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

Discentes: Guilherme Souza e Sara Costa

Gerenciador de tarefas para engenharia civil

Salvador
2024

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

1.1 Objetivo

Criar um Gerenciador de tarefas que forneça informações que circulam entre a administração de uma empresa e os funcionários. Fornecendo relatórios das tarefas.

1.2 Metodologia

Para o embasamento téorico e uma melhor compreensão 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 Motivação e justificativa

Pensando no futuro, onde se deseja um ambiente produtivo e organizado, teve-se como ideia, a criação de um programa em python que fosse capaz gerenciar tarefas básicas de uma obra da construção civil, buscando proporcionar uma maior organização nas tarefas que deverão 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, rápido e simples".

2.1.2 Uvicorn

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

2.1.3 Requests

Segundo [3], a biblioteca requests é o padrão para fazer solicitações HTTP em Python. Ele abstrai as complexidades de fazer solicitações por trás de uma API simples para que você possa se concentrar na interação com serviços e no consumo de dados em seu aplicativo.

2.1.4 Pydantic

Segundo [4], "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, análise de dados e muito mais. Uma das características mais marcantes do Pydantic é a sua facilidade de uso e a integração perfeita com outros componentes Python".

2.2 SQL

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

2.3 SQLite

Uma das principais vantagens do SQLite é que ele não 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 manipulação e distribuição extremamente fácil. Além disso, o SQLite é conhecido por sua alta confiabilidade e estabilidade, garantindo a integridade dos dados mesmo em situações de falha ou desligamento inesperado do sistema.

2.4 Orientação à objetos

Uma classe é representada por atributos e métodos. Os atributos de uma classe representam as características que esta classe possui, já 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 representação 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 [5], "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 Evolução 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 funcionários, que também conseguem visualizar essas atividades e criar relatórios as mesmas.

3.2 Funcionamento do programa

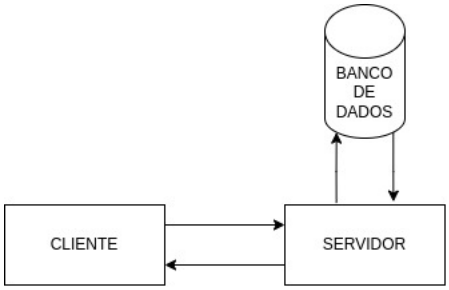

Figura 1: Funcionamento do programa

O programa funciona com base na ideia de client-server. O cliente faz requisições a um servidor, que por sua vez busca dados em banco de dados, e devolve uma resposta ao cliente. Esta abordagem garante uma gestão mais eficiente e escalável dos recursos e serviços, além de permiti a centralização das informações.

3.3 Banco de dados

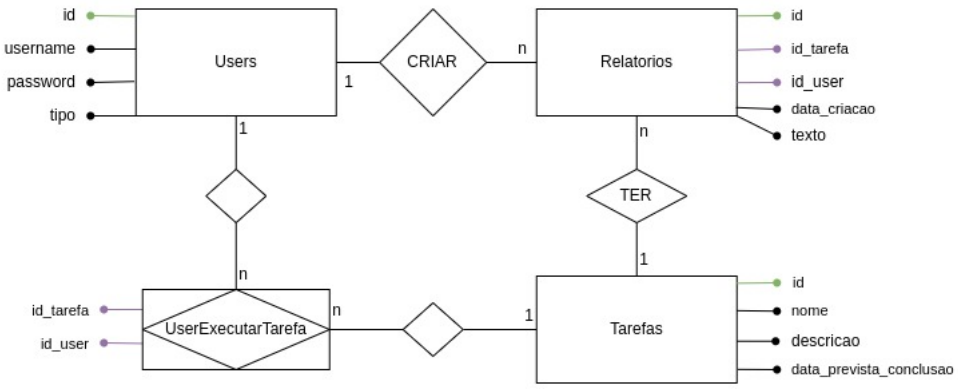

Figura 2: Diagrama entidade relacionamento

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 são criados por usuários e associados a tarefas específicas. Além disso, os usuários podem ser responsáveis pela execução de várias tarefas, conforme registrado na entidade UserExecutarTarefa.

3.4 Redes de Petri

3.4.1 Rede de Petri Ordinária

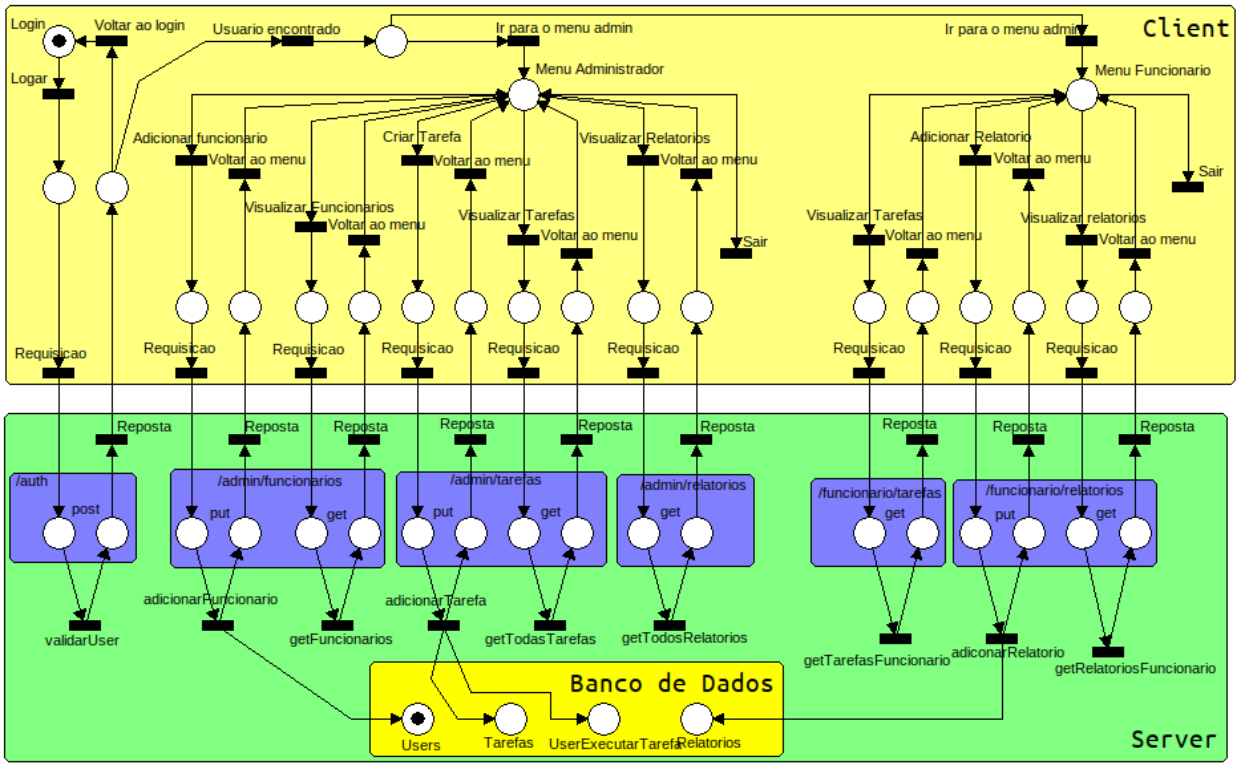

Figura 3: Rede de Petri Ordinária

3.4.2 Rede de Petri Colorida

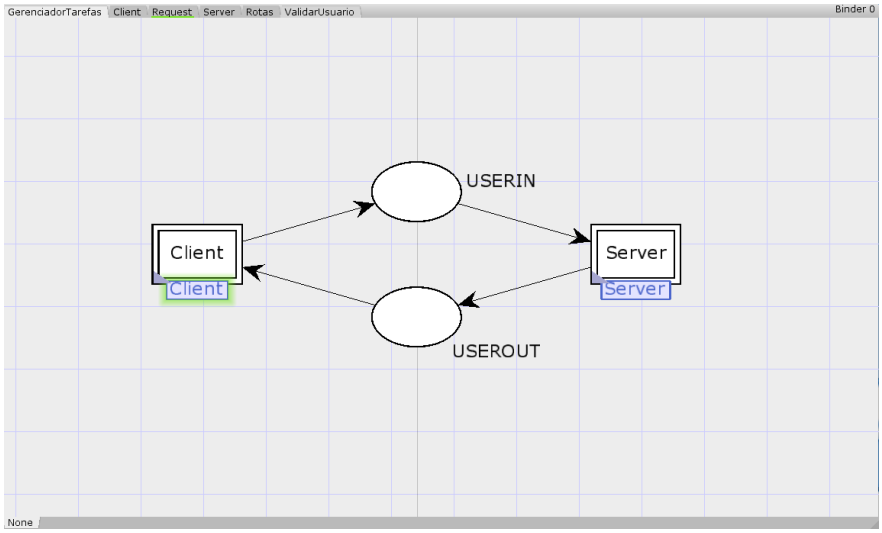

Figura 4: Visão geral do projeto

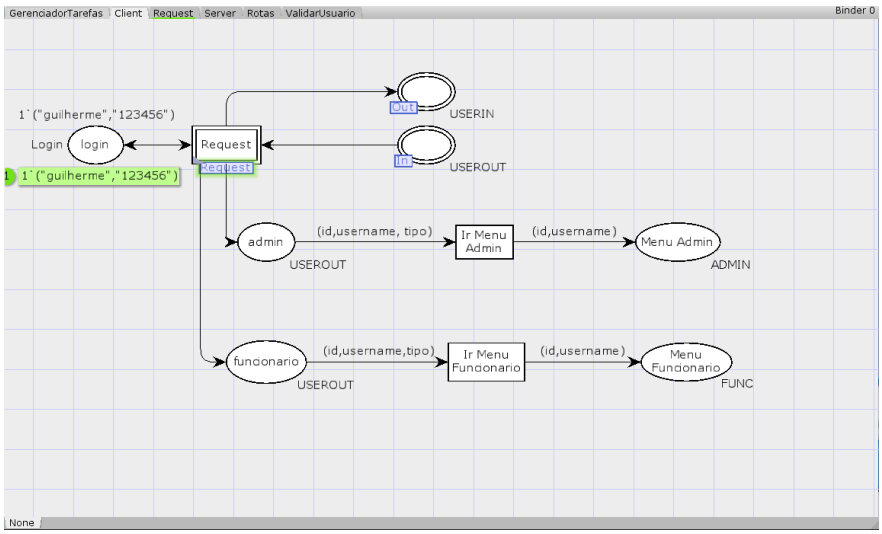

Figura 5: Cliente

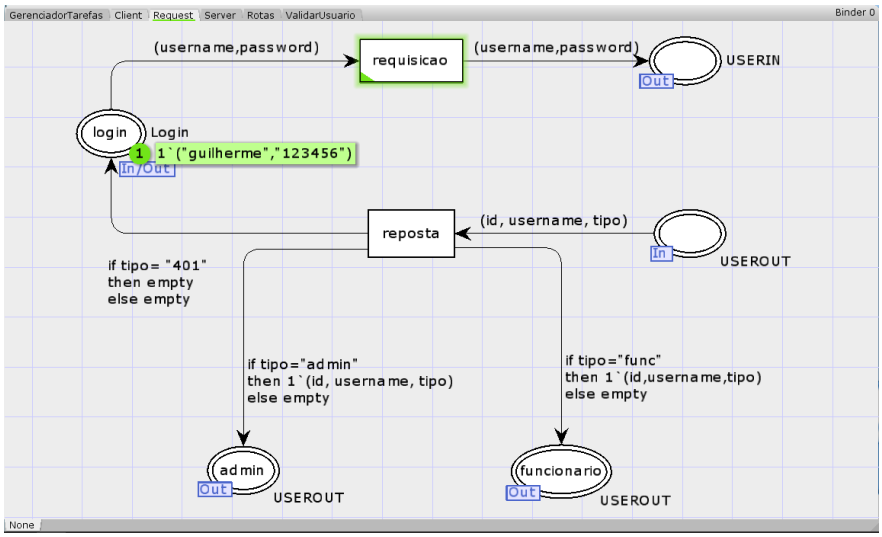

Figura 6: Requisição para o servidor e resposta

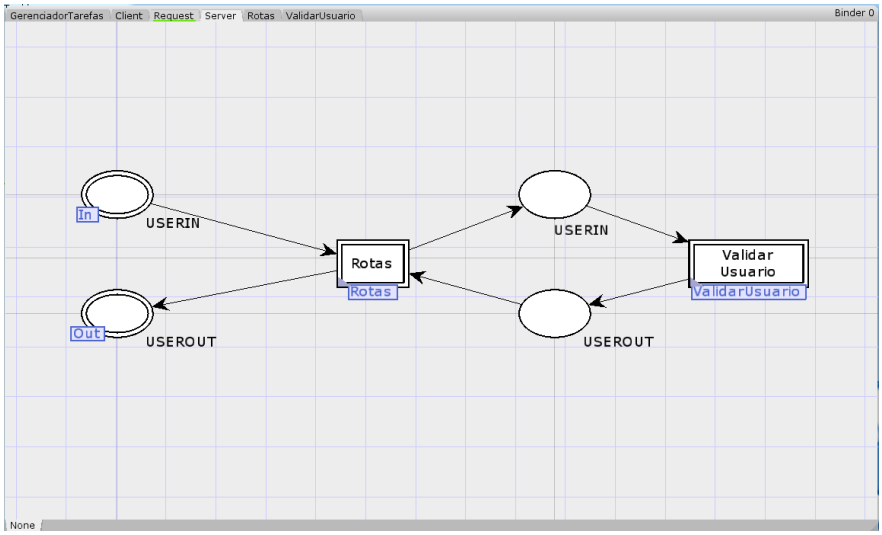

Figura 7: Servidor

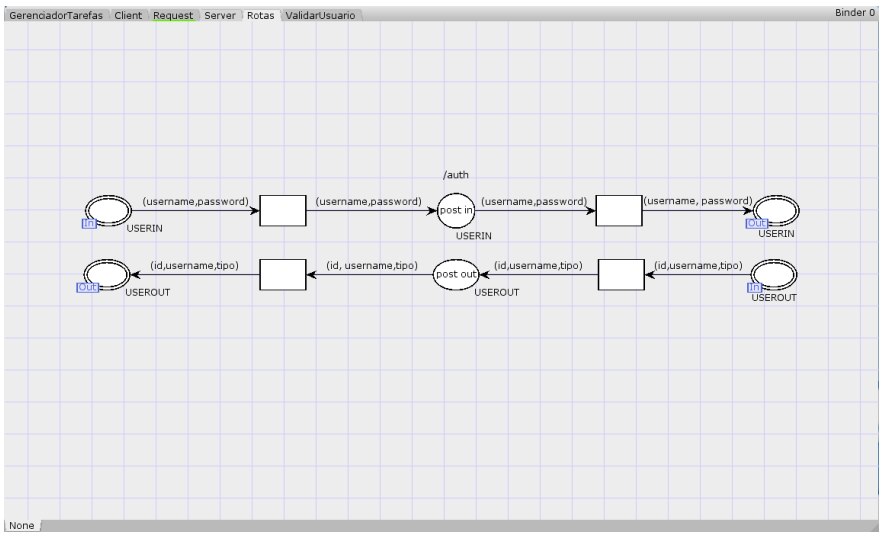

Figura 8: Rota de autenticação que recebe o nome de usuário e senha, e envia a resposta para o cliente

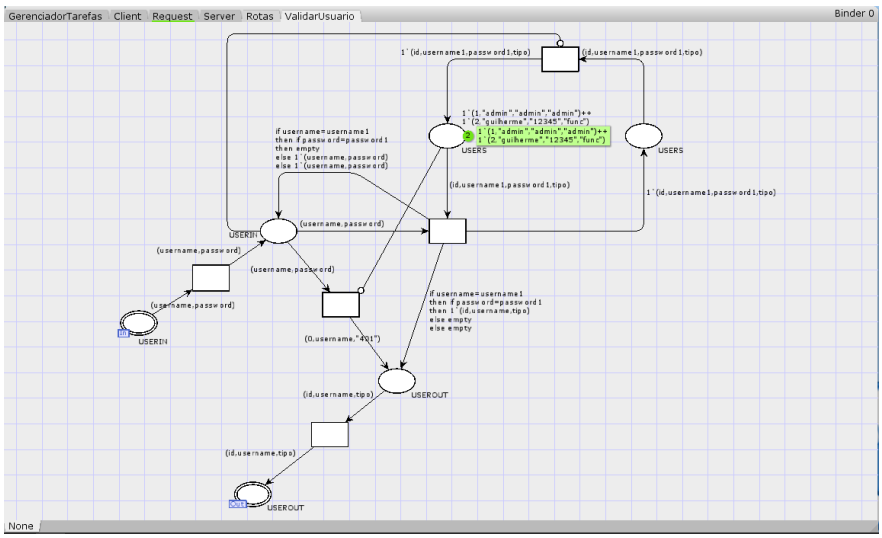

Figura 9: Recebe usuário e senha e valida se o usuário está no sistema

4 Código do programa

4.1 Servidor

```python
from fastapi import FastAPI, HTTPException, status
import uvicorn
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()

# inicializa o banco de dados de forma assincrona
@asynccontextmanager
```

```python
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"):
        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):
    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!"
        )

# Rota que lista todos os funcionario cadastrados no banco de dados
@app.get("/admin/funcionarios")
async def lista_funcionarios():
    return user_database.getFuncionarios()

# 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):
    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!"
        )

# Rota que lista todas as tarefas cadastradas no sistema
@app.get("/admin/tarefas")
async def get_todas_tarefas():
    return tarefa_database.getTodasTarefas()

# Rota que lista todos os relatorios cadastrados no sistema
@app.get("/admin/relatorios")
async def get_todos_relatorios():
    return relatorio_database.getTodosRelatorios()

# Rota que lista todas as tarefas do funcionario. Recebe o id do usuario
```

```python
@app.get("/funcionario/tarefas")
async def get_funcionario_tarefas(idUser: int):
    return tarefa_database.getTarefasFuncionario(idUser=idUser)

# 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):
    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!"
        )
# Rota que lista todos relatorios cadastrados pelo funcionario
@app.get("/funcionario/relatorios")
async def relatorios_cadastrados(idUser: int):
    return relatorio_database.getRelatoriosFuncionario(idUser=idUser)
# Inicializa o servidor se o arquivo for o principal
if __name__ == "__main__":
    uvicorn.run(app=app)
```

```python
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

# 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
```

```
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
        }

# 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

```

```
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

```
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

```
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

```
import requests
import os
import json
from view.admin_view import menuAdmin
from view.funcionario_view import menuFunc

url = "http://127.0.0.1:8000"

def login():
    while True:
        os.system("clear")

```

```
        print("-"*10, " Login ", "-"*10)

        username = input("Username: ")
        password = input("Password: ")

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

        if request.status_code == 200:
            break
        else:
            print("Usuario ou senha incorretos!")
            input("(PRESSIONE ENTER PARA TENTAR NOVAMENTE)")

    file = open("user.json", "w")
    json.dump(request.json(), file)
    file.close()

    print("Logado com sucesso!")
    input("(PRESSIONE ENTER PARA CONTINUAR)")

    if request.json()["tipo"] == "admin":
        menuAdmin()
    elif request.json()["tipo"] == "funcionario":
        menuFunc()

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

```
import requests
import os
import sys

url = "http://127.0.0.1:8000"

def menuAdmin():
    while True:
        os.system("clear")

        menuOptions = {
            "1": ["Adicionar Funcionario", adicionarFuncionario],
            "2": ["Visualizar Funcionarios", visualizarFuncionarios],
            "3": ["Criar Nova Tarefa", criarTarefa],
            "4": ["Visualizar Tarefas", visualizarTarefas],
            "5": ["Visualizar Relatorios", visualizarRelatorios],
            "6": ["Sair", sys.exit],
        }

        print("-"*10, 'ADMIN', "-"*10)
        for key, value in menuOptions.items():
            print(key, " - ", value[0])

        print("Escolha uma das opcoes:")

        option = input(">> ").strip()

        try:
            (menuOptions[option][1])()
        except Exception:
            pass

```

```python
def adicionarFuncionario():
    os.system("clear")

    print("-"*10, " Adicionar Funcionario ", "-"*10)

    print("Nome do funcionario: ")
    usernameFunc = input(">> ")
    print("Senha do funcionario: ")
    passwordFunc = input(">> ")

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

    if request.status_code == 200:
        print("Funcionario adicionado com sucesso!")
    else:
        print("Erro ao adicionar funcionario! Tente novamente.")

    input("(PRESSIONE ENTER PARA VOLTAR AO MENU)")

def visualizarFuncionarios():
    os.system("clear")

    print("-"*5, " Visualizar Funcionarios ", "-"*5)

    request = requests.get(url+"/admin/funcionarios")

    if request.json() == []:
        print("Sem funcionarios cadastrados!")
    else:
        for funcionario in request.json():
            print("-"*30)
            for key, value in funcionario.items():
                print(f"{key}: {value}")
            print("-"*30)

    input("(PRESSIONE ENTER PARA VOLTAR AO MENU)")

def criarTarefa():
    os.system("clear")

    print("-"*10, " Criar Tarefa ", "-"*10)
    tarefa = {
        "nome": "",
        "descricao": "",
        "data_prevista_conclusao": "",
        "funcionariosId": []
    }

    print("Digite o nome da tarefa: ")
    tarefa["nome"] = input(">> ")

    print("Digite a descricao da tarefa: ")
    tarefa["descricao"] = input(">> ")

    print("Digite a data prevista para conclusao(formato dd-mm-aaaa): ")
    tarefa["data_prevista_conclusao"] = input(">> ")

    request = requests.get(url+"/admin/funcionarios")

```

```
92      print("Funcionarios")
93
94      for funcionario in request.json():
95          print("   id: {}, nome: {}".format(funcionario.get("id"),
        funcionario.get("nome")))
96
97      print("Digite o id do funcionario (deixe vazio quando concluir): ")
98
99      while True:
100         funcionario = input(">> ")
101
102         if funcionario is "":
103             break
104         else:
105             try:
106                 tarefa.get("funcionariosId").append(int(funcionario))
107             except Exception:
108                 print("Insira um valor valido!")
109
110     if tarefa["funcionariosId"] == []:
111         print("A tarefa deve ter funcionarios!")
112     else:
113         request = requests.put(url+"/admin/tarefas", json=tarefa)
114
115         if request.status_code == 200:
116             print("Tarefa criada com sucesso!")
117         else:
118             print("Erro ao adicionar tarefa! Tente novamente.")
119
120     input("(PRESSIONE ENTER PARA VOLTAR AO MENU)")
121
122 def visualizarTarefas():
123     os.system("clear")
124
125     print("-"*10, " Visualizar Tarefas ", "-"*10)
126
127     request = requests.get(url+"/admin/tarefas")
128
129     if request.json() == []:
130         print("Sem tarefas cadastradas!")
131     else:
132         for tarefa in request.json():
133             print("-"*60)
134             for key, value in tarefa.items():
135                 if key == "funcionarios":
136                     print(f"{key}: ")
137                     for nomeFunc in value:
138                         print(f"   {nomeFunc}")
139                 else:
140                     print(f"{key}: {value}")
141             print("-"*60)
142
143     input("(PRESSIONE ENTER PARA VOLTAR AO MENU)")
144
145 def visualizarRelatorios():
146     os.system("clear")
147
148     print("-"*10, " Visualizar Relatorios ", "-"*10)
149     request = requests.get(url+"/admin/relatorios")
150     if request.json() == []:
```

```
        print("Sem relatorios cadastradas!")
    else:
        for relatorio in request.json():
            print("-"*60)
            for key, value in relatorio.items():
                print(f"{key}: {value}")
            print("-"*60)
    input("(PRESSIONE ENTER PARA VOLTAR AO MENU
```

```
import requests
import os
import sys
import json
from datetime import date

url = "http://127.0.0.1:8000"

def menuFunc():
     while True:
        os.system("clear")

        menuOptions = {
            "1": ["Visualizar Tarefas", visualizarTarefas],
            "2": ["Adicionar Relatorio", adicionarRelatorio],
            "3": ["Visualizar Relatorios", visualizarRelatorios],
            "4": ["Sair", sys.exit],
        }

        print("-"*10, 'FUNCIONARIO', "-"*10)
        for key, value in menuOptions.items():
            print(key, " - ", value[0])

        print("Escolha uma das opcoes:")

        option = input(">> ").strip()

        try:
            (menuOptions[option][1])()
        except Exception:
            pass

def visualizarTarefas():
    os.system("clear")

    print("-"*10, " Visualizar Tarefas ", "-"*10)

    file = open("user.json", "r")
    idUser = (json.load(file))["id"]
    file.close()

    request = requests.get(f"{url}/funcionario/tarefas?idUser={idUser}")

    if request.json() == []:
        print("Sem tarefas cadastradas!")
    else:
        for tarefa in request.json():
            print("-"*60)
            for key, value in tarefa.items():
                print(f"{key}: {value}")
            print("-"*60)

```

```
    input("(PRESSIONE ENTER PARA VOLTAR AO MENU)")

def adicionarRelatorio():
    os.system("clear")

    print("-"*10, "Adicionar Relatorios", "-"*10)

    file = open("user.json", "r")
    idUser = (json.load(file))["id"]
    file.close()

    request = requests.get(f"{url}/funcionario/tarefas?idUser={idUser}")

    print("Tarefas: ")

    for tarefa in request.json():
        print("   id: {}, nome: {}".format(tarefa.get("id"), tarefa.get("
    nome")))

    print("Digite o id da tarefa que voce deseja fazer o relatorio: ")
    idTarefa = int(input(">> "))

    print("Digite o texto do relatorio: ")
    textoRelatorio = input(">> ")

    dataAtual = date.today()
    dataAtual = f'{dataAtual.day}-{dataAtual.month}-{dataAtual.year}'

    relatorio = {
        "id_user": idUser,
        "id_tarefa": idTarefa,
        "texto": textoRelatorio,
        "data_criacao": dataAtual
    }

    request = requests.put(f"{url}/funcionario/relatorios", json=relatorio
    )

    if request.status_code == 200:
        print("Relatorio adicionado com sucesso!")
    else:
        print("Erro ao adicionar relatorio! Tente novamente.")

    input("(PRESSIONE ENTER PARA VOLTAR MENU)")
def visualizarRelatorios():
    os.system("clear")
    print("-"*10, "Visualizar Relatorios", "-"*10)
    file = open("user.json", "r")
    idUser = (json.load(file))["id"]
    file.close()

    request = requests.get(f"{url}/funcionario/relatorios?idUser={idUser}"
    )

    if request.json() == []:
        print("Sem relatorios cadastradas!")
    else:
        for relatorio in request.json():
            print("-"*60)
            for key, value in relatorio.items():
```

```
110                     print(f"{key}: {value}")
111                 print("-"*60)
112 
113     input("(PRESSIONE ENTER PARA VOLTAR AO MENU)")
```

5 Conclusões parciais

Depois dos avanços obtidos até aqui, tem-se como perspectiva futura na parte do servidor à adição de um token de validação para a segurança do sistema, e na parte do cliente a criação de uma interface gráfica.

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

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

[6] 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.

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

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

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

[10] 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.

[11] 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.