Gerenciador de tarefas

Guilherme Lopes e Sara Stephanie

Universidade do Estado da Bahia

2 de junho de 2024

Sumário

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Sumário

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Introdução

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Introdução: Objetivo

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

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Introdução: 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.

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Introdução: 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.

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Referencial Teórico

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Referencial Teórico: Bibliotecas

- FastAPI
- Uvicorn
- Requests
- Pydantic

Referencial Teórico: SQL

A Linguagem de consulta estruturada (SQL) é uma linguagem utilizada 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.

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Referencial Teórico: 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.

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Referencial Teórico: 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

Proposta do programa

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

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.

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Proposta do programa: Funcionamento do programa

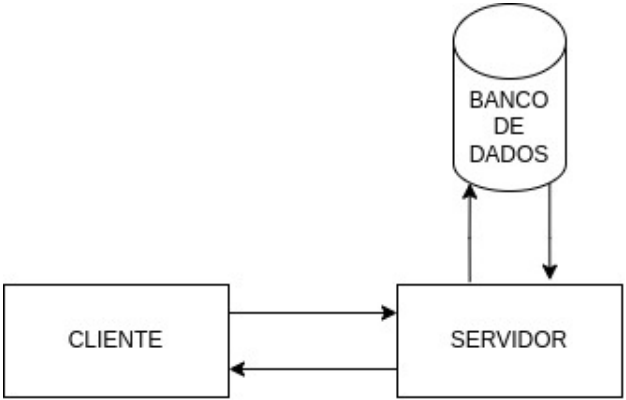


Figura: Funcionamento do programa

Banco de dados

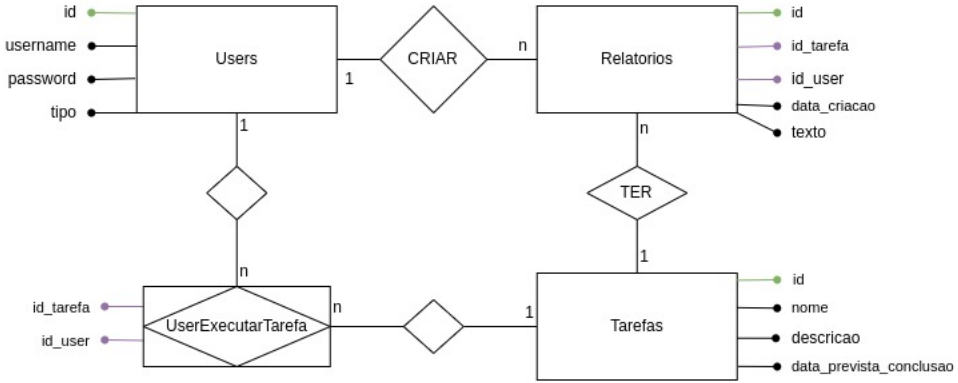
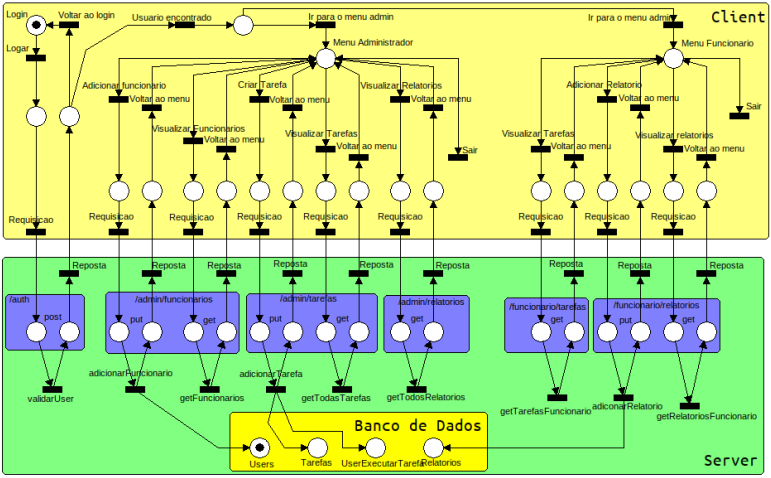
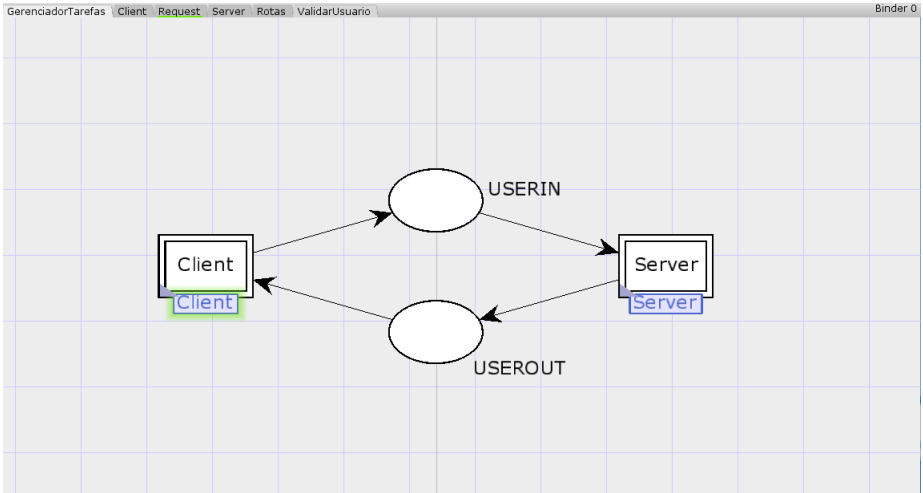


Figura: Diagrama entidade relacionamento

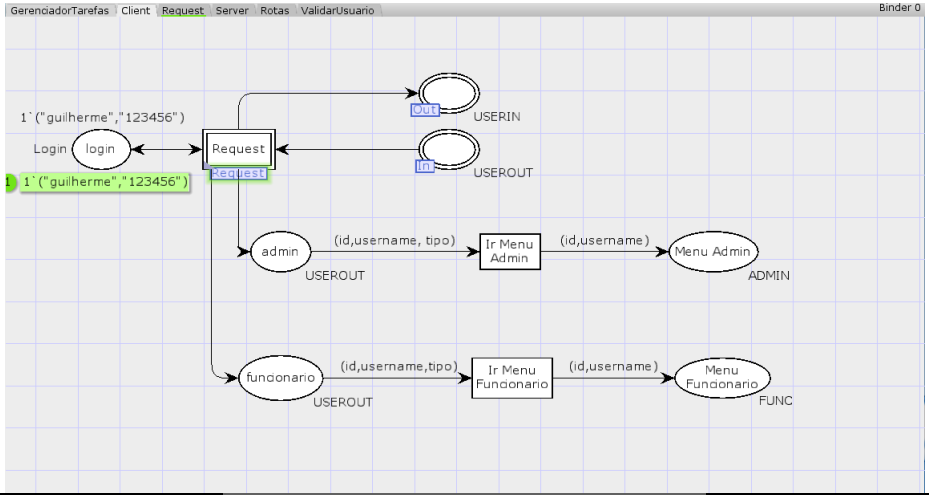
Proposta do programa: Rede de Petri Ordinaria



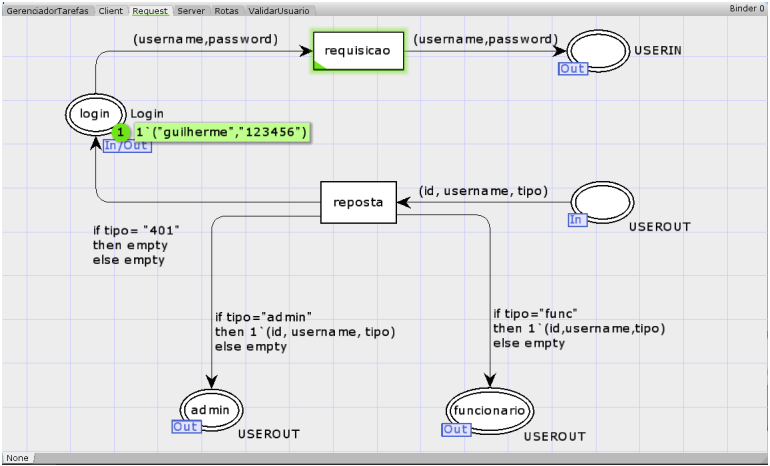
Redes de Petri: Rede de Petri Colorida



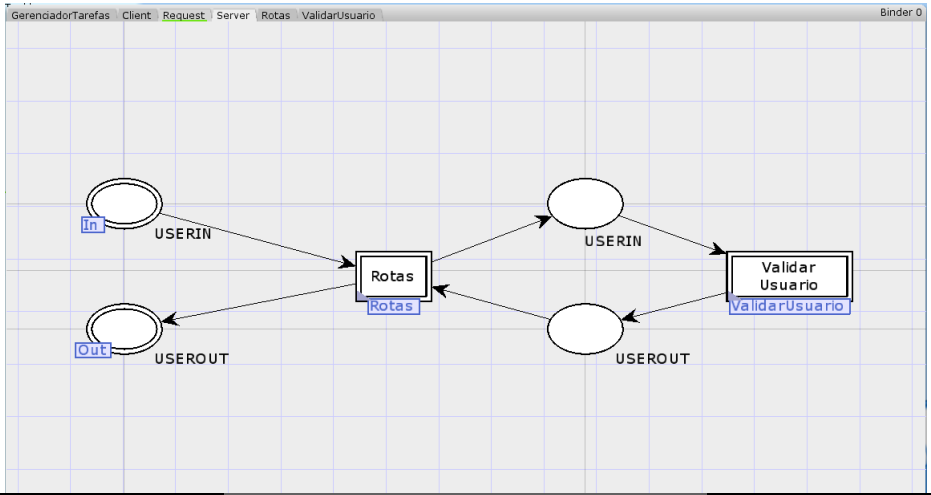
Redes de Petri: Rede de Petri Colorida



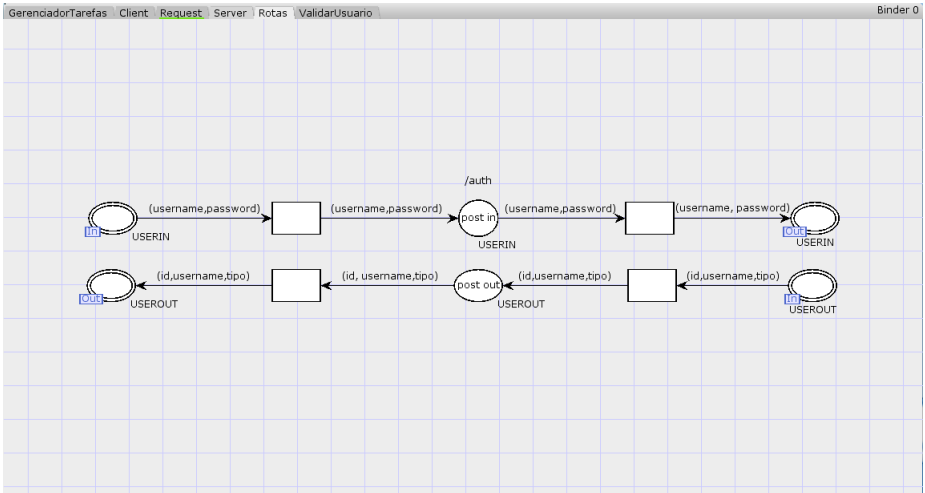
Redes de Petri: Rede de Petri Colorida



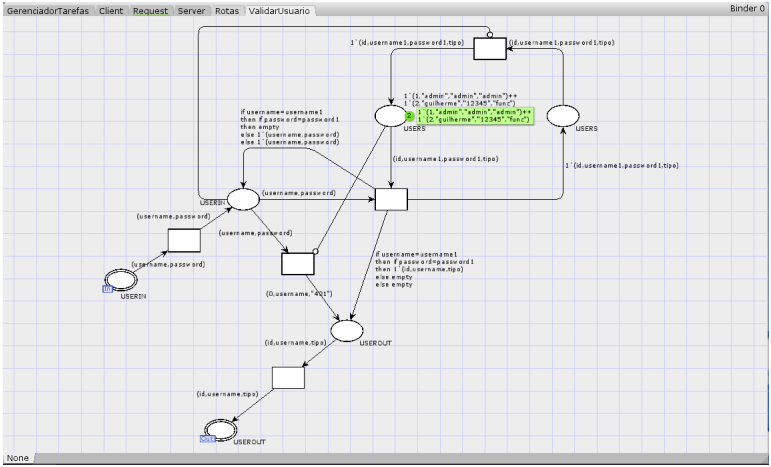
Redes de Petri: Rede de Petri Colorida



Redes de Petri: Rede de Petri Colorida



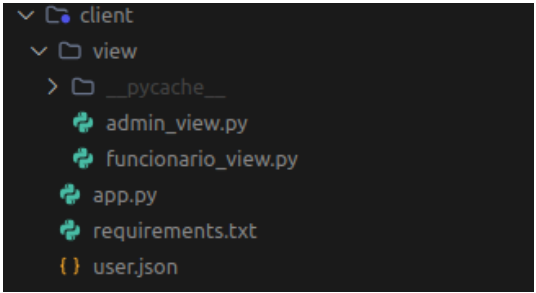
Redes de Petri: Rede de Petri Colorida

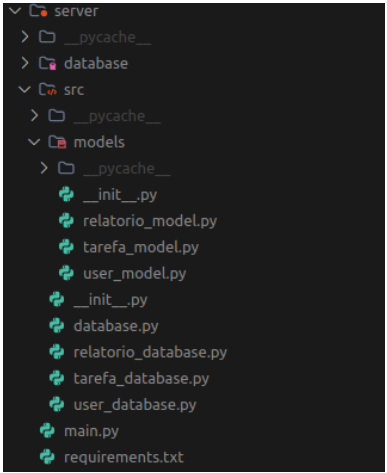


Codigo do Programa

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Estrutura do Projeto





UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Servidor: main

```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
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)

```

Servidor: main

```
25      if validarUser.get("status"):
26          return validarUser.get("user")
27      else:
28          raise HTTPException(
29              status_code=status.HTTP_401_UNAUTHORIZED,
30              detail="Usuario nao encontrado!"
31          )
32
33  # Rota para adicionar funcionario, recebe nome de usuario e senha
34  @app.put("/admin/funcionarios")
35  async def adicionar_funcionario(func: UserIn):
36      if user_database.adicionarFuncionario(func=func):
37          return "Funcionario Adicionado com Sucesso!"
38      else:
39          raise HTTPException(
40              status_code=status.HTTP_501_NOT_IMPLEMENTED,
41              detail="Erro ao adicionar novo funcionario!"
42          )
43
44  # Rota que lista todos os funcionario cadastrados no banco de dados
45  @app.get("/admin/funcionarios")
46  async def lista_funcionarios():
47      return user_database.getFuncionarios()
```

Servidor: main

```
48
49 # Rota que adiciona uma nova tarefa do banco de dados. Recebe nome, descricao, data prevista de conclusao e id
       dos funcionarios
50 @app.put("/admin/tarefas")
51 async def adicionar_tarefa(tarefa: Tarefa):
52     if tarefa_database.adicionarTarefa(tarefa=tarefa):
53         return "Tarefa adicionada com Sucesso!"
54     else:
55         raise HTTPException(
56             status_code=status.HTTP_501_NOT_IMPLEMENTED,
57             detail="Erro ao adicionar tarefa!"
58         )
59
60 # Rota que lista todas as tarefas cadastradas no sistema
61 @app.get("/admin/tarefas")
62 async def get_todas_tarefas():
63     return tarefa_database.getTodasTarefas()
64
65 # Rota que lista todos os relatorios cadastrados no sistema
66 @app.get("/admin/relatorios")
67 async def get_todos_relatorios():
68     return relatorio_database.getTodosRelatorios()
69
```

Servidor: main

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

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Codigo do Programa: database

```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(
```

Codigo do Programa: database

```
25                  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
```

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Codigo do Programa: database

```
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) VALUES ('admin', 'admin', 'admin');''')
53
54      conn.commit()
55      print("Conexao com banco de dados bem sucedida!")
56      conn.close()
57
```

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Servidor: tarefa_database

```python
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))

```

Servidor: tarefa_database

```
25          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
```

Servidor: tarefa_database

```
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()
```

Servidor: tarefa_database

```
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
```

Servidor: relatorio_database

```python
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:
```

Servidor: relatorio_database

```
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()
```

Servidor: relatorio_database

```
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({
```

Servidor: relatorio_database

```
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
```

Servidor: user_database

```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
        }

# Adiciona um funcionario no banco de dados. Recebe um objeto UserIn
```

UNEB
IVERSIDADE DO
TADO DA BAHIA

Servidor: user_database

```
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()
```

Servidor: user_database

```
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
```

Servidor: user_model, tarefa_model, relatorio_model

Listing: user_model.py

```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: tarefa_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: relatorio_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
```

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Cliente: app

```python
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)")
```

Cliente: app

```
25
26      file = open("user.json", "w")
27      json.dump(request.json(), file)
28      file.close()
29
30      print("Logado com sucesso!")
31      input("(PRESSIONE ENTER PARA CONTINUAR)")
32
33      if request.json()["tipo"] == "admin":
34          menuAdmin()
35      elif request.json()["tipo"] == "funcionario":
36          menuFunc()
37
38  if __name__ == "__main__":
39      login()
```

Cliente: admin_view

```python
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:")
```

Cliente: admin_view

```
25
26          option = input(">> ").strip()
27
28          try:
29              (menuOptions[option][1])()
30          except Exception:
31              pass
32
33  def adicionarFuncionario():
34      os.system("clear")
35
36      print("-"*10, " Adicionar Funcionario ", "-"*10)
37
38      print("Nome do funcionario: ")
39      usernameFunc = input(">> ")
40      print("Senha do funcionario: ")
41      passwordFunc = input(">> ")
42
43      request = requests.put(url+"/admin/funcionarios", json={"username": usernameFunc, "password": passwordFunc
          })
44
45      if request.status_code == 200:
46          print("Funcionario adicionado com sucesso!")
```

Cliente: admin_view

```
47      else:
48          print("Erro ao adicionar funcionario! Tente novamente.")
49
50      input("(PRESSIONE ENTER PARA VOLTAR AO MENU)")
51
52  def visualizarFuncionarios():
53      os.system("clear")
54
55      print("-"*5, " Visualizar Funcionarios ", "-"*5)
56
57      request = requests.get(url+"/admin/funcionarios")
58
59      if request.json() == []:
60          print("Sem funcionarios cadastrados!")
61      else:
62          for funcionario in request.json():
63              print("-"*30)
64              for key, value in funcionario.items():
65                  print(f"{key}: {value}")
66              print("-"*30)
67
68      input("(PRESSIONE ENTER PARA VOLTAR AO MENU)")
69
```

Cliente: admin_view

```
70 def criarTarefa():
71     os.system("clear")
72 
73     print("-"*10, " Criar Tarefa ", "-"*10)
74     tarefa = {
75         "nome": "",
76         "descricao": "",
77         "data_prevista_conclusao": "",
78         "funcionariosId": []
79     }
80 
81     print("Digite o nome da tarefa: ")
82     tarefa["nome"] = input(">> ")
83 
84     print("Digite a descricao da tarefa: ")
85     tarefa["descricao"] = input(">> ")
86 
87     print("Digite a data prevista para conclusao(formato dd-mm-aaaa): ")
88     tarefa["data_prevista_conclusao"] = input(">> ")
89 
90     request = requests.get(url+"/admin/funcionarios")
91 
92     print("Funcionarios")
```

Cliente: admin_view

```
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:
```

Cliente: admin_view

```
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}")
```

Cliente: admin_view

```
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() == []:
151         print("Sem relatorios cadastradas!")
152     else:
153         for relatorio in request.json():
154             print("-"*60)
155             for key, value in relatorio.items():
156                 print(f"{key}: {value}")
157             print("-"*60)
158     input("(PRESSIONE ENTER PARA VOLTAR AO MENU)")
```

Cliente: funcionario_view

```
1  import requests
2  import os
3  import sys
4  import json
5  from datetime import date
6
7  url = "http://127.0.0.1:8000"
8
9  def menuFunc():
10     while True:
11         os.system("clear")
12
13         menuOptions = {
14             "1": ["Visualizar Tarefas", visualizarTarefas],
15             "2": ["Adicionar Relatorio", adicionarRelatorio],
16             "3": ["Visualizar Relatorios", visualizarRelatorios],
17             "4": ["Sair", sys.exit],
18         }
19
20         print("-"*10, 'FUNCIONARIO', "-"*10)
21         for key, value in menuOptions.items():
22             print(key, " - ", value[0])
23
24         print("Escolha uma das opcoes:")
```

Cliente: funcionario_view

```
25
26          option = input(">> ").strip()
27
28          try:
29              (menuOptions[option][1])()
30          except Exception:
31              pass
32
33  def visualizarTarefas():
34      os.system("clear")
35
36      print("-"*10, " Visualizar Tarefas ", "-"*10)
37
38      file = open("user.json", "r")
39      idUser = (json.load(file))["id"]
40      file.close()
41
42      request = requests.get(f"{url}/funcionario/tarefas?idUser={idUser}")
43
44      if request.json() == []:
45          print("Sem tarefas cadastradas!")
46      else:
47          for tarefa in request.json():
```

Cliente: funcionario_view

```
48              print("-"*60)
49              for key, value in tarefa.items():
50                  print(f"{key}: {value}")
51              print("-"*60)
52
53      input("(PRESSIONE ENTER PARA VOLTAR AO MENU)")
54
55  def adicionarRelatorio():
56      os.system("clear")
57
58      print("-"*10, "Adicionar Relatorios", "-"*10)
59
60      file = open("user.json", "r")
61      idUser = (json.load(file))["id"]
62      file.close()
63
64      request = requests.get(f"{url}/funcionario/tarefas?idUser={idUser}")
65
66      print("Tarefas: ")
67
68      for tarefa in request.json():
69          print("   id: {}, nome: {}".format(tarefa.get("id"), tarefa.get("nome")))
70
```

Cliente: funcionario_view

```
71      print("Digite o id da tarefa que voce deseja fazer o relatorio: ")
72      idTarefa = int(input(">> "))
73
74      print("Digite o texto do relatorio: ")
75      textoRelatorio = input(">> ")
76
77      dataAtual = date.today().strftime("%d-%m-%Y")
78
79      relatorio = {
80          "id_user": idUser,
81          "id_tarefa": idTarefa,
82          "texto": textoRelatorio,
83          "data_criacao": dataAtual
84      }
85
86      request = requests.put(f"{url}/funcionario/relatorios", json=relatorio)
87
88      if request.status_code == 200:
89          print("Relatorio adicionado com sucesso!")
90      else:
91          print("Erro ao adicionar relatorio! Tente novamente.")
92
93      input("(PRESSIONE ENTER PARA VOLTAR MENU)")
```

Cliente: funcionario_view

```
94  def visualizarRelatorios():
95      os.system("clear")
96      print("-"*10, "Visualizar Relatorios", "-"*10)
97      file = open("user.json", "r")
98      idUser = (json.load(file))["id"]
99      file.close()
100
101     request = requests.get(f"{url}/funcionario/relatorios?idUser={idUser}")
102
103     if request.json() == []:
104         print("Sem relatorios cadastradas!")
105     else:
106         for relatorio in request.json():
107             print("-"*60)
108             for key, value in relatorio.items():
109                 print(f"{key}: {value}")
110             print("-"*60)
111
112     input("(PRESSIONE ENTER PARA VOLTAR AO MENU)")
```

Conclusões parciais

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

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.

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Referências

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Referências

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

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Referências

- First steps - fastapi.
  https://fastapi.tiangolo.com/tutorial/first-steps/, 2024.
  Accessed: 2024-06-01.
- Curso de banco de dados mysql - youtube.
  https://www.youtube.com/playlist?list=PLHz_AreHm4dkBs-795Dsgvau_ekxg8g1r, 2024.
  Accessed: 2024-06-01.

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Referências

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

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Referências

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

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Referências

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

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Referências

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

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA

Esta apresentação foi criada utilizando o template disponibilizado pelo CEFET-RJ, licenciado sob a GPL 3. O template pode ser encontrado em https://github.com/hsneto/if-beamer/blob/master/LICENSE.

UNEB
UNIVERSIDADE DO ESTADO DA BAHIA