



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

<b>Lista de Figuras e Tabelas</b>	<b>1</b>
<b>1 Introducao</b>	<b>2</b>
1.1 Objetivo . . . . .	2
1.2 Metodologia . . . . .	2
1.3 Motivacao e justificativa . . . . .	2
<b>2 Referencial Teórico</b>	<b>2</b>
2.1 Bibliotecas . . . . .	2
2.1.1 FastAPI . . . . .	2
2.1.2 Uvicorn . . . . .	2
2.1.3 Requests . . . . .	2
2.1.4 PySimpleGUI . . . . .	2
2.1.5 Pydantic . . . . .	3
2.2 SQL . . . . .	3
2.3 SQLite . . . . .	3
2.4 Orientacao a objetos . . . . .	3
<b>3 Proposta do Programa</b>	<b>3</b>
3.1 Evolucao do programa . . . . .	3
3.2 Funcionamento do programa . . . . .	4
3.3 Banco de dados . . . . .	4
3.4 Redes de Petri . . . . .	5
3.4.1 Rede de Petri Ordinaria . . . . .	5
3.4.2 Rede de Petri Colorida . . . . .	5
<b>4 Código do programa</b>	<b>7</b>
4.1 Servidor . . . . .	7
4.2 Cliente . . . . .	17
<b>5 Conclusões Finais</b>	<b>27</b>
<b>Referências</b>	<b>28</b>

## Lista de Figuras

1	Funcionamento do programa . . . . .	4
2	Diagrama entidade relacionamento . . . . .	4
3	Rede de Petri Ordinaria . . . . .	5
4	Visao geral do projeto . . . . .	5
5	Cliente . . . . .	6
6	Requisicao para o servidor e resposta . . . . .	6
7	Servidor . . . . .	6
8	Rota de autenticacao que recebe o nome de usuario e senha, e envia a resposta para o cliente . . . . .	7
9	Recebe usuario e senha e valida se o usuario esta no sistema . . . . .	7

## Lista de Tabelas

# 1 Introducao

## 1.1 Objetivo

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

## 1.2 Metodologia

Para o embasamento teórico 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 Motivacao 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 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 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 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, 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 servidor 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 a 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 [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 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

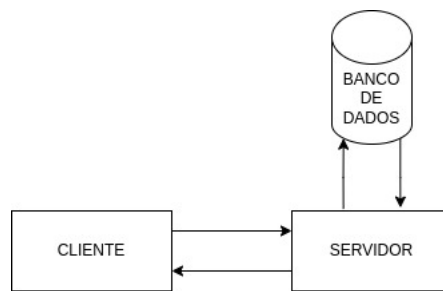


Figura 1: Funcionamento do programa

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 permitir a centralização das informações.

## 3.3 Banco de dados

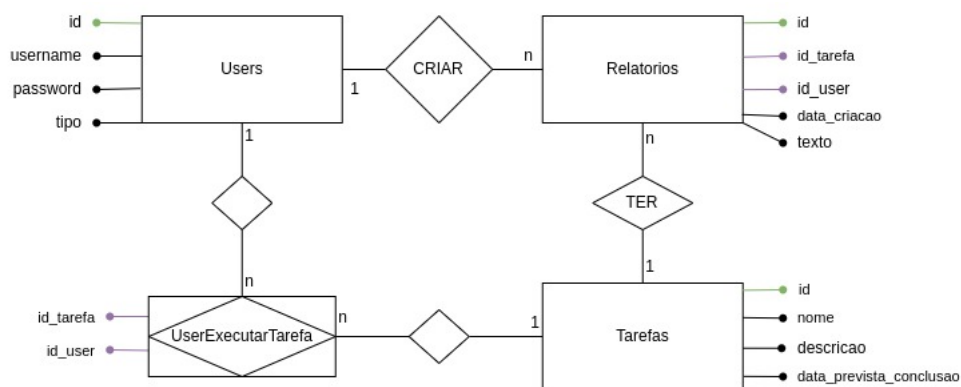


Figura 2: Diagrama entidade relacionamento

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 Ordinaria

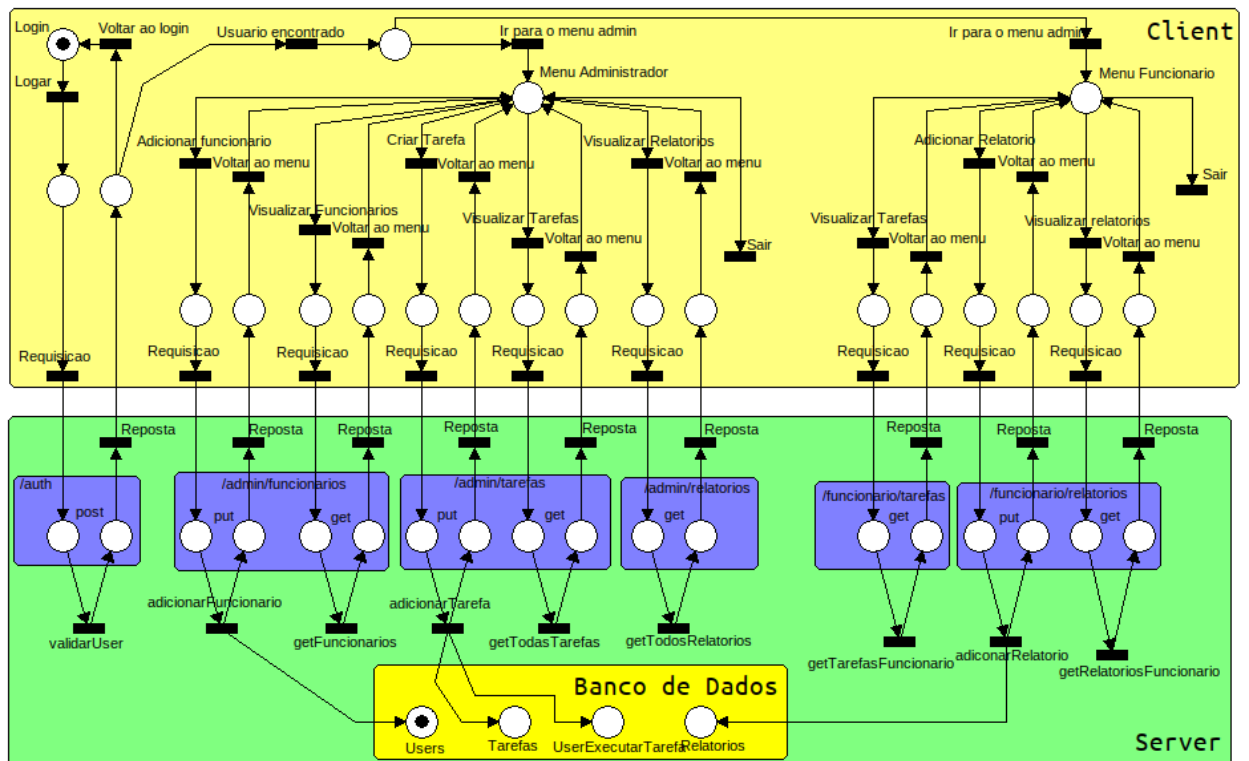


Figura 3: Rede de Petri Ordinaria

### 3.4.2 Rede de Petri Colorida

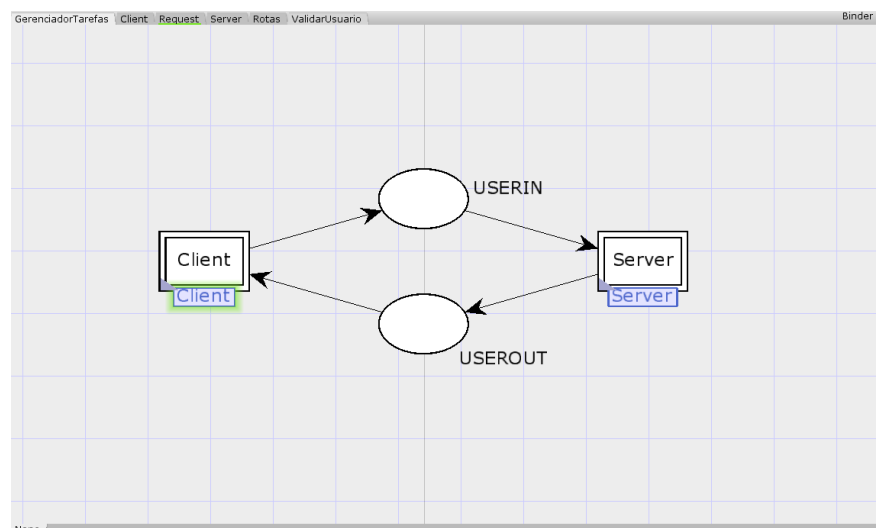


Figura 4: Visao geral do projeto

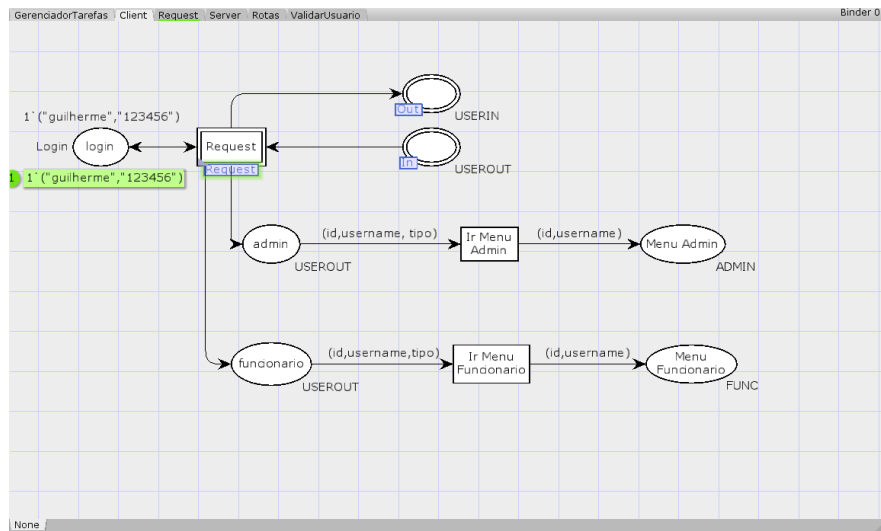


Figura 5: Cliente

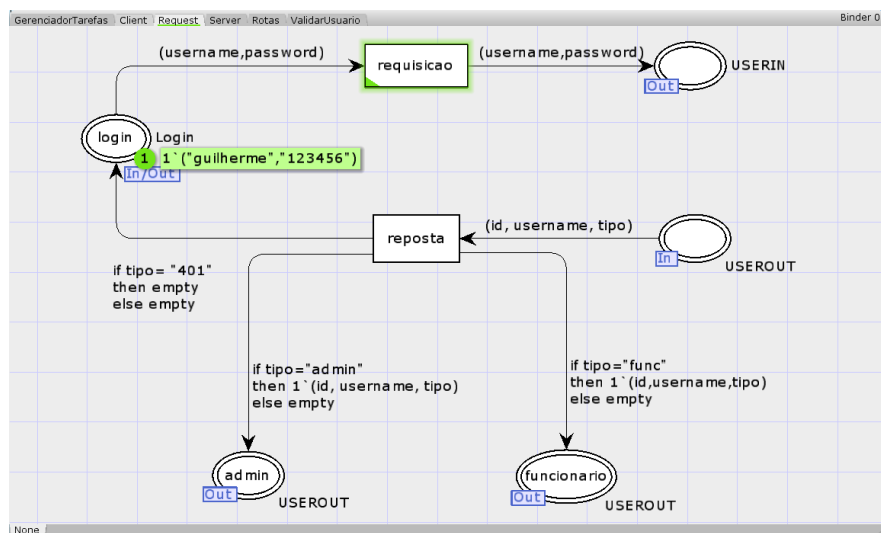


Figura 6: Requisicao para o servidor e resposta

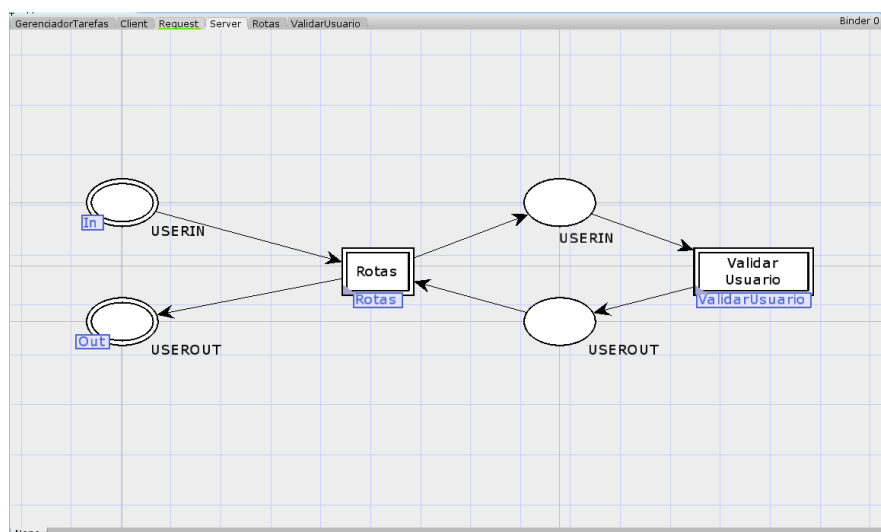


Figura 7: Servidor



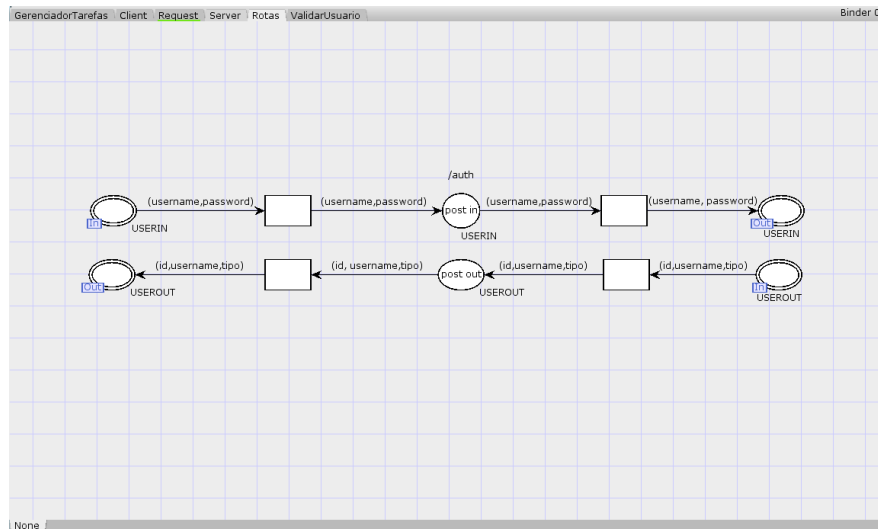


Figura 8: Rota de autenticação que recebe o nome de usuário e senha, e envia a resposta para o cliente

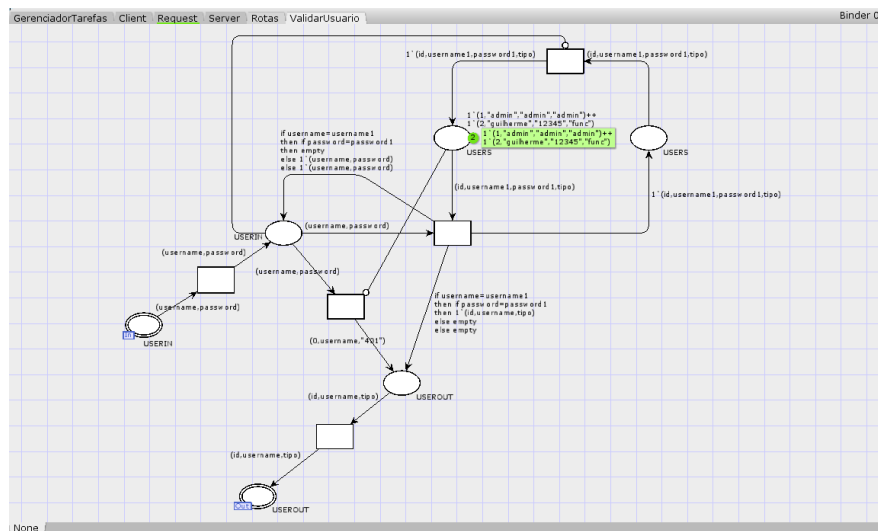


Figura 9: Recebe usuário e senha e valida se o usuário está no sistema

## 4 Código do programa

### 4.1 Servidor

```

1 from fastapi import FastAPI, HTTPException, status
2 import uvicorn
3 import jwt
4 from datetime import datetime, timedelta, timezone
5 from contextlib import asynccontextmanager
6 from src import database, user_database, tarefa_database,
   relatorio_database
7 from src.models.user_model import UserIn
8 from src.models.tarefa_model import Tarefa
9 from src.models.relatorio_model import Relatorio
10
11 app = FastAPI()
12

```

```

13 token_secret = "12345"
14
15 # inicializa o banco de dados de forma assincrona
16 @asynccontextmanager
17 async def lifespan(app: FastAPI):
18     await database.inicializarBancoDados()
19     yield
20
21 # Executa a funcao lifespan junto com a inicializacao do servidor
22 app.router.lifespan_context = lifespan
23
24 # Rota de autenticao. Recebe usuario e senha do client e valida se usuario
    esta cadastrado no banco de dados
25 @app.post("/auth")
26 async def autenticar_user(user: UserIn):
27     validarUser = user_database.validarUser(user=user)
28
29     if validarUser.get("status"):
30         token = jwt.encode({
31             'exp': datetime.now(timezone.utc) + timedelta(minutes=30),
32             "nivel-acesso": validarUser.get("user").tipo,
33             "id": validarUser.get("user").id
34         }, key=token_secret, algorithm='HS256')
35
36         validarUser.get("user").token = token
37
38         return validarUser.get("user")
39     else:
40         raise HTTPException(
41             status_code=status.HTTP_401_UNAUTHORIZED,
42             detail="Usuario nao encontrado!"
43         )
44
45 # Rota para adicionar funcionario, recebe nome de usuario e senha
46 @app.put("/admin/funcionarios")
47 async def adicionar_funcionario(func: UserIn, token: str):
48     try:
49         token_deco = jwt.decode(token, key=token_secret, algorithms="HS256")
50
51         if token_deco.get("nivel-acesso") == 'admin':
52             if user_database.adicionarFuncionario(func=func):
53                 return "Funcionario Adicionado com Sucesso!"
54             else:
55                 raise HTTPException(
56                     status_code=status.HTTP_501_NOT_IMPLEMENTED,
57                     detail="Erro ao adicionar novo funcionario!"
58                 )
59         else:
60             raise PermissionError
61     except jwt.DecodeError:
62         return HTTPException(
63             status_code=status.HTTP_401_UNAUTHORIZED,
64             detail="Token invalido!"
65         )
66     except jwt.ExpiredSignatureError:
67         return HTTPException(
68             status_code=status.HTTP_401_UNAUTHORIZED,
69             detail="Token Expirado!"
70         )

```

```

71     except PermissionError:
72         return HTTPException(
73             status_code=status.HTTP_401_UNAUTHORIZED,
74             detail="Permissao Negada!"
75         )
76
77 # Rota que lista todos os funcionario cadastrados no banco de dados
78 @app.get("/admin/funcionarios")
79 async def lista_funcionarios(token: str):
80     try:
81         token_deco = jwt.decode(token, key=token_secret, algorithms="HS256")
82
83         if token_deco.get("nivel-acesso") == 'admin':
84             return user_database.getFuncionarios()
85         else:
86             raise PermissionError
87     except jwt.DecodeError:
88         return HTTPException(
89             status_code=status.HTTP_401_UNAUTHORIZED,
90             detail="Token invalido!"
91         )
92     except jwt.ExpiredSignatureError:
93         return HTTPException(
94             status_code=status.HTTP_401_UNAUTHORIZED,
95             detail="Token Expirado!"
96         )
97     except PermissionError:
98         return HTTPException(
99             status_code=status.HTTP_401_UNAUTHORIZED,
100             detail="Permissao Negada!"
101         )
102
103 # Rota que adiciona uma nova tarefa do banco de dados. Recebe nome,
104 # descricao, data prevista de conclusao e id dos funcionarios
105 @app.put("/admin/tarefas")
106 async def adicionar_tarefa(tarefa: Tarefa, token: str):
107     try:
108         token_deco = jwt.decode(token, key=token_secret, algorithms="HS256")
109
110         if token_deco.get("nivel-acesso") == 'admin':
111             if tarefa_database.adicionarTarefa(tarefa=tarefa):
112                 return "Tarefa adicionada com Sucesso!"
113             else:
114                 raise HTTPException(
115                     status_code=status.HTTP_501_NOT_IMPLEMENTED,
116                     detail="Erro ao adicionar tarefa!"
117                 )
118         else:
119             raise PermissionError
120     except jwt.DecodeError:
121         return HTTPException(
122             status_code=status.HTTP_401_UNAUTHORIZED,
123             detail="Token invalido!"
124         )
125     except jwt.ExpiredSignatureError:
126         return HTTPException(
127             status_code=status.HTTP_401_UNAUTHORIZED,
128             detail="Token Expirado!"

```

```

128     )
129     except PermissionError:
130         return HTTPException(
131             status_code=status.HTTP_401_UNAUTHORIZED,
132             detail="Permissao Negada!"
133         )
134
135 # Rota que lista todas as tarefas cadastradas no sistema
136 @app.get("/admin/tarefas")
137 async def get_todas_tarefas(token: str):
138     try:
139         token_deco = jwt.decode(token, key=token_secret, algorithms="HS256")
140
141         if token_deco.get("nivel-acesso") == 'admin':
142             return tarefa_database.getTodasTarefas()
143         else:
144             raise PermissionError
145     except jwt.DecodeError:
146         return HTTPException(
147             status_code=status.HTTP_401_UNAUTHORIZED,
148             detail="Token invalido!"
149         )
150     except jwt.ExpiredSignatureError:
151         return HTTPException(
152             status_code=status.HTTP_401_UNAUTHORIZED,
153             detail="Token Expirado!"
154         )
155     except PermissionError:
156         return HTTPException(
157             status_code=status.HTTP_401_UNAUTHORIZED,
158             detail="Permissao Negada!"
159         )
160
161 # Rota que lista todos os relatorios cadastrados no sistema
162 @app.get("/admin/relatorios")
163 async def get_todos_relatorios(token: str):
164     try:
165         token_deco = jwt.decode(token, key=token_secret, algorithms="HS256")
166
167         if token_deco.get("nivel-acesso") == 'admin':
168             return relatorio_database.getTodosRelatorios()
169         else:
170             raise PermissionError
171     except jwt.DecodeError:
172         return HTTPException(
173             status_code=status.HTTP_401_UNAUTHORIZED,
174             detail="Token invalido!"
175         )
176     except jwt.ExpiredSignatureError:
177         return HTTPException(
178             status_code=status.HTTP_401_UNAUTHORIZED,
179             detail="Token Expirado!"
180         )
181     except PermissionError:
182         return HTTPException(
183             status_code=status.HTTP_401_UNAUTHORIZED,
184             detail="Permissao Negada!"
185         )

```

```

186
187 # Rota que lista todas as tarefas do funcionario. Recebe o id do usuario
188 @app.get("/funcionario/tarefas")
189 async def get_funcionario_tarefas(token: str):
190     try:
191         token_deco = jwt.decode(token, key=token_secret, algorithms="HS256")
192
193         if token_deco.get("nivel-acesso") == 'funcionario':
194             return tarefa_database.getTarefasFuncionario(idUser=token_deco.get("id"))
195         else:
196             raise PermissionError
197     except jwt.DecodeError:
198         return HTTPException(
199             status_code=status.HTTP_401_UNAUTHORIZED,
200             detail="Token invalido!"
201         )
202     except jwt.ExpiredSignatureError:
203         return HTTPException(
204             status_code=status.HTTP_401_UNAUTHORIZED,
205             detail="Token Expirado!"
206         )
207     except PermissionError:
208         return HTTPException(
209             status_code=status.HTTP_401_UNAUTHORIZED,
210             detail="Permissao Negada!"
211         )
212
213 # Rota que adiciona um novo relatorio. Recebe id do usuario, id da tarefa,
214 # texto, data de criacao
215 @app.put("/funcionario/relatorios")
216 async def adicionar_relatorio(relatorio: Relatorio, token: str):
217     try:
218         token_deco = jwt.decode(token, key=token_secret, algorithms="HS256")
219
220         if token_deco.get("nivel-acesso") == 'funcionario':
221             if relatorio_database.adicionarRelatorio(relatorio=relatorio):
222                 return "Relatorio adicionado com sucesso!"
223             else:
224                 raise HTTPException(
225                     status_code=status.HTTP_501_NOT_IMPLEMENTED,
226                     detail="Erro ao adicionar tarefa!"
227                 )
228         else:
229             raise PermissionError
230     except jwt.DecodeError:
231         return HTTPException(
232             status_code=status.HTTP_401_UNAUTHORIZED,
233             detail="Token invalido!"
234         )
235     except jwt.ExpiredSignatureError:
236         return HTTPException(
237             status_code=status.HTTP_401_UNAUTHORIZED,
238             detail="Token Expirado!"
239         )
240     except PermissionError:
241         return HTTPException(
242             status_code=status.HTTP_401_UNAUTHORIZED,

```

```

242         detail="Permissao Negada!")
243     )
244
245 # Rota que lista todos relatorios cadastrados pelo funcionario
246 @app.get("/funcionario/relatorios")
247 async def relatorios_cadastrados(token: str):
248     try:
249         token_deco = jwt.decode(token, key=token_secret, algorithms="HS256")
250
251         if token_deco.get("nivel-acesso") == 'funcionario':
252             return relatorio_database.getRelatoriosFuncionario(idUser=
token_deco.get("id"))
253         else:
254             raise PermissionError
255     except jwt.DecodeError:
256         return HTTPException(
257             status_code=status.HTTP_401_UNAUTHORIZED,
258             detail="Token invalido!"
259         )
260     except jwt.ExpiredSignatureError:
261         return HTTPException(
262             status_code=status.HTTP_401_UNAUTHORIZED,
263             detail="Token Expirado!"
264         )
265     except PermissionError:
266         return HTTPException(
267             status_code=status.HTTP_401_UNAUTHORIZED,
268             detail="Permissao Negada!"
269         )
270
271 # Inicializa o servidor se o arquivo for o principal
272 if __name__ == "__main__":
273     uvicorn.run(app=app)

```

```

1 import sqlite3
2
3 # Cria a conexao com banco de dados
4 def conexaoBancoDados() -> sqlite3.Connection:
5
6     caminhoBd = "./server/database/gerenciador-tarefas.db"
7
8     conn = sqlite3.connect(caminhoBd)
9
10    return conn
11
12 # Inicializa banco de dados
13 async def inicializarBancoDados() -> None:
14     conn = conexaoBancoDados()
15     cursor = conn.cursor()
16
17     cursor.execute('''CREATE TABLE IF NOT EXISTS Users(
18         id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
19         username TEXT NOT NULL UNIQUE,
20         password TEXT,
21         tipo TEXT CHECK(tipo IN ('admin', 'funcionario')) NOT
22         NULL DEFAULT 'funcionario'
23     );''')
24
25     cursor.execute('''CREATE TABLE IF NOT EXISTS Tarefas(
26         id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,

```

```

26         nome TEXT NOT NULL UNIQUE,
27         descricao TEXT NOT NULL,
28         data_prevista_conclusao TEXT
29     );'''
30
31     cursor.execute('''CREATE TABLE IF NOT EXISTS Relatorios(
32         id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
33         id_user INTEGER NOT NULL,
34         id_tarefa INTEGER NOT NULL,
35         data_criacao TEXT NOT NULL,
36         texto TEXT,
37         FOREIGN KEY(id_user) REFERENCES Users(id),
38         FOREIGN KEY(id_tarefa) REFERENCES Tarefas(id)
39     );'''
40
41     cursor.execute('''CREATE TABLE IF NOT EXISTS UserExecutarTarefa(
42         id_user INTEGER NOT NULL,
43         id_tarefa INTEGER NOT NULL,
44         FOREIGN KEY(id_user) REFERENCES Users(id),
45         FOREIGN KEY(id_tarefa) REFERENCES Tarefas(id)
46     );'''
47
48     cursor.execute("SELECT * FROM Users WHERE username = 'admin';")
49     query = cursor.fetchone()
50
51     if query == None:
52         cursor.execute('''INSERT INTO Users (username, password, tipo)
53 VALUES ('admin', 'admin', 'admin');''')
54
55     conn.commit()
56     print("Conexao com banco de dados bem sucedida!")
57     conn.close()

```

```

1 from src.models.tarefa_model import Tarefa
2 from src.database import conexaoBancoDados
3
4 # Adiciona uma nova tarefa no banco de dados. Recebe como argumento um
5 # objeto Tarefa
6 def adicionarTarefa(tarefa: Tarefa) -> bool:
7     try:
8         conn = conexaoBancoDados()
9         cursor = conn.cursor()
10
11         cursor.execute("""INSERT INTO Tarefas
12             (nome, descricao, data_prevista_conclusao) VALUES
13             (?, ?, ?);
14             """, (tarefa.nome, tarefa.descricao, tarefa.
15                 data_prevista_conclusao))
16
17         cursor.execute("""SELECT id FROM Tarefas
18             WHERE nome=?;""", (tarefa.nome,))
19
20         id_tarefa = cursor.fetchone()[0]
21
22         for func in tarefa.funcionariosId:
23             cursor.execute("""INSERT INTO UserExecutarTarefa
24                 (id_user, id_tarefa) VALUES
25                 (?, ?);""", (func, id_tarefa))
26
27         conn.commit()

```

```

26         conn.close()
27         return True
28     except Exception as err:
29         print(err)
30         return False
31 # Retorna uma lista de dicionarios com todas as tarefas cadastradas no
    banco de dados
32 def getTodasTarefas() -> list:
33     conn = conexaoBancoDados()
34     cursor = conn.cursor()
35     tarefasId = {}
36     tarefas = []
37     cursor.execute("""SELECT * FROM UserExecutarTarefa;""")
38     relacaoUserTarefa = cursor.fetchall()
39
40     for rel in relacaoUserTarefa:
41         if rel[1] in tarefasId:
42             tarefasId[rel[1]].append(rel[0])
43         else:
44             tarefasId[rel[1]] = [rel[0]]
45     c = 0
46     for key, value in tarefasId.items():
47         cursor.execute("""SELECT nome, descricao, data_prevista_conclusao
    FROM Tarefas
48                             WHERE id = ?;""", (key,))
49         tarefa = cursor.fetchone()
50
51         tarefas.append({
52             "id": key,
53             "nome": tarefa[0],
54             "descricao": tarefa[1],
55             "data_prevista_conclusao": tarefa[2],
56             "funcionarios": []
57         })
58
59     for funcId in value:
60         cursor.execute("""SELECT username FROM Users
61                             WHERE id = ?;""", (funcId,))
62         funcName = cursor.fetchone()[0]
63         tarefas[c]["funcionarios"].append(funcName)
64     c += 1
65     conn.close()
66     return tarefas
67 # Retorna uma lista de dicionario com todas as tarefas do funcionario.
    Recebe o id do funcionario como argumento
68 def getTarefasFuncionario(idUser: int) -> list:
69     conn = conexaoBancoDados()
70     cursor = conn.cursor()
71     cursor.execute("""SELECT id_tarefa FROM UserExecutarTarefa
72                             WHERE id_user = ?;""", (idUser,))
73     idTarefas = cursor.fetchall()
74     tarefas = []
75     c = 0
76     while c < len(idTarefas):
77         cursor.execute("""SELECT * FROM Tarefas
78                             WHERE id = ?;""", (idTarefas[c][0],))
79         tarefa = cursor.fetchall()
80         tarefas.append({
81             "id": tarefa[0][0],
82             "nome": tarefa[0][1],

```



```

83         "descricao": tarefa[0][2],
84         "data_prevista_conclusao": tarefa[0][3]
85     })
86     c += 1
87     conn.close()
88     return tarefas

1 from src.database import conexaoBancoDados
2 from src.models.relatorio_model import Relatorio
3
4 # Adiciona um novo relatorio no banco de dados. Recebe um objeto Relatorio
  como argumento
5 def adicionarRelatorio(relatorio: Relatorio) -> bool:
6     try:
7         conn = conexaoBancoDados()
8         cursor = conn.cursor()
9
10        cursor.execute("""INSERT INTO Relatorios
11                        (id_user, id_tarefa, data_criacao, texto) VALUES
12                        (?, ?, ?, ?);
13                        """, (relatorio.id_user, relatorio.id_tarefa,
relatorio.data_criacao, relatorio.texto))
14
15        conn.commit()
16        conn.close()
17
18        return True
19    except Exception as err:
20        print(err)
21        return False
22
23 # Retorna uma lista de dicionarios com os relatorios do funcionario.
  Recebe o id do funcionario como argumento
24 def getRelatoriosFuncionario(idUser: int) -> list:
25     conn = conexaoBancoDados()
26     cursor = conn.cursor()
27
28     cursor.execute("""SELECT id, id_tarefa, data_criacao, texto FROM
Relatorios
29                     WHERE id_user = ?;""", (idUser,))
30
31     relatorios = []
32     queryRelatorios = cursor.fetchall()
33
34     for relatorio in queryRelatorios:
35         cursor.execute("""SELECT nome FROM Tarefas
36                         WHERE id = ?;""", (relatorio[1],))
37         nomeTarefa = cursor.fetchone()[0]
38
39         relatorios.append({
40             "id": relatorio[0],
41             "id_tarefa": relatorio[1],
42             "nome_tarefa": nomeTarefa,
43             "data_criacao": relatorio[2],
44             "texto": relatorio[3]
45         })
46
47     conn.close()
48
49     return relatorios
50

```

```

51 # Retorna uma lista de dicionarios com todos os relatorios cadastrados no
    sistema
52 def getTodosRelatorios() -> list:
53     conn = conexaoBancoDados()
54     cursor = conn.cursor()
55
56     cursor.execute("""SELECT id, id_user, id_tarefa, data_criacao, texto
    FROM Relatorios;""")
57     query = cursor.fetchall()
58
59     relatorios = []
60
61     for relatorio in query:
62         cursor.execute("""SELECT username FROM Users
63                             WHERE id = ?;""", (relatorio[1],))
64         nomeFunc = cursor.fetchone()[0]
65
66         cursor.execute("""SELECT nome FROM Tarefas
67                             WHERE id = ?;""", (relatorio[2],))
68         nomeTarefa = cursor.fetchone()[0]
69
70         relatorios.append({
71             "id": relatorio[0],
72             "id_user": relatorio[1],
73             "id_tarefa": relatorio[2],
74             "nome_funcionario": nomeFunc,
75             "nome_tarefa": nomeTarefa,
76             "data_criacao": relatorio[3],
77             "texto": relatorio[4]
78         })
79
80     conn.close()
81
82     return relatorios

```

```

1 from src.database import conexaoBancoDados
2 from src.models.user_model import UserIn, UserOut
3
4 # Valida o usuario no banco de dados. Recebe como argumento um objeto
    UserIn
5 def validarUser(user: UserIn) -> bool:
6     conn = conexaoBancoDados()
7     cursor = conn.cursor()
8
9     cursor.execute("""SELECT id, username, password, tipo FROM Users
10                     WHERE username = ? AND password = ?;""", (user.username
11 , user.password))
12
13     query = cursor.fetchone()
14     conn.close()
15
16     if query != None:
17         return {
18             "status": True,
19             "user": UserOut(id=query[0], username=query[1], tipo=query[3])
20         }
21
22     return {
23         "status": False
24     }

```

```

24 # Adiciona um funcionario no banco de dados. Recebe um objeto UserIn
25 def adicionarFuncionario(func: UserIn) -> bool:
26     try:
27         conn = conexaoBancoDados()
28         cursor = conn.cursor()
29
30         cursor.execute("""INSERT INTO Users (username, password, tipo)
VALUES
31             (?, ?, 'funcionario');""", (func.username, func.
password))
32
33         conn.commit()
34         conn.close()
35         return True
36     except Exception as err:
37         print(err)
38         return False
39
40 # Retorna uma lista de dicionarios com todos os funcionarios
41 def getFuncionarios() -> list:
42     conn = conexaoBancoDados()
43     cursor = conn.cursor()
44     cursor.execute("""SELECT id, username FROM Users
45                     WHERE tipo='funcionario';""")
46
47     query = cursor.fetchall()
48     conn.close()
49     funcionarios = []
50     for func in query:
51         funcionarios.append({
52             "id": func[0],
53             "nome": func[1]
54         })
55     return funcionarios
56

```

```

1 from pydantic import
   BaseModel
2
3 # Classe que gerencia a
   entrada dos dados do
   usuario
4 class UserIn(BaseModel):
5     username: str
6     password: str
7
8 # Classe que gerencia a
   saida dos dados do
   usuario
9 class UserOut(BaseModel)
   :
10     id: int
11     username: str
12     tipo: str
13

```

Listing 1: user\_model.py

```

1 from pydantic import
   BaseModel
2
3 #Classe que gerencia a
   entrada de uma nova
   tarefa
4 class Tarefa(BaseModel):
5     nome: str
6     descricao: str
7     data_prevista_conclusao:
       str
8     funcionariosId: list
9
10
11
12
13

```

Listing 2: tarefa\_model.py

```

1 from pydantic import
   BaseModel
2
3 # Classe que gerencia a
   entrada de um novo
   relatorio
4 class Relatorio(
   BaseModel):
5     id_user: int
6     id_tarefa: int
7     texto: str
8     data_criacao: str
9
10
11
12
13

```

Listing 3: relatorio\_model.py

## 4.2 Cliente

```

1 import requests
2 import PySimpleGUI as sg
3 import json
4 from view import funcionario_view
5 from view import admin_view
6 from view import windows
7
8 url = "http://127.0.0.1:8000"
9
10 def main():
11     window_login = windows.login_window()
12     window_menu_admin = None
13     window_menu_func = None
14     window_admin_adicionar_func = None
15     window_admin_visualizar_func = None
16     window_admin_adicionar_tarefa = None
17     window_admin_visualizar_tarefas = None
18     window_admin_visualizar_relatorios = None
19     window_func_visualizar_tarefas = None
20     window_func_adicionar_relatorio = None
21     window_func_visualizar_relatorios = None
22
23     while True:
24         window, event, values = sg.read_all_windows()
25
26         if window == window_login and event == sg.WIN_CLOSED:
27             break
28         if window == window_login and event == "Continuar":
29
30             username = window_login["username"].get()
31             password = window_login["password"].get()
32             statusLogin = login(username, password)
33
34             if statusLogin.get("status") == True:
35                 window_login.hide()
36                 if statusLogin.get("tipo") == "admin":
37                     window_menu_admin = windows.menu_admin_window()
38                 else:
39                     window_menu_func = windows.menu_func_window()
40             else:
41                 window_login["incorrect_text"].update(visible=True)
42         if (window == window_menu_admin or window == window_menu_func) and (
43             event == sg.WIN_CLOSED or event == "Sair"):
44             window.close()
45             window_menu_admin = None
46             window_menu_func = None
47             window_login.un_hide()
48             window_login["incorrect_text"].update(visible=False)
49             window_login["username"].SetFocus()
50             window_login["username"].update("")
51             window_login["password"].Update("")
52         if event == sg.WIN_CLOSED or event == "Voltar":
53             if window_menu_admin != None:
54                 window_menu_admin.un_hide()
55             elif window_menu_func != None:
56                 window_menu_func.un_hide()
57             window.close()
58             window_admin_adicionar_func = None
59             window_admin_visualizar_func = None
60             window_admin_adicionar_tarefa = None

```

```

60         window_admin_visualizar_tarefas = None
61         window_admin_visualizar_relatorios = None
62         window_func_visualizar_tarefas = None
63         window_func_adicionar_relatorio = None
64         window_func_visualizar_relatorios = None
65
66         if window == window_menu_admin and event == "Adicionar Funcionario
67         ":
68             window_menu_admin.hide()
69             window_admin_adicionar_func = windows.adicionar_func_window()
70
71         if window == window_menu_admin and event == "Visualizar
72         Funcionarios":
73             window_menu_admin.hide()
74             window_admin_visualizar_func = windows.visualizar_funcs_window
75             ()
76
77         if window == window_menu_admin and event == "Criar Tarefa":
78             window_menu_admin.hide()
79             window_admin_adicionar_tarefa,numFuncionarios,
80             listaFuncionarios = windows.adicionar_tarefa_window()
81
82         if window == window_menu_admin and event == "Visualizar Tarefas":
83             window_menu_admin.hide()
84             window_admin_visualizar_tarefas = windows.
85             visualizar_tarefas_window()
86
87         if window == window_menu_admin and event == "Visualizar Relatorios
88         ":
89             window_menu_admin.hide()
90             window_admin_visualizar_relatorios = windows.
91             visualizar_relatorios_window()
92
93         if window == window_menu_func and event == "Visualizar Tarefas":
94             window_menu_func.hide()
95             window_func_visualizar_tarefas = windows.
96             visualizar_tarefas_funcionario_window()
97
98         if window == window_menu_func and event == "Adicionar Relatorio":
99             window_menu_func.hide()
100             window_func_adicionar_relatorio,numTarefas, listasTarefas =
101             windows.adicionar_relatorio_window()
102
103         if window == window_menu_func and event == "Visualizar Relatorios"
104         :
105             window_menu_func.hide()
106             window_func_visualizar_relatorios = windows.
107             visualizar_relatorios_window()
108
109         if event == "Cadastrar":
110             status=None
111             if window== window_admin_adicionar_func:
112                 nomeFunc=values["nomeFunc"]
113                 senhaFunc=values["senhaFunc"]
114                 status = admin_view.adicionarFuncionario(nomeFunc,
115                 senhaFunc)
116             elif window == window_admin_adicionar_tarefa:
117                 nomeTarefa = values["nomeTarefa"]
118                 descricao = values["descricao"]
119                 data_prevista_conclusao = values["data_prevista_conclusao"]

```

```

108         funcionariosId = []
109         i = 0
110         while i < numFuncionarios:
111             if values[f"{i}"]:
112                 funcionariosId.append(listaFuncionarios[i].get("id
113             i+=1
114         status=admin_view.criarTarefa(nomeTarefa,descricao,
data_prevista_conclusao,funcionariosId)
115         elif window==window_func_adicionar_relatorio:
116             textoRelatorio=values["texto_relatorio"]
117             idTarefa=[]
118             i=0
119             while i<numTarefas:
120                 if values[f"{i}"]:
121                     idTarefa.append(listasTarefas[i].get("id"))
122                 i+=1
123             status=funcionario_view.adicionarRelatorio(idTarefa[0],
textoRelatorio)
124             if status:
125                 window["status_true"].update(visible=True)
126             else:
127                 window["status_false"].update(visible=True)
128             window_login.close()
129
130 def login(username,password):
131
132     request = requests.post(url+"/auth", json={"username": username, "
password": password})
133
134     if request.status_code == 200:
135         # Abre o arquivo user.json para escrita
136         file = open("user.json", "w")
137         # Escreve os dados da resposta do servidor no arquivo user.json
138         json.dump(request.json(), file)
139         # Fecha o arquivo
140         file.close()
141         if request.json()["tipo"] == "admin":
142             return {
143                 "status": True,
144                 "tipo":"admin"
145             }
146         elif request.json()["tipo"] == "funcionario":
147             return{
148                 "status": True,
149                 "tipo":"funcionario"
150             }
151         else:
152             return{
153                 "status": False
154             }
155
156 if __name__ == "__main__":
157     main()

```

Listing 4: app.py

```

1 import PySimpleGUI as sg
2 from view import admin_view
3 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')

```

```

59     layout = [
60         [sg.Text("Nome da Tarefa")],
61         [sg.Input(key="nomeTarefa")],
62         [sg.Text("Descricao da tarefa")],
63         [sg.Input(key="descricao")],
64         [sg.Text("Data Prevista de Conclusao (dd-mm-aa)")],
65         [sg.Input(key="data_prevista_conclusao")],
66         [sg.Text("Selecione os Funcionarios:")]
67     ]
68
69     listaFuncionarios = admin_view.visualizarFuncionarios()
70     c = 0
71     column = []
72
73     if listaFuncionarios == []:
74         column.append([sg.Text("Sem funcionarios cadastrados!")])
75     else:
76         for funcionario in listaFuncionarios:
77             column.append([sg.Checkbox(funcionario["nome"],key="{}".format
(c))])
78             c += 1
79
80     layout.append([sg.Column(column, size=(200, 200), scrollable=True,
vertical_scroll_only=True)])
81
82     layout.append([sg.Text("Tarefa cadastrado com sucesso!", text_color="
white", background_color="green", visible=False, key="status_true")])
83
84     layout.append([sg.Text("Erro ao cadastrar a tarefa! \nTente Novamente!
", text_color="white", background_color="red", visible=False, key="
status_false")])
85
86     layout.append([sg.Button("Cadastrar")])
87
88     layout.append([sg.Button("Voltar")])
89
90     return sg.Window("Criar Tarefa", layout=layout, finalize=True), c,
listaFuncionarios
91
92 def visualizar_funcs_window():
93     sg.theme('GreenTan')
94     layout = [
95         [sg.Text("Funcionarios: ")],
96     ]
97
98     listaFuncionarios = admin_view.visualizarFuncionarios()
99
100    if listaFuncionarios == []:
101        layout.append([sg.Text("Sem funcionarios cadastrados!")])
102    else:
103        column = []
104        for funcionario in listaFuncionarios:
105            column.append([sg.Text("-----")])
106            for key, value in funcionario.items():
107                column.append([sg.Text("{}: {}".format(key, value)])])
108            layout.append([sg.Column(column, size=(500, 500), scrollable=True,
vertical_scroll_only=True)])
109
110    layout.append([sg.Button("Voltar")])
111

```



```

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

```

```

221     listaRelatorios = funcionario_view.visualizarRelatorios()
222
223     if listaRelatorios == []:
224         layout.append([sg.Text("Sem relatorios cadastrados!")])
225     else:
226         column = []
227         for relatorio in listaRelatorios:
228             column.append([sg.Text("-----")])
229             for key, value in relatorio.items():
230                 column.append([sg.Text("{}: {}".format(key, value))])
231         layout.append([sg.Column(column, size=(500, 500), scrollable=True,
vertical_scroll_only=True)])
232
233     layout.append([sg.Button("Voltar")])
234
235     return sg.Window("Visualizar Relatorios", layout=layout, finalize=True
)

```

Listing 5: windows.py

```

1 import requests
2 import json
3
4 url = "http://127.0.0.1:8000"
5
6 def adicionarFuncionario(usernameFunc, passwordFunc):
7     with open("user.json", "r") as file:
8         user_dados = json.load(file)
9         token = user_dados["token"]
10
11     request = requests.put(url+"/admin/funcionarios?token="+token, json={"
username": usernameFunc, "password": passwordFunc})
12
13     if request.status_code == 200:
14         return True
15     else:
16         return False
17
18
19 def visualizarFuncionarios():
20
21     with open("user.json", "r") as file:
22         user_dados = json.load(file)
23         token = user_dados["token"]
24
25     request = requests.get(url+"/admin/funcionarios?token="+token)
26     return request.json()
27
28 def criarTarefa(nome, descricao, data_prevista_conclusao, funcionariosId):
29
30     with open("user.json", "r") as file:
31         user_dados = json.load(file)
32         token = user_dados["token"]
33
34     tarefa = {
35         "nome": nome,
36         "descricao": descricao,
37         "data_prevista_conclusao": data_prevista_conclusao,
38         "funcionariosId": funcionariosId
39     }
40     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}",
33                             json=relatorio)
34
35     if request.status_code == 200:
36         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 avanços obtidos e as metas estabelecidas no crédito anterior, criamos na parte do servidor a adição de um token de validação para a segurança do sistema, e na parte do cliente a criação de uma interface gráfica, que permitiu a visualização 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 py-simplegui 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](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/Python00.html](https://supygirls.readthedocs.io/en/latest/intro_comp/Python00.html), 2024. Accessed: 2024-06-01.