
**Universidade do Estado da Bahia**
**Departamento de Ciências Exatas e da Terra**
**Curso de Bacharelado em Engenharia de Producao Civil**

**Discentes: Guilherme Souza e Sara Costa**

# Gerenciador de tarefas para engenharia civil

Salvador
2024

# Sumário

# Lista de Figuras



# Lista de Tabelas

# 1 Introducao

## 1.1 Objetivo

Criar um Gerenciador de tarefas que forneca informacões que circulam entre a administracao de uma empresa e os funcionarios. Fornecendo relatórios das tarefas.

## 1.2 Metodologia

Para o embasamento téorico e uma melhor compreensao do tema, foram utilizadas vídeos aulas e pesquisas acadêmicas que pudessem fomentar o trabalho. Aumentando as fontes e elevando o nível do projeto final.

## 1.3 Motivacao e justificativa

Pensando no futuro, onde se deseja um ambiente produtivo e organizado, teve-se como ideia, a criacao de um programa em python que fosse capaz gerenciar tarefas basicas de uma obra da construcao civil, buscando proporcionar uma maior organizacao nas tarefas que deverao ser executadas.

# 2 Referencial Teórico

## 2.1 Bibliotecas

As principais bibliotecas utilizadas foram: FastAPI, Uvicorn, Requests, Pydantic.

### 2.1.1 FastAPI

Segundo [1], "O FastAPI é um framework Python focado no desenvolvimento de API's, tem como principais características ser moderno, rapido e simples".

### 2.1.2 Uvicorn

Segundo [2], o uvicorn é utilizado para carregar e servir a aplicacao.

### 2.1.3 Requests

Segundo [3], a biblioteca requests é o padrao para fazer solicitacões HTTP em Python. Ele abstrai as complexidades de fazer solicitacões por tras de uma API simples para que você possa se concentrar na interacao com servicos e no consumo de dados em seu aplicativo.

### 2.1.4 PySimpleGUI

Segundo [4], "Criada por Mike Campbell, o "PySimpleGUI" é uma biblioteca "Python" que permite com que programadores de todos os níveis de conhecimento possam criar suas codificações em interfaces gráficas".

### 2.1.5 Pydantic

Segundo [5], "O Pydantic é uma biblioteca Python de código aberto que oferece uma maneira simples e elegante de validar dados. Ele foi criado por Samuel Colvin e é amplamente utilizado na comunidade Python para validar dados em aplicativos web, APIs, analise de dados e muito mais. Uma das características mais marcantes do Pydantic é a sua facilidade de uso e a integracao perfeita com outros componentes Python".

## 2.2 SQL

A Linguagem de consulta estruturada (SQL) é uma linguagem de consultas estruturadas para armazenar e processar informacões em um banco de dados relacional. Um banco de dados relacional armazena informacões em formato tabular, com linhas e colunas representando diferentes atributos de dados e as varias relacões entre os valores dos dados.

## 2.3 SQLite

Uma das principais vantagens do SQLite é que ele nao requer um servi dor separado para funcionar. Isso significa que o banco de dados SQLite é armazenado em um único arquivo, o que torna a sua manipulacao e distribuicao extremamente facil. Além disso, o SQLite é conhecido por sua alta confiabilidade e estabilidade, garantindo a integridade dos dados mesmo em situacões de falha ou desligamento inesperado do sistema.

## 2.4 Orientacao a objetos

Uma classe é representada por atributos e métodos. Os atributos de uma classe representam as características que esta classe possui, ja os métodos representam o comportamento da classe. Sempre que precisamos criar "algo" com base em uma classe, dizemos que estamos "instanciando objetos". O ato de instanciar um objeto significa que estamos criando a representacao de uma classe em nosso programa. Para instanciar um objeto no Python com base em uma classe previamente declarada, basta indicar a classe que desejamos utilizar como base e, caso possua, informar os valores referentes aos seus atributos.

Segundo [6], "Classes proporcionam uma forma de organizar dados e funcionalidades juntos. Criar uma nova classe cria um novo "tipo" de objeto, permitindo que novas "instâncias" desse tipo sejam produzidas. Cada instância da classe pode ter atributos anexados a ela, para manter seu estado. Instâncias da classe também podem ter métodos (definidos pela classe) para modificar seu estado".

# 3 Proposta do Programa

## 3.1 Evolucao do programa

Antes o programa era capaz apenas de criar tarefas localmente sem que fosse possível que outras pessoas tivessem acesso a essa tarefa. Agora é possível que o administrador crie as tarefas e delegue essas tarefas aos funcionarios, que também conseguem visualizar essas atividades e criar relatórios as mesmas.

## 3.2 Funcionamento do programa

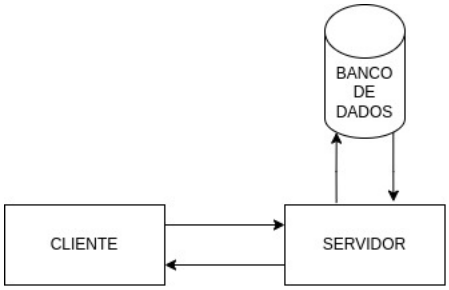


Figura 1: Funcionamento do programa

O programa funciona com base na ideia de client-server. O cliente faz requisicões a um servidor, que por sua vez busca dados em banco de dados, e devolve uma resposta ao cliente. Esta abordagem garante uma gestao mais eficiente e escalavel dos recursos e servicos, além de permiti a centralizacao das informacões.

## 3.3 Banco de dados

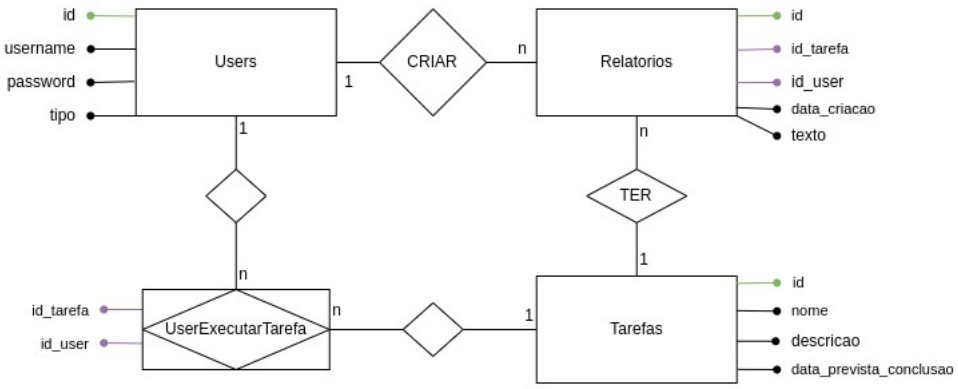


Figura 2: Diagrama entidade relacionamento

A figura 2 representa o banco de dados, que possui 4 entidades ou tabelas Users, Relatorios, Tarefas e UserExecutarTarefa. Cada tarefa pode ter múltiplos relatórios associados, que documentam seu progresso ou problemas encontrados. Relatórios sao criados por usuarios e associados a tarefas específicas. Além disso, os usuarios podem ser responsaveis pela execucao de varias tarefas, conforme registrado na entidade UserExecutarTarefa.

## 3.4 Redes de Petri

### 3.4.1 Rede de Petri Ordinaria

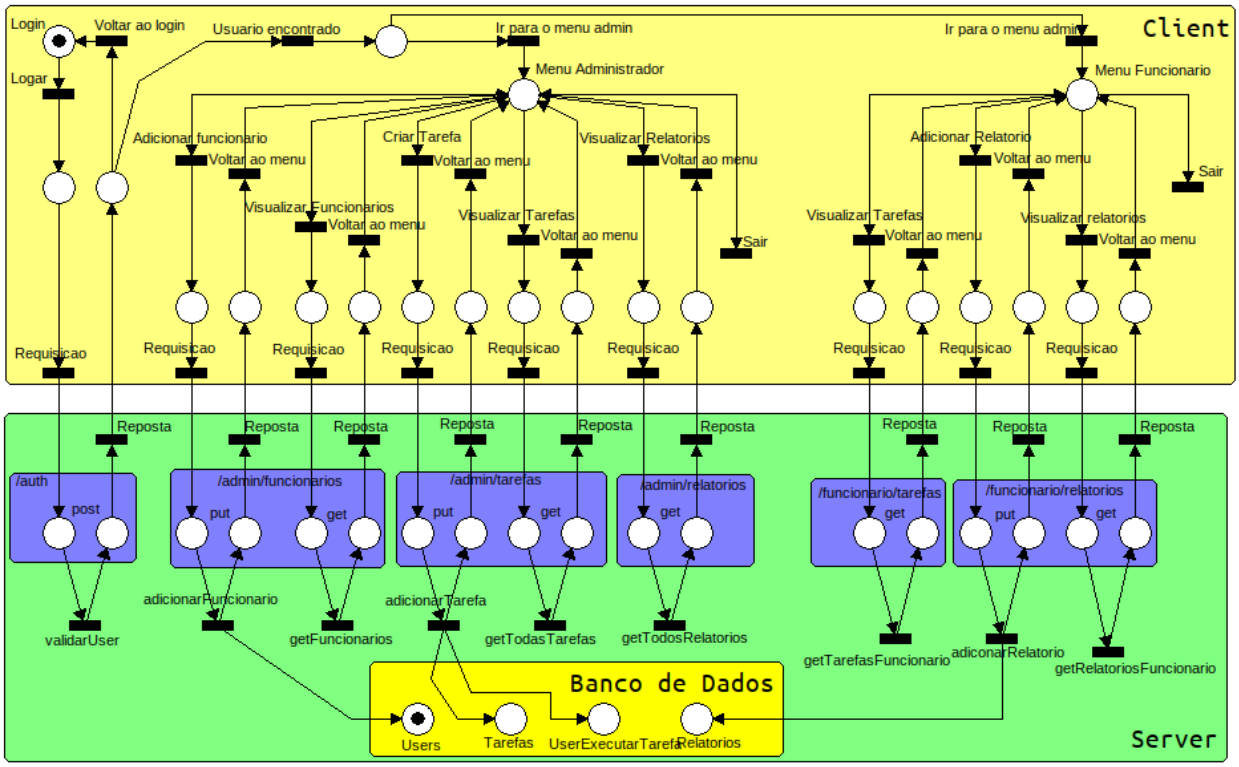


Figura 3: Rede de Petri Ordinaria

### 3.4.2 Rede de Petri Colorida

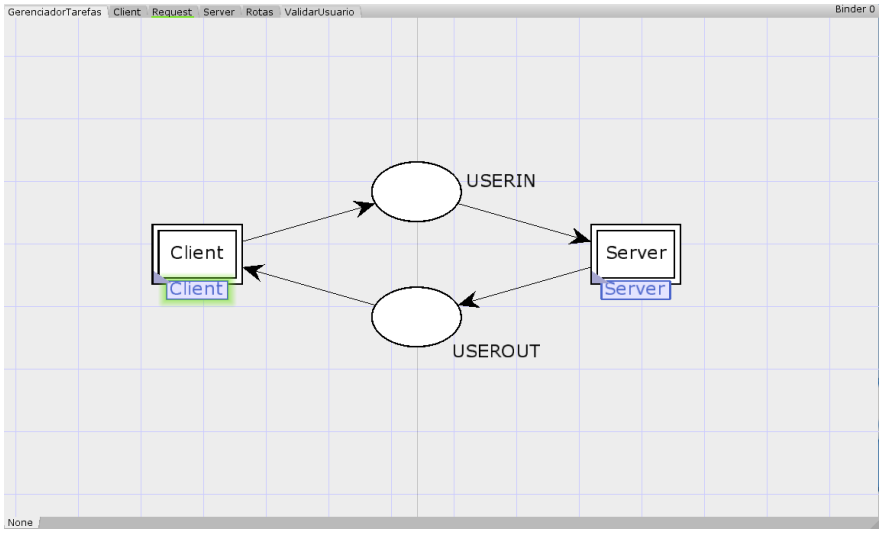


Figura 4: Visao geral do projeto

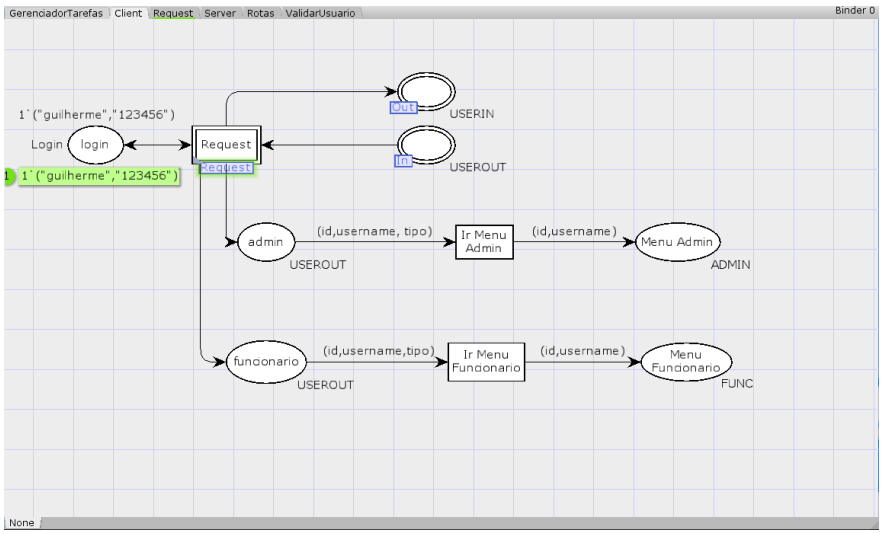


Figura 5: Cliente

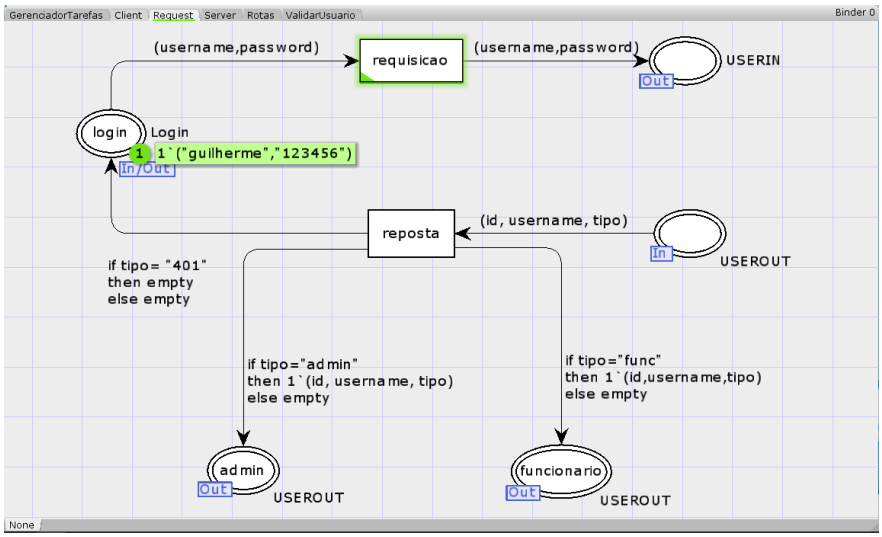


Figura 6: Requisicao para o servidor e resposta

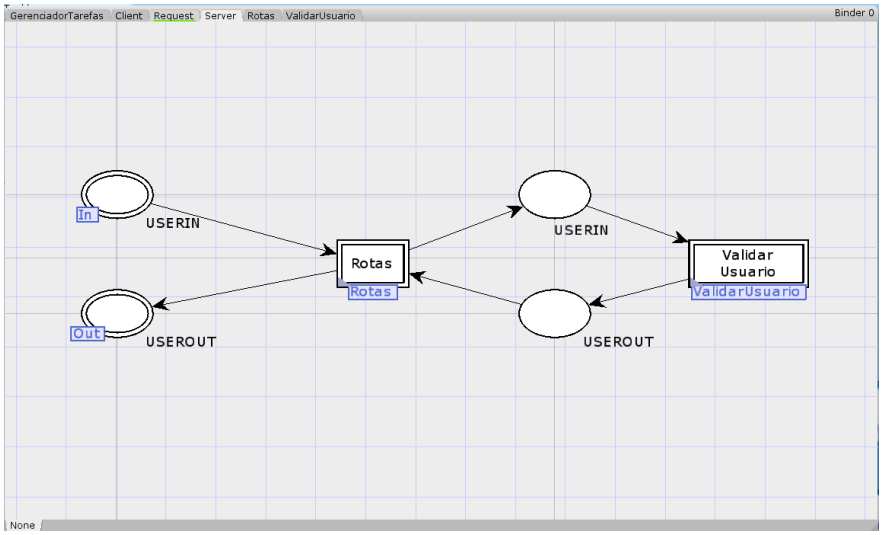


Figura 7: Servidor

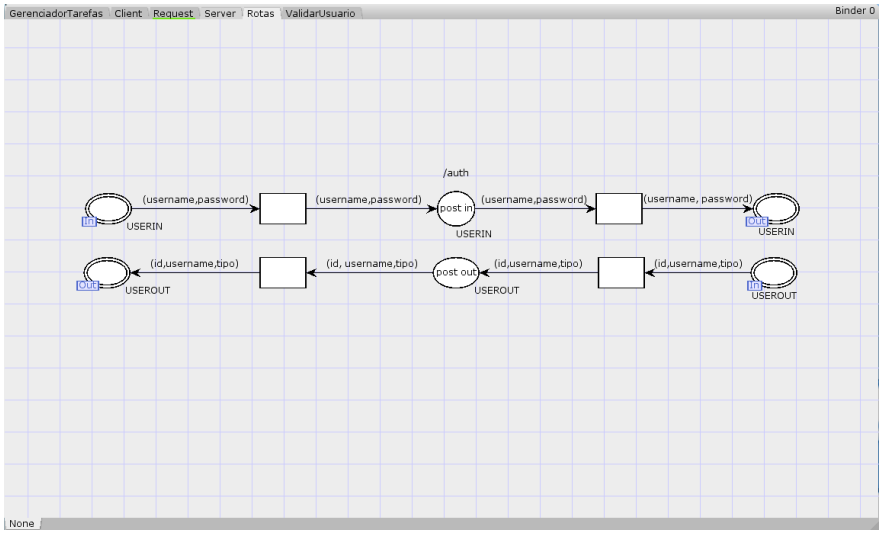


Figura 8: Rota de autenticacao que recebe o nome de usuario e senha, e envia a resposta para o cliente

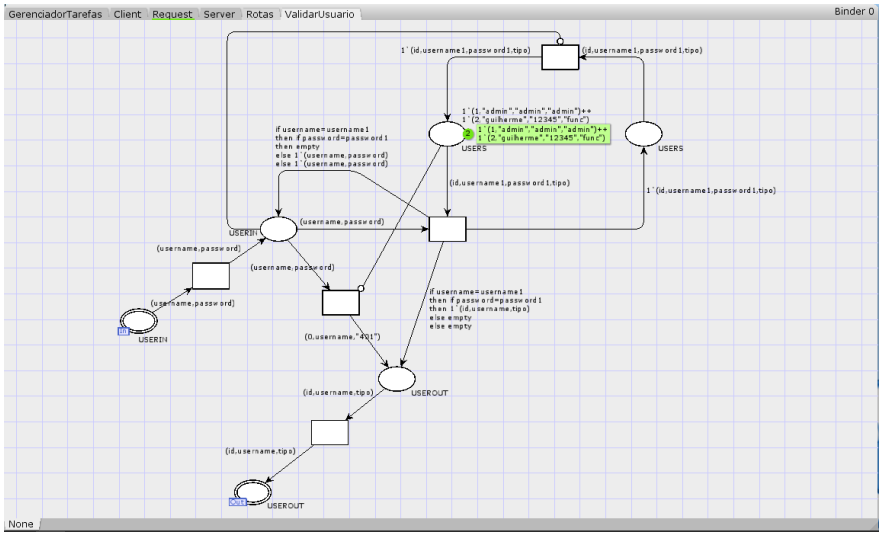


Figura 9: Recebe usuario e senha e valida se o usuario esta no sistema

# 4 Código do programa

## 4.1 Servidor

```
from fastapi import FastAPI, HTTPException, status
import uvicorn
import jwt
from datetime import datetime, timedelta, timezone
from contextlib import asynccontextmanager
from src import database, user_database, tarefa_database,
    relatorio_database
from src.models.user_model import UserIn
from src.models.tarefa_model import Tarefa
from src.models.relatorio_model import Relatorio

app = FastAPI()

```

```python
token_secret = "12345"

# inicializa o banco de dados de forma assincrona
@asynccontextmanager
async def lifespan(app: FastAPI):
    await database.inicializarBancoDados()
    yield

# Executa a funcao lifespan junto com a inicializacao do servidor
app.router.lifespan_context = lifespan

# Rota de autenticao. Recebe usuario e senha do client e valida se usuario esta cadastrado no banco de dados
@app.post("/auth")
async def autenticar_user(user: UserIn):
    validarUser = user_database.validarUser(user=user)

    if validarUser.get("status"):
        token = jwt.encode({
            'exp': datetime.now(timezone.utc) + timedelta(minutes=30),
            "nivel-acesso": validarUser.get("user").tipo,
            "id": validarUser.get("user").id
        }, key=token_secret, algorithm='HS256')

        validarUser.get("user").token = token

        return validarUser.get("user")
    else:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Usuario nao encontrado!"
        )

# Rota para adicionar funcionario, recebe nome de usuario e senha
@app.put("/admin/funcionarios")
async def adicionar_funcionario(func: UserIn, token: str):
    try:
        token_deco = jwt.decode(token, key=token_secret, algorithms="HS256")

        if token_deco.get("nivel-acesso") == 'admin':
            if user_database.adicionarFuncionario(func=func):
                return "Funcionario Adicionado com Sucesso!"
            else:
                raise HTTPException(
                    status_code=status.HTTP_501_NOT_IMPLEMENTED,
                    detail="Erro ao adicionar novo funcionario!"
                )
        else:
            raise PermissionError
    except jwt.DecodeError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token invalido!"
        )
    except jwt.ExpiredSignatureError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token Expirado!"
        )
```

```
    except PermissionError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Permissao Negada!"
        )

# Rota que lista todos os funcionario cadastrados no banco de dados
@app.get("/admin/funcionarios")
async def lista_funcionarios(token: str):
    try:
        token_deco = jwt.decode(token, key=token_secret, algorithms="HS256
    ")

        if token_deco.get("nivel-acesso") == 'admin':
            return user_database.getFuncionarios()
        else:
            raise PermissionError
    except jwt.DecodeError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token invalido!"
        )
    except jwt.ExpiredSignatureError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token Expirado!"
        )
    except PermissionError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Permissao Negada!"
        )

# Rota que adiciona uma nova tarefa do banco de dados. Recebe nome,
    descricao, data prevista de conclusao e id dos funcionarios
@app.put("/admin/tarefas")
async def adicionar_tarefa(tarefa: Tarefa, token: str):
    try:
        token_deco = jwt.decode(token, key=token_secret, algorithms="HS256
    ")

        if token_deco.get("nivel-acesso") == 'admin':
            if tarefa_database.adicionarTarefa(tarefa=tarefa):
                return "Tarefa adicionada com Sucesso!"
            else:
                raise HTTPException(
                    status_code=status.HTTP_501_NOT_IMPLEMENTED,
                    detail="Erro ao adicionar tarefa!"
                )
        else:
            raise PermissionError
    except jwt.DecodeError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token invalido!"
        )
    except jwt.ExpiredSignatureError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token Expirado!"
```

```
        )
    except PermissionError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Permissao Negada!"
        )

# Rota que lista todas as tarefas cadastradas no sistema
@app.get("/admin/tarefas")
async def get_todas_tarefas(token: str):
    try:
        token_deco = jwt.decode(token, key=token_secret, algorithms="HS256
    ")

        if token_deco.get("nivel-acesso") == 'admin':
            return tarefa_database.getTodasTarefas()
        else:
            raise PermissionError
    except jwt.DecodeError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token invalido!"
        )
    except jwt.ExpiredSignatureError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token Expirado!"
        )
    except PermissionError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Permissao Negada!"
        )

# Rota que lista todos os relatorios cadastrados no sistema
@app.get("/admin/relatorios")
async def get_todos_relatorios(token: str):
    try:
        token_deco = jwt.decode(token, key=token_secret, algorithms="HS256
    ")

        if token_deco.get("nivel-acesso") == 'admin':
            return relatorio_database.getTodosRelatorios()
        else:
            raise PermissionError
    except jwt.DecodeError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token invalido!"
        )
    except jwt.ExpiredSignatureError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token Expirado!"
        )
    except PermissionError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Permissao Negada!"
        )
```

```python

# Rota que lista todas as tarefas do funcionario. Recebe o id do usuario
@app.get("/funcionario/tarefas")
async def get_funcionario_tarefas(token: str):
    try:
        token_deco = jwt.decode(token, key=token_secret, algorithms="HS256")

        if token_deco.get("nivel-acesso") == 'funcionario':
            return tarefa_database.getTarefasFuncionario(idUser=token_deco.get("id"))
        else:
            raise PermissionError
    except jwt.DecodeError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token invalido!"
        )
    except jwt.ExpiredSignatureError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token Expirado!"
        )
    except PermissionError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Permissao Negada!"
        )

# Rota que adiciona um novo relatorio. Recebe id do usuario, id da tarefa, texto, data de criacao
@app.put("/funcionario/relatorios")
async def adicionar_relatorio(relatorio: Relatorio, token: str):
    try:
        token_deco = jwt.decode(token, key=token_secret, algorithms="HS256")

        if token_deco.get("nivel-acesso") == 'funcionario':
            if relatorio_database.adicionarRelatorio(relatorio=relatorio):
                return "Relatorio adicionado com sucesso!"
            else:
                raise HTTPException(
                    status_code=status.HTTP_501_NOT_IMPLEMENTED,
                    detail="Erro ao adicionar tarefa!"
                )
        else:
            raise PermissionError
    except jwt.DecodeError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token invalido!"
        )
    except jwt.ExpiredSignatureError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token Expirado!"
        )
    except PermissionError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
```

```
            detail="Permissao Negada!"
        )

# Rota que lista todos relatorios cadastrados pelo funcionario
@app.get("/funcionario/relatorios")
async def relatorios_cadastrados(token: str):
    try:
        token_deco = jwt.decode(token, key=token_secret, algorithms="HS256")

        if token_deco.get("nivel-acesso") == 'funcionario':
            return relatorio_database.getRelatoriosFuncionario(idUser=token_deco.get("id"))
        else:
            raise PermissionError
    except jwt.DecodeError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token invalido!"
        )
    except jwt.ExpiredSignatureError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token Expirado!"
        )
    except PermissionError:
        return HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Permissao Negada!"
        )

# Inicializa o servidor se o arquivo for o principal
if __name__ == "__main__":
    uvicorn.run(app=app)
```

```
import sqlite3

# Cria a conexao com banco de dados
def conexaoBancoDados() -> sqlite3.Connection:

    caminhoBd = "./server/database/gerenciador-tarefas.db"

    conn = sqlite3.connect(caminhoBd)

    return conn

# Inicializa banco de dados
async def inicializarBancoDados() -> None:
    conn = conexaoBancoDados()
    cursor = conn.cursor()

    cursor.execute('''CREATE TABLE IF NOT EXISTS Users(
                    id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
                    username TEXT NOT NULL UNIQUE,
                    password TEXT,
                    tipo TEXT CHECK(tipo IN ('admin', 'funcionario')) NOT NULL DEFAULT 'funcionario'
    );''')

    cursor.execute('''CREATE TABLE IF NOT EXISTS Tarefas(
                    id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
```

```
                    nome TEXT NOT NULL UNIQUE,
                    descricao TEXT NOT NULL,
                    data_prevista_conclusao TEXT
    );''')

    cursor.execute('''CREATE TABLE IF NOT EXISTS Relatorios(
                    id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
                    id_user INTEGER NOT NULL,
                    id_tarefa INTEGER NOT NULL,
                    data_criacao TEXT NOT NULL,
                    texto TEXT,
                    FOREIGN KEY(id_user) REFERENCES Users(id),
                    FOREIGN KEY(id_tarefa) REFERENCES Tarefas(id)
    );''')

    cursor.execute('''CREATE TABLE IF NOT EXISTS UserExecutarTarefa(
                    id_user INTEGER NOT NULL,
                    id_tarefa INTEGER NOT NULL,
                    FOREIGN KEY(id_user) REFERENCES Users(id),
                    FOREIGN KEY(id_tarefa) REFERENCES Tarefas(id)
    );''')

    cursor.execute("SELECT * FROM Users WHERE username = 'admin';")
    query = cursor.fetchone()

    if query == None:
        cursor.execute('''INSERT INTO Users (username, password, tipo)
    VALUES ('admin', 'admin', 'admin');''')

    conn.commit()
    print("Conexao com banco de dados bem sucedida!")
    conn.close()
```

```
from src.models.tarefa_model import Tarefa
from src.database import conexaoBancoDados

# Adiciona uma nova tarefa no banco de dados. Recebe como argumento um
    objeto Tarefa
def adicionarTarefa(tarefa: Tarefa) -> bool:
    try:
        conn = conexaoBancoDados()
        cursor = conn.cursor()

        cursor.execute("""INSERT INTO Tarefas
                    (nome, descricao, data_prevista_conclusao) VALUES
                    (?, ?, ?);
                    """, (tarefa.nome, tarefa.descricao, tarefa.
    data_prevista_conclusao))

        cursor.execute("""SELECT id FROM Tarefas
                        WHERE nome=?;""", (tarefa.nome,))

        id_tarefa = cursor.fetchone()[0]

        for func in tarefa.funcionariosId:
            cursor.execute("""INSERT INTO UserExecutarTarefa
                        (id_user, id_tarefa) VALUES
                        (?, ?);""", (func, id_tarefa))

        conn.commit()
```

```
        conn.close()
        return True
    except Exception as err:
        print(err)
        return False
# Retorna uma lista de dicionarios com todas as tarefas cadastradas no banco de dados
def getTodasTarefas() -> list:
    conn = conexaoBancoDados()
    cursor = conn.cursor()
    tarefasId = {}
    tarefas = []
    cursor.execute("""SELECT * FROM UserExecutarTarefa;""")
    relacaoUserTarefa = cursor.fetchall()

    for rel in relacaoUserTarefa:
        if rel[1] in tarefasId:
            tarefasId[rel[1]].append(rel[0])
        else:
            tarefasId[rel[1]] = [rel[0]]
    c = 0
    for key, value in tarefasId.items():
        cursor.execute("""SELECT nome, descricao, data_prevista_conclusao FROM Tarefas
                          WHERE id = ?;""", (key,))
        tarefa = cursor.fetchone()

        tarefas.append({
            "id": key,
            "nome": tarefa[0],
            "descricao": tarefa[1],
            "data_prevista_conclusao": tarefa[2],
            "funcionarios": []
        })

        for funcId in value:
            cursor.execute("""SELECT username FROM Users
                              WHERE id = ?;""", (funcId,))
            funcName = cursor.fetchone()[0]
            tarefas[c]["funcionarios"].append(funcName)
        c += 1
    conn.close()
    return tarefas
# Retorna uma lista de dicionario com todas as tarefas do funcionario. Recebe o id do funcionario como argumento
def getTarefasFuncionario(idUser: int) -> list:
    conn = conexaoBancoDados()
    cursor = conn.cursor()
    cursor.execute("""SELECT id_tarefa FROM UserExecutarTarefa
                      WHERE id_user = ?;""", (idUser,))
    idTarefas = cursor.fetchall()
    tarefas = []
    c = 0
    while c < len(idTarefas):
        cursor.execute("""SELECT * FROM Tarefas
                          WHERE id = ?;""", (idTarefas[c][0],))
        tarefa = cursor.fetchall()
        tarefas.append({
            "id": tarefa[0][0],
            "nome": tarefa[0][1],
```

```
            "descricao": tarefa[0][2],
            "data_prevista_conclusao": tarefa[0][3]
        })
        c += 1
    conn.close()
    return tarefas
```

```
from src.database import conexaoBancoDados
from src.models.relatorio_model import Relatorio

# Adiciona um novo relatorio no banco de dados. Recebe um objeto Relatorio como argumento
def adicionarRelatorio(relatorio: Relatorio) -> bool:
    try:
        conn = conexaoBancoDados()
        cursor = conn.cursor()

        cursor.execute("""INSERT INTO Relatorios
                    (id_user, id_tarefa, data_criacao, texto) VALUES
                    (?, ?, ?, ?);
                    """, (relatorio.id_user, relatorio.id_tarefa, relatorio.data_criacao, relatorio.texto))

        conn.commit()
        conn.close()

        return True
    except Exception as err:
        print(err)
        return False

# Retorna uma lista de dicionarios com os relatorios do funcionario. Recebe o id do funcionario como argumento
def getRelatoriosFuncionario(idUser: int) -> list:
    conn = conexaoBancoDados()
    cursor = conn.cursor()

    cursor.execute("""SELECT id, id_tarefa, data_criacao, texto FROM Relatorios
                   WHERE id_user = ?;""", (idUser,))

    relatorios = []
    queryRelatorios = cursor.fetchall()

    for relatorio in queryRelatorios:
        cursor.execute("""SELECT nome FROM Tarefas
                       WHERE id = ?;""", (relatorio[1],))
        nomeTarefa = cursor.fetchone()[0]

        relatorios.append({
            "id": relatorio[0],
            "id_tarefa": relatorio[1],
            "nome_tarefa": nomeTarefa,
            "data_criacao": relatorio[2],
            "texto": relatorio[3]
        })

    conn.close()

    return relatorios

```

```python
# Retorna uma lista de dicionarios com todos os relatorios cadastrados no sistema
def getTodosRelatorios() -> list:
    conn = conexaoBancoDados()
    cursor = conn.cursor()

    cursor.execute("""SELECT id, id_user, id_tarefa, data_criacao, texto FROM Relatorios;""")
    query = cursor.fetchall()

    relatorios = []

    for relatorio in query:
        cursor.execute("""SELECT username FROM Users
                       WHERE id = ?;""", (relatorio[1],))
        nomeFunc = cursor.fetchone()[0]

        cursor.execute("""SELECT nome FROM Tarefas
                       WHERE id = ?;""", (relatorio[2],))
        nomeTarefa = cursor.fetchone()[0]

        relatorios.append({
            "id": relatorio[0],
            "id_user": relatorio[1],
            "id_tarefa": relatorio[2],
            "nome_funcionario": nomeFunc,
            "nome_tarefa": nomeTarefa,
            "data_criacao": relatorio[3],
            "texto": relatorio[4]
        })

    conn.close()

    return relatorios
```

```python
from src.database import conexaoBancoDados
from src.models.user_model import UserIn, UserOut

# Valida o usuario no banco de dados. Recebe como argumento um objeto UserIn
def validarUser(user: UserIn) -> bool:
    conn = conexaoBancoDados()
    cursor = conn.cursor()

    cursor.execute("""SELECT id, username, password, tipo FROM Users
                   WHERE username = ? AND password = ?;""", (user.username, user.password))

    query = cursor.fetchone()
    conn.close()

    if query != None:
        return {
            "status": True,
            "user": UserOut(id=query[0], username=query[1], tipo=query[3])
    }

    return {
            "status": False
        }

```

```python
# Adiciona um funcionario no banco de dados. Recebe um objeto UserIn
def adicionarFuncionario(func: UserIn) -> bool:
    try:
        conn = conexaoBancoDados()
        cursor = conn.cursor()

        cursor.execute("""INSERT INTO Users (username, password, tipo) VALUES
                       (?, ?, 'funcionario');""", (func.username, func.password))

        conn.commit()
        conn.close()
        return True
    except Exception as err:
        print(err)
        return False

# Retorna uma lista de dicionarios com todos os funcionarios
def getFuncionarios() -> list:
    conn = conexaoBancoDados()
    cursor = conn.cursor()
    cursor.execute("""SELECT id, username FROM Users
                   WHERE tipo='funcionario';""")

    query = cursor.fetchall()
    conn.close()
    funcionarios = []
    for func in query:
        funcionarios.append({
                "id": func[0],
                "nome": func[1]
            })
    return funcionarios

```

```python
from pydantic import BaseModel

# Classe que gerencia a entrada dos dados do usuario
class UserIn(BaseModel):
username: str
password: str

# Classe que gerencia a saida dos dados do usuario
class UserOut(BaseModel):
id: int
username: str
tipo: str

```

Listing 1: user_model.py

```python
from pydantic import BaseModel

#Classe que gerencia a entrada de uma nova tarefa
class Tarefa(BaseModel):
nome: str
descricao: str
data_prevista_conclusao: str
funcionariosId: list

```

Listing 2: tarefa_model.py

```python
from pydantic import BaseModel

# Classe que gerencia a entrada de um novo relatorio
class Relatorio(BaseModel):
  id_user: int
  id_tarefa: int
  texto: str
  data_criacao: str

```

Listing 3: relatorio_model.py

## 4.2 Cliente

```python
import requests
import PySimpleGUI as sg
import json
from view import funcionario_view
from view import admin_view
from view import windows

url = "http://127.0.0.1:8000"

def main():
    window_login = windows.login_window()
    window_menu_admin = None
    window_menu_func = None
    window_admin_adicionar_func = None
    window_admin_visualizar_func = None
    window_admin_adicionar_tarefa= None
    window_admin_visualizar_tarefas = None
    window_admin_visualizar_relatorios = None
    window_func_visualizar_tarefas = None
    window_func_adicionar_relatorio = None
    window_func_visualizar_relatorios = None

    while True:
        window,event,values = sg.read_all_windows()

        if window == window_login and event == sg.WIN_CLOSED:
            break
        if window == window_login and event== "Continuar":

            username = window_login["username"].get()
            password = window_login["password"].get()
            statusLogin= login(username,password)

            if statusLogin.get("status") == True:
                window_login.hide()
                if statusLogin.get("tipo") == "admin":
                    window_menu_admin = windows.menu_admin_window()
                else:
                    window_menu_func = windows.menu_func_window()
            else:
                window_login["incorrect_text"].update(visible=True)
        if (window==window_menu_admin or window==window_menu_func) and (event==sg.WIN_CLOSED or event=="Sair"):
            window.close()
            window_menu_admin = None
            window_menu_func = None
            window_login.un_hide()
            window_login["incorrect_text"].update(visible=False)
            window_login["username"].SetFocus()
            window_login["username"].update("")
            window_login["password"].Update("")
        if event == sg.WIN_CLOSED or event== "Voltar":
            if window_menu_admin != None:
                window_menu_admin.un_hide()
            elif window_menu_func != None:
                window_menu_func.un_hide()
            window.close()
            window_admin_adicionar_func = None
            window_admin_visualizar_func = None
            window_admin_adicionar_tarefa= None
```

```
            window_admin_visualizar_tarefas = None
            window_admin_visualizar_relatorios = None
            window_func_visualizar_tarefas = None
            window_func_adicionar_relatorio = None
            window_func_visualizar_relatorios = None

        if window == window_menu_admin and event == "Adicionar Funcionario
    ":
            window_menu_admin.hide()
            window_admin_adicionar_func = windows.adicionar_func_window()

        if window == window_menu_admin and event == "Visualizar
    Funcionarios":
            window_menu_admin.hide()
            window_admin_visualizar_func = windows.visualizar_funcs_window
    ()

        if window == window_menu_admin and event == "Criar Tarefa":
            window_menu_admin.hide()
            window_admin_adicionar_tarefa,numFuncionarios,
    listaFuncionarios = windows.adicionar_tarefa_window()

        if window == window_menu_admin and event == "Visualizar Tarefas":
            window_menu_admin.hide()
            window_admin_visualizar_tarefas = windows.
    visualizar_tarefas_window()

        if window == window_menu_admin and event == "Visualizar Relatorios
    ":
            window_menu_admin.hide()
            window_admin_visualizar_relatorios = windows.
    visualizar_relatorios_window()

        if window == window_menu_func and event == "Visualizar Tarefas":
            window_menu_func.hide()
            window_func_visualizar_tarefas = windows.
    visualizar_tarefas_funcionario_window()

        if window == window_menu_func and event == "Adicionar Relatorio":
            window_menu_func.hide()
            window_func_adicionar_relatorio,numTarefas, listasTarefas =
    windows.adicionar_relatorio_window()

        if window == window_menu_func and event == "Visualizar Relatorios"
    :
            window_menu_func.hide()
            window_func_visualizar_relatorios = windows.
    visualizar_relatorios_window()

        if event == "Cadastrar":
            status=None
            if  window== window_admin_adicionar_func:
                nomeFunc=values["nomeFunc"]
                senhaFunc=values["senhaFunc"]
                status = admin_view.adicionarFuncionario(nomeFunc,
    senhaFunc)
            elif window == window_admin_adicionar_tarefa:
                nomeTarefa = values["nomeTarefa"]
                descricao = values["descricao"]
                data_prevista_conclusao = values["data_prevista_conclusao"
```

```
    ]
                funcionariosId = []
                i = 0
                while i < numFuncionarios:
                    if values[f"{i}"]:
                        funcionariosId.append(listaFuncionarios[i].get("id
"))
                    i+=1
                status=admin_view.criarTarefa(nomeTarefa,descricao,
data_prevista_conclusao,funcionariosId)
            elif window==window_func_adicionar_relatorio:
                textoRelatorio=values["texto_relatorio"]
                idTarefa=[]
                i=0
                while i<numTarefas:
                    if values[f"{1}"]:
                        idTarefa.append(listasTarefas[i].get("id"))
                    i+=1
                status=funcionario_view.adicionarRelatorio(idTarefa[0],
textoRelatorio)
            if status:
                window["status_true"].update(visible=True)
            else:
                window["status_false"].update(visible=True)
    window_login.close()

def login(username,password):

    request = requests.post(url+"/auth", json={"username": username, "
password": password})

    if request.status_code == 200:
        # Abre o arquivo user.json para escrita
        file = open("user.json", "w")
        # Escreve os dados da resposta do servidor no arquivo user.json
        json.dump(request.json(), file)
        # Fecha o arquivo
        file.close()
        if request.json()["tipo"] == "admin":
            return {
                "status": True,
                "tipo":"admin"
            }
        elif request.json()["tipo"] == "funcionario":
            return{
                "status": True,
                "tipo":"funcionario"
            }
    else:
        return{
            "status": False
        }

if __name__ == "__main__":
    main()
```

Listing 4: app.py

```
import PySimpleGUI as sg
from view import admin_view
from view import funcionario_view
```

```
4
5  def login_window():
6      sg.theme('GreenTan')
7      layout = [
8          [sg.Text("Nome de usuario")],
9          [sg.Input(key="username")],
10         [sg.Text("Senha")],
11         [sg.Input(password_char="*",key="password")],
12         [sg.Button("Continuar")],
13         [sg.Text("Usuario ou Senha Incorretos!", text_color="red",visible=
       False, key="incorrect_text")]
14     ]
15
16     return sg.Window("Login", layout=layout, finalize=True)
17
18 def menu_admin_window():
19     sg.theme('GreenTan')
20     layout = [
21         [sg.Button("Adicionar Funcionario")],
22         [sg.Button("Visualizar Funcionarios")],
23         [sg.Button("Criar Tarefa")],
24         [sg.Button("Visualizar Tarefas")],
25         [sg.Button("Visualizar Relatorios")],
26         [sg.Button("Sair")]
27     ]
28
29     return sg.Window("Menu Admin", layout=layout, finalize=True)
30
31 def menu_func_window():
32     sg.theme('GreenTan')
33     layout = [
34         [sg.Button("Visualizar Tarefas")],
35         [sg.Button("Adicionar Relatorio")],
36         [sg.Button("Visualizar Relatorios")],
37         [sg.Button("Sair")]
38     ]
39
40     return sg.Window("Menu Funcionario", layout=layout, finalize=True)
41
42 def adicionar_func_window():
43     sg.theme('GreenTan')
44     layout = [
45         [sg.Text("Nome do funcionario")],
46         [sg.Input(key="nomeFunc")],
47         [sg.Text("Senha do funcionario")],
48         [sg.Input(key="senhaFunc")],
49         [sg.Text("Relatorio cadastrado com sucesso!", text_color="white",
       background_color="green", visible=False, key="status_true")],
50         [sg.Text("Erro ao cadastrar o relatorio! \nTente Novamente!",
       text_color="white", background_color="red", visible=False, key="
       status_false")],
51         [sg.Button("Cadastrar")],
52         [sg.Button("Voltar")]
53     ]
54
55     return sg.Window("Adicionar Funcionario", layout=layout, finalize=True
       )
56
57 def adicionar_tarefa_window():
58     sg.theme('GreenTan')
```

```
    layout = [
        [sg.Text("Nome da Tarefa")],
        [sg.Input(key="nomeTarefa")],
        [sg.Text("Descricao da tarefa")],
        [sg.Input(key="descricao")],
        [sg.Text("Data Prevista de Conclusao (dd-mm-aa)")],
        [sg.Input(key="data_prevista_conclusao")],
        [sg.Text("Selecione os Funcionarios:")]
    ]

    listaFuncionarios = admin_view.visualizarFuncionarios()
    c = 0
    column = []

    if listaFuncionarios == []:
        column.append([sg.Text("Sem funcionarios cadastrados!")])
    else:
        for funcionario in listaFuncionarios:
            column.append([sg.Checkbox(funcionario["nome"],key="{}".format(c))])
            c += 1

    layout.append([sg.Column(column, size=(200, 200), scrollable=True, vertical_scroll_only=True)])

    layout.append([sg.Text("Tarefa cadastrado com sucesso!", text_color="white", background_color="green", visible=False, key="status_true")])

    layout.append([sg.Text("Erro ao cadastrar a tarefa! \nTente Novamente!", text_color="white", background_color="red", visible=False, key="status_false")])

    layout.append([sg.Button("Cadastrar")])

    layout.append([sg.Button("Voltar")])

    return sg.Window("Criar Tarefa", layout=layout, finalize=True), c, listaFuncionarios

def visualizar_funcs_window():
    sg.theme('GreenTan')
    layout = [
        [sg.Text("Funcionarios: ")],
    ]

    listaFuncionarios = admin_view.visualizarFuncionarios()

    if listaFuncionarios == []:
        layout.append([sg.Text("Sem funcionarios cadastrados!")])
    else:
        column = []
        for funcionario in listaFuncionarios:
            column.append([sg.Text("-----------------------------")])
            for key, value in funcionario.items():
                column.append([sg.Text("{}: {}".format(key, value))])
        layout.append([sg.Column(column, size=(500, 500), scrollable=True, vertical_scroll_only=True)])

    layout.append([sg.Button("Voltar")])

```

```
112     return sg.Window("Visualizar Funcionarios", layout=layout, finalize=
    True)
113 
114 def visualizar_tarefas_window():
115     sg.theme('GreenTan')
116     layout = [
117         [sg.Text("Tarefas: ")],
118     ]
119 
120     listaTarefas = admin_view.visualizarTarefas()
121 
122     if listaTarefas == []:
123         layout.append([sg.Text("Sem tarefas cadastradas!")])
124     else:
125         column = []
126         for tarefa in listaTarefas:
127             column.append([sg.Text("-----------------------------")])
128             for key, value in tarefa.items():
129                 if key == "funcionarios":
130                     column.append([sg.Text("Funcionarios:")])
131                     for funcionario in value:
132                         column.append([sg.Text(funcionario)])
133                 else:
134                     column.append([sg.Text("{}: {}".format(key, value))])
135         layout.append([sg.Column(column, size=(500, 500), scrollable=True,
     vertical_scroll_only=True)])
136 
137     layout.append([sg.Button("Voltar")])
138 
139     return sg.Window("Visualizar Tarefas", layout=layout, finalize=True)
140 
141 def visualizar_relatorios_window():
142     sg.theme('GreenTan')
143     layout = [
144         [sg.Text("Relatorios: ")],
145     ]
146 
147     listaRelatorios = funcionario_view.visualizarRelatorios()
148 
149     if listaRelatorios == []:
150         layout.append([sg.Text("Sem relatorios cadastrados!")])
151     else:
152         column = []
153         for relatorio in listaRelatorios:
154             column.append([sg.Text("-----------------------------")])
155             for key, value in relatorio.items():
156                 column.append([sg.Text("{}: {}".format(key, value))])
157         layout.append([sg.Column(column, size=(500, 500),scrollable=True,
    vertical_scroll_only=True)])
158 
159     layout.append([sg.Button("Voltar")])
160 
161     return sg.Window("Visualizar Relatorios", layout=layout, finalize=True
    )
162 
163 def visualizar_tarefas_funcionario_window():
164     sg.theme('GreenTan')
165     layout = [
166         [sg.Text("Tarefas: ")],
167     ]
```

```
168
169     listaTarefas = funcionario_view.visualizarTarefas()
170
171     if listaTarefas == []:
172         layout.append([sg.Text("Sem tarefas cadastrados!")])
173     else:
174         column = []
175         for tarefa in listaTarefas:
176             column.append([sg.Text("-----------------------------")])
177             for key, value in tarefa.items():
178                 column.append([sg.Text("{}: {}".format(key, value))])
179         layout.append([sg.Column(column, size=(500, 500),scrollable=True,
    vertical_scroll_only=True)])
180
181     layout.append([sg.Button("Voltar")])
182
183     return sg.Window("Visualizar Tarefas", layout=layout, finalize=True)
184
185 def adicionar_relatorio_window():
186     sg.theme('GreenTan')
187     layout = [[sg.Text("Selecione a tarefa do relatorio")]]
188
189     listaTarefas = funcionario_view.visualizarTarefas()
190     c = 0
191     column = []
192
193     if listaTarefas == []:
194         column.append([sg.Text("Sem tarefas cadastrados!")])
195     else:
196         for tarefa in listaTarefas:
197             column.append([sg.Radio(tarefa["nome"], group_id=1, key="{}".
    format(c))])
198             c += 1
199
200     layout.append([sg.Column(column, size=(400, 200), scrollable=True,
    vertical_scroll_only=True)])
201
202     layout.append([sg.Text("Texto do relatorio")])
203     layout.append([sg.Input(key="texto_relatorio")])
204
205     layout.append([sg.Text("Relatorio cadastrado com sucesso!", text_color
    ="white", background_color="green", visible=False, key="status_true")])
206
207     layout.append([sg.Text("Erro ao cadastrar relatorio! \nTente Novamente
    !", text_color="white", background_color="red", visible=False, key="
    status_false")])
208
209     layout.append([sg.Button("Cadastrar")])
210
211     layout.append([sg.Button("Voltar")])
212
213     return sg.Window("Adicionar relatorio", layout=layout, finalize=True),
     c, listaTarefas
214
215 def visualizar_relatorios_funcionario_window():
216     sg.theme('GreenTan')
217     layout = [
218         [sg.Text("Relatorios: ")],
219     ]
220
```

```
    listaRelatorios = funcionario_view.visualizarRelatorios()

    if listaRelatorios == []:
        layout.append([sg.Text("Sem relatorios cadastrados!")])
    else:
        column = []
        for relatorio in listaRelatorios:
            column.append([sg.Text("------------------------------")])
            for key, value in relatorio.items():
                column.append([sg.Text("{}: {}".format(key, value))])
        layout.append([sg.Column(column, size=(500, 500),scrollable=True, vertical_scroll_only=True)])

    layout.append([sg.Button("Voltar")])

    return sg.Window("Visualizar Relatorios", layout=layout, finalize=True)
```

Listing 5: windows.py

```
import requests
import json

url = "http://127.0.0.1:8000"

def adicionarFuncionario(usernameFunc,passwordFunc):
    with open("user.json", "r") as file:
        user_dados = json.load(file)
        token = user_dados["token"]

    request = requests.put(url+"/admin/funcionarios?token="+token, json={"username": usernameFunc, "password": passwordFunc})

    if request.status_code == 200:
       return True
    else:
       return False


def visualizarFuncionarios():

    with open("user.json", "r") as file:
        user_dados = json.load(file)
        token = user_dados["token"]

    request = requests.get(url+"/admin/funcionarios?token="+token)
    return request.json()

def criarTarefa(nome,descricao,data_prevista_conclusao,funcionariosId):

    with open("user.json", "r") as file:
        user_dados = json.load(file)
        token = user_dados["token"]

    tarefa = {
        "nome":nome,
        "descricao": descricao,
        "data_prevista_conclusao": data_prevista_conclusao,
        "funcionariosId": funcionariosId
    }
    request = requests.put(url+"/admin/tarefas?token="+token, json=tarefa)
```

```
41
42      if request.status_code == 200:
43          return True
44      else:
45          return False
46
47  def visualizarTarefas():
48      with open("user.json", "r") as file:
49          user_dados = json.load(file)
50          token = user_dados["token"]
51
52      request = requests.get(url+"/admin/tarefas?token="+token)
53
54      return request.json()
55  def visualizarRelatorios():
56      with open("user.json", "r") as file:
57          user_dados = json.load(file)
58          token = user_dados["token"]
59
60      request = requests.get(url+"/admin/relatorios?token="+token)
61
62      return request.json()
```

Listing 6: admin_view.py

```
1  import requests
2  import json
3  from datetime import date
4
5  url = "http://127.0.0.1:8000"
6
7  def visualizarTarefas():
8
9      with open("user.json", "r") as file:
10         user_dados = json.load(file)
11         token = user_dados["token"]
12
13     request = requests.get(f"{url}/funcionario/tarefas?token={token}")
14     return request.json()
15 def adicionarRelatorio(idTarefa, textoRelatorio):
16     with open("user.json", "r") as file:
17         user_dados = json.load(file)
18         idUser = user_dados["id"]
19         token = user_dados["token"]
20
21     request = requests.get(f"{url}/funcionario/tarefas?token={token}")
22
23     dataAtual = date.today().strftime("%d-%m-%Y")
24
25     relatorio = {
26         "id_user": idUser,
27         "id_tarefa": int(idTarefa),
28         "texto": textoRelatorio,
29         "data_criacao": dataAtual
30     }
31
32     request = requests.put(f"{url}/funcionario/relatorios?token={token}",
       json=relatorio)
33
34     if request.status_code == 200:
35         return True
```

```
36      else:
37          return False
38
39  def visualizarRelatorios():
40      with open("user.json", "r") as file:
41          user_dados = json.load(file)
42          token = user_dados["token"]
43
44      request = requests.get(url+"/funcionario/relatorios?token="+token)
45
46      return request.json()
```

Listing 7: funcionario_view.py

# 5 Conclusões Finais

Depois dos avancos obtidos e as metas estabelecidas no crédito anterior, criamos na parte do servidor à adicao de um token de validacao para a seguranca do sistema,e na parte do cliente a criacao de uma interface grafica, que permitiu a visualizacao do programa.

# Referências


[1] O que é fastapi. https://www.treinaweb.com.br/blog/o-que-e-fastapi. Accessed: 2024-06-02.

[2] Fastapi. https://fastapi.tiangolo.com/, 2024. Accessed: 2024-06-02.

[3] Real Python. Biblioteca de solicitações do python (guia) – python real. https://realpython.com/python-requests/, 2024. Accessed: 2024-06-02.

[4] Interfaces gráficas de usuários de maneira simples - o uso da biblioteca pysimplegui para criação de interfaces gráficas utilizando a linguagem python. https://bdta.ufra.edu.br/jspui/bitstream/123456789/2995/1/Interfaces%20gr%C3%A1ficas%20de%20usu%C3%A1rios%20de%20maneira%20simples.pdf, 2024. Accessed: 2024-06-15.

[5] Hugo Habbema. Pydantic. simplificando a validação de dados em python. https://medium.com/@habbema/pydantic-23fe91b5749b, 2024. Accessed: 2024-06-02.

[6] 9. classes — documentação python 3.12.3. https://docs.python.org/pt-br/3/tutorial/classes.html#classes, 2024. Accessed: 2024-06-02.

[7] Como criar apis em python usando fastapi — alura. https://www.alura.com.br/artigos/como-criar-apis-python-usando-fastapi, 2024. Accessed: 2024-06-01.

[8] First steps - fastapi. https://fastapi.tiangolo.com/tutorial/first-steps/, 2024. Accessed: 2024-06-01.

[9] Curso de banco de dados mysql - youtube. https://www.youtube.com/playlist?list=PLHz_AreHm4dkBs-795Dsgvau_ekxg8g1r, 2024. Accessed: 2024-06-01.

[10] Mysql. https://pythoniluminado.netlify.app/mysql#evitando-sql-injection, 2024. Accessed: 2024-06-01.

[11] Gerenciando banco de dados sqlite3 com python - parte 1 por regis da silva#pythonclub. https://pythonclub.com.br/gerenciando-banco-dados-sqlite3-python-parte1.html, 2024. Accessed: 2024-06-01.

[12] Projetos com python orientado a objetos — supygirls 2.0.0 documentation. https://supygirls.readthedocs.io/en/latest/intro_comp/PythonOO.html, 2024. Accessed: 2024-06-01.