()

    if user and check_password_hash(user.senha, senha):
        response_message = "Login bem-sucedido!"
        if user.cargo == Cargo.funcionario_geral:
            return jsonify(mensagem=f"{response_message} Redirecionando para a página de visualização."), 200
        elif user.cargo == Cargo.funcionario_manutencao:
            return jsonify(mensagem=f"{response_message} Redirecionando para a página de problemas designados."), 200
        elif user.cargo == Cargo.chefe_manutencao:
            return jsonify(mensagem=f"{response_message} Redirecionando para a tela de designação."), 200
        else:
            return jsonify(mensagem="Cargo inválido!"), 400
    else:
        return jsonify(mensagem="NIF ou senha incorretos!"), 400
```

- Validação de NIF e Senha: Verifica se o NIF e senha fornecidos são válidos, buscando no banco de dados e verificando se a senha está correta.
- Tratamento de Erros: Se o NIF ou a senha estiverem incorretos, um erro 400 é retornado. Caso o cargo seja inválido, é retornada uma mensagem com o erro apropriado.
- Objetivo: Realizar a autenticação do funcionário no sistema, redirecionando para a interface correspondente ao cargo.

E todos esses métodos (POST, PUT, GET E DELETE) ocorrem com todas as classes já citadas acima:

- Funcionários;
- Materiais;
- Problema_Material.

6. Conclusão

A gestão eficiente de manutenção é fundamental para garantir o bom funcionamento das instituições educacionais, além de promover a segurança e reduzir custos com reparos. No caso do SENAI, o sistema FixClick proposto neste trabalho apresenta uma solução inovadora para melhorar a comunicação entre funcionários, coordenadores e a equipe de manutenção. Com ele, a solicitação de reparos se torna mais ágil e organizada, permitindo que os problemas sejam registrados e classificados de forma clara.

O sistema proposto visa otimizar o processo de reparo, tornando-o mais eficiente e reduzindo o tempo de resposta, o que impacta diretamente na continuidade das atividades educacionais. Além disso, a centralização das informações em uma plataforma digital elimina o uso de canais informais, como o WhatsApp, que muitas vezes geram confusão e perda de dados importantes.

Outro ponto relevante do FixClick é a possibilidade de integrar tecnologias preditivas no futuro, o que permitiria antecipar falhas antes que ocorram, evitando custos adicionais com reparos inesperados e contribuindo para a prolongação da vida útil dos equipamentos. A manutenção preditiva, quando combinada com a gestão ágil proporcionada pelo sistema, pode gerar uma economia significativa de recursos, além de melhorar a eficiência das operações da instituição.

Portanto, a implementação dessa plataforma no SENAI representa uma solução eficiente para os desafios enfrentados na gestão de manutenção, proporcionando uma abordagem mais moderna e integrada para o processo. Ao promover uma comunicação mais eficiente, redução de custos e maior segurança, o sistema pode servir como um modelo para outras instituições educacionais, ampliando o impacto positivo na gestão de recursos e na continuidade das atividades pedagógicas.

7. Referências bibliográficas

- PLANT ENGINEERING MAGAZINE. Redução de Paradas Não Planejadas. Plant Engineering Magazine, 2021. Dados obtidos de estudo publicado pela revista, com foco na manutenção preventiva e redução de falhas inesperadas. Acesso em: 27 de agosto de 2024.
- SOCIETY FOR MAINTENANCE AND RELIABILITY PROFESSIONALS (SMRP). Aumento da Vida Útil dos Equipamentos. SMRP, 2021. Estudo sobre o impacto da manutenção estratégica na vida útil de equipamentos industriais. Acesso em: 27 de agosto de 2024.
- HEALTH AND SAFETY EXECUTIVE (HSE). Impacto na Segurança do Trabalho. Relatório sobre acidentes industriais e falhas devido à manutenção inadequada, divulgado pelo HSE, 2020. Acesso em: 27 de agosto 2024.
- DELOITTE. Efeito na Produtividade e Moral dos Funcionários. Deloitte Insights, 2021. Pesquisa sobre o impacto da manutenção em ambientes de trabalho no aumento da moral e produtividade dos funcionários. Acesso em: 27 de agosto de 2024.
- ABERDEEN GROUP. Custos Operacionais e de Reparos. Estudo comparando custos de manutenção reativa versus preventiva, Aberdeen Group, 2020. Acesso em: 27 de agosto de 2024.
- AMERICAN PRODUCTIVITY & QUALITY CENTER (APQC). Impacto Financeiro. Pesquisa sobre economias em custos operacionais com programas de manutenção robustos, APQC, 2021. Acesso em: 27 de agosto de 2024.
- ASSOCIAÇÃO BRASILEIRA DE MANUTENÇÃO E GESTÃO DE ATIVOS (ABRAMAN). Redução de Custos e Aumento da Produtividade. *ABRAMAN*, 2020. Dados sobre os benefícios da manutenção preventiva e preditiva na redução de custos e aumento da produtividade, segundo pesquisa realizada pela ABRAMAN. Acesso em: 27 de agosto de 2024.

- MINISTÉRIO DA ECONOMIA. Anuário Estatístico de Acidentes de Trabalho. Relatório sobre acidentes de trabalho no Brasil e sua relação com falhas de manutenção, Ministério da Economia, 2019. Acesso em: 27 de agosto de 2024.
- CONFEDERAÇÃO NACIONAL DA INDÚSTRIA (CNI). Vida Útil dos Equipamentos. Estudo da CNI sobre a prolongação da vida útil de equipamentos industriais em empresas que investem regularmente em manutenção, CNI, 2020. Acesso em: 27 de agosto de 2024.
- OPERADOR NACIONAL DO SISTEMA ELÉTRICO (ONS) e AGÊNCIA NACIONAL DE ENERGIA ELÉTRICA (ANEEL). Impacto no Setor Elétrico. Estudo sobre falhas de manutenção no setor elétrico brasileiro e sua relação com desligamentos não programados, ONS e ANEEL, 2021. Acesso em: 27 de agosto de 2024.
- ASSOCIAÇÃO BRASILEIRA DA INDÚSTRIA DE MÁQUINAS E EQUIPAMENTOS (ABIMAQ). Impacto na Indústria Automotiva. Pesquisa sobre os benefícios da manutenção preditiva na indústria automotiva, ABIMAQ, 2021. Acesso em: 27 de agosto de 2024.
- INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA (IBGE). Redução de Desperdícios. Relatório do IBGE sobre o impacto da manutenção na redução de desperdícios no setor industrial brasileiro, IBGE, 2020. Acesso em: 27 de agosto de 2024.

SYNEXIS
TECHNOLOGY

SYNEXIS
TECHNOLOGY