** Projeto J - Sistema de Chamados

Aluno : Adalberto Caldeira Brant Filho

link github

Estrutura do projeto

```
tree
.
├── admin_roteiro.txt
├── data
│   ├── chamados.csv
│   ├── senhas.csv
│   ├── setores.csv
│   ├── tecnicos.csv
│   └── usuarios.csv
├── erros.txt
├── main.py
├── models
│   ├── chamado.py
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── chamado.cpython-311.pyc
│   │   ├── chamado.cpython-312.pyc
│   │   ├── __init__.cpython-311.pyc
│   │   ├── __init__.cpython-312.pyc
│   │   ├── setor.cpython-311.pyc
│   │   ├── setor.cpython-312.pyc
│   │   ├── tecnico.cpython-311.pyc
│   │   ├── tecnico.cpython-312.pyc
│   │   ├── usuario.cpython-311.pyc
│   │   └── usuario.cpython-312.pyc
│   ├── setor.py
│   ├── tecnico.py
│   └── usuario.py
├── README.md
├── relatorio
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-311.pyc
│   │   ├── __init__.cpython-312.pyc
│   │   ├── relatorios.cpython-311.pyc
│   │   └── relatorios.cpython-312.pyc
│   └── relatorios.py
├── repositorio
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-311.pyc
│   │   ├── __init__.cpython-312.pyc
│   │   ├── repositorio_autenticacao.cpython-312.pyc
│   │   ├── repositorio_chamados.cpython-311.pyc
```

```
│   │   ├── repositorio_chamados.cpython-312.pyc
│   │   ├── repositorio_setores.cpython-311.pyc
│   │   ├── repositorio_setores.cpython-312.pyc
│   │   ├── repositorio_tecnicos.cpython-311.pyc
│   │   ├── repositorio_tecnicos.cpython-312.pyc
│   │   ├── repositorio_usuarios.cpython-311.pyc
│   │   └── repositorio_usuarios.cpython-312.pyc
│   ├── repositorio_autenticacao.py
│   ├── repositorio_chamados.py
│   ├── repositorio_setores.py
│   ├── repositorio_tecnicos.py
│   └── repositorio_usuarios.py
└── user_roteiro.txt
```

*** MAIN

```python
# main.py

from datetime import datetime
from models.chamado import Chamado, Prioridade, StatusChamado
from models.setor import Setor
from models.tecnico import Tecnico
from models.usuario import Usuario
from relatorio import relatorios
from repositorio.repositorio_chamados import RepositorioChamados
from repositorio.repositorio_setores import RepositorioSetores
from repositorio.repositorio_tecnicos import RepositorioTecnicos
from repositorio.repositorio_usuarios import RepositorioUsuarios
from repositorio.repositorio_autenticacao import RepositorioAutenticacao


def exibir_lista_com_indices(itens, atributo_id, atributo_display):
    for i, item in enumerate(itens):
        print(f"{i + 1}. {getattr(item, atributo_display)} (ID: {getattr(item,
atributo_id)})")

def selecionar_opcao_por_lista(itens, atributo_id, atributo_display, prompt):
    if not itens:
        print("Nenhum item disponível para seleção.")
        return None

    while True:
        exibir_lista_com_indices(itens, atributo_id, atributo_display)
        try:
            escolha = int(input(prompt))
            if 1 <= escolha <= len(itens):
                return getattr(itens[escolha - 1], atributo_id)
            else:
                print("Escolha inválida. Tente novamente.")
        except ValueError:
            print("Entrada inválida. Digite um número.")


def login(repo_autenticacao: RepositorioAutenticacao):
```

```
    while True:
        print("\n--- Login ---")
        username = input("Usuário: ")
        password = input("Senha: ")
        role, identity = repo_autenticacao.autenticar(username, password)
        if role and identity:
            print(f"Login bem-sucedido! Bem-vindo, {username} ({role}).")
            return role, identity
        elif role == 'admin' and not identity:
            print(f"Login bem-sucedido! Bem-vindo, {username} ({role}).")
            return role, None # Admin might not have a specific identity like CPF
        else:
            print("Usuário ou senha inválidos. Tente novamente.")

def menu_usuarios(repo_usuarios: RepositorioUsuarios, repo_setores:
RepositorioSetores):
    while True:
        print("\n--- Menu Usuários ---")
        print("1. Cadastrar Usuário")
        print("2. Listar Usuários")
        print("0. Voltar")
        opcao = input("Escolha uma opção: ")

        if opcao == "1":
            cpf = input("CPF: ")
            nome = input("Nome: ")
            email = input("Email: ")

            setores_disponiveis = repo_setores.obter_todos()
            id_setor_selecionado =
selecionar_opcao_por_lista(setores_disponiveis, 'id', 'nome', "Selecione o Setor
do Usuário: ")

            if id_setor_selecionado is None:
                print("Nenhum setor foi selecionado. Cadastro cancelado.")
                continue

            try:
                usuario = Usuario(cpf, nome, email, id_setor_selecionado)
                repo_usuarios.adicionar(usuario)
                print("Usuário cadastrado com sucesso!")
            except ValueError as e:
                print(f"Erro ao cadastrar usuário: {e}")
        elif opcao == "2":
            for usuario in repo_usuarios.obter_todos():
                print(f"CPF: {usuario.cpf}, Nome: {usuario.nome}, Email:
{usuario.email}, Setor: {usuario.id_setor}")
        elif opcao == "0":
            break
        else:
            print("Opção inválida.")


def menu_tecnicos(repo_tecnicos: RepositorioTecnicos):
    while True:
```

```
        print("\n--- Menu Técnicos ---")
        print("1. Cadastrar Técnico")
        print("2. Listar Técnicos")
        print("0. Voltar")
        opcao = input("Escolha uma opção: ")

        if opcao == "1":
            nome = input("Nome: ")
            email = input("Email: ")
            especialidade = input("Especialidade: ")
            try:
                # O ID é gerado automaticamente pelo repositório
                tecnico = repo_tecnicos.adicionar(nome, email, especialidade)
                print(f"Técnico cadastrado com sucesso! ID: {tecnico.id}")
            except ValueError as e:
                print(f"Erro ao cadastrar técnico: {e}")
        elif opcao == "2":
            for tecnico in repo_tecnicos.obter_todos():
                print(f"ID: {tecnico.id}, Nome: {tecnico.nome}, Email:
{tecnico.email}, Especialidade: {tecnico.especialidade}")
        elif opcao == "0":
            break
        else:
            print("Opção inválida.")


def menu_setores(repo_setores: RepositorioSetores):
    while True:
        print("\n--- Menu Setores ---")
        print("1. Cadastrar Setor")
        print("2. Listar Setores")
        print("0. Voltar")
        opcao = input("Escolha uma opção: ")

        if opcao == "1":
            nome = input("Nome: ")

            # Precisamos do ID do gestor. Assumindo que gestores são usuários.
            # Se gestores forem uma entidade separada, precisaríamos de outro
repo/lista.
            # Por simplicidade, vamos permitir input manual por enquanto ou
selecionar de uma lista de usuários
            # Mas, como id_gestor é um ID, e usuários tem CPF, vamos tratar como
um ID numérico.
            # Para este cenário, vamos manter o input manual para id_gestor, mas
o ID do setor é automático.
            id_gestor = int(input("ID do Gestor: ")) # Manter manual para este
campo

            try:
                # O ID do setor é gerado automaticamente
                setor = repo_setores.adicionar(nome, id_gestor)
                print(f"Setor cadastrado com sucesso! ID: {setor.id}")
            except ValueError as e:
                print(f"Erro ao cadastrar setor: {e}")
```

```
        elif opcao == "2":
            for setor in repo_setores.obter_todos():
                print(f"ID: {setor.id}, Nome: {setor.nome}, Gestor:
{setor.id_gestor}")
        elif opcao == "0":
            break
        else:
            print("Opção inválida.")


def menu_chamados(repo_chamados: RepositorioChamados, repo_usuarios:
RepositorioUsuarios, repo_tecnicos: RepositorioTecnicos, repo_setores:
RepositorioSetores, user_role: str, user_identity: str | None):
    while True:
        print("\n--- Menu Chamados ---")
        print("1. Abrir Chamado")

        if user_role == "admin":
            print("2. Atribuir Técnico")
            print("3. Atualizar Status")
            print("4. Listar Todos os Chamados")
        else:
            print("2. Listar Meus Chamados")
            print("3. Ver Status de um Chamado")

        print("0. Voltar")
        opcao = input("Escolha uma opção: ")

        if opcao == "1": # Abrir Chamado
            cpf_usuario = user_identity # Para usuário comum, o CPF já está
definido

            if user_role == "admin": # Admin pode abrir chamado para qualquer
usuário
                usuarios_disponiveis = repo_usuarios.obter_todos()
                cpf_usuario = selecionar_opcao_por_lista(usuarios_disponiveis,
'cpf', 'nome', "Selecione o Usuário que está abrindo o Chamado (CPF): ")
                if cpf_usuario is None:
                    print("Nenhum usuário selecionado. Abertura de chamado
cancelada.")
                    continue

            titulo = input("Título: ")
            descricao = input("Descrição: ")

            setores_disponiveis = repo_setores.obter_todos()
            id_setor_selecionado =
selecionar_opcao_por_lista(setores_disponiveis, 'id', 'nome', "Selecione o Setor
associado ao Chamado: ")
            if id_setor_selecionado is None:
                print("Nenhum setor selecionado. Abertura de chamado cancelada.")
                continue

            prioridade_str = input("Prioridade (baixa, media, alta): ").lower()
            try:
```

```
                prioridade = Prioridade(prioridade_str)
                # O ID do chamado é gerado automaticamente pelo repositório
                chamado_aberto = repo_chamados.adicionar(cpf_usuario, titulo,
descricao, id_setor_selecionado, prioridade)
                print(f"Chamado aberto com sucesso! ID: {chamado_aberto.id}")
            except (ValueError, KeyError) as e:
                print(f"Erro ao abrir chamado: {e}")

        elif opcao == "2":
            if user_role == "admin": # Admin option: Atribuir Técnico
                chamados_abertos = [c for c in repo_chamados.obter_todos() if
c.status != StatusChamado.RESOLVIDO and c.status != StatusChamado.CANCELADO]
                id_chamado_selecionado =
selecionar_opcao_por_lista(chamados_abertos, 'id', 'titulo', "Selecione o Chamado
para atribuir (ID): ")
                if id_chamado_selecionado is None:
                    print("Nenhum chamado selecionado. Atribuição cancelada.")
                    continue

                tecnicos_disponiveis = repo_tecnicos.obter_todos()
                id_tecnico_selecionado =
selecionar_opcao_por_lista(tecnicos_disponiveis, 'id', 'nome', "Selecione o
Técnico para atribuir (ID): ")
                if id_tecnico_selecionado is None:
                    print("Nenhum técnico selecionado. Atribuição cancelada.")
                    continue

                chamado = repo_chamados.buscar_por_id(id_chamado_selecionado)
                if chamado:
                    chamado.id_tecnico = id_tecnico_selecionado
                    chamado.status = StatusChamado.EM_ATENDIMENTO
                    repo_chamados.atualizar(chamado)
                    print("Técnico atribuído com sucesso!")
                else:
                    print("Chamado não encontrado.")
            else: # User option: Listar Meus Chamados
                chamados_do_usuario = [c for c in repo_chamados.obter_todos() if
c.usuario == user_identity]
                if chamados_do_usuario:
                    for c in chamados_do_usuario:
                        print(f"ID: {c.id}, Título: {c.titulo}, Status:
{c.status.value}, Prioridade: {c.prioridade.value}")
                else:
                    print("Você não possui chamados abertos.")

        elif opcao == "3":
            if user_role == "admin": # Admin option: Atualizar Status
                chamados_para_atualizar = [c for c in repo_chamados.obter_todos()
if c.status != StatusChamado.RESOLVIDO and c.status != StatusChamado.CANCELADO]
                id_chamado_selecionado =
selecionar_opcao_por_lista(chamados_para_atualizar, 'id', 'titulo', "Selecione o
Chamado para atualizar status (ID): ")
                if id_chamado_selecionado is None:
                    print("Nenhum chamado selecionado. Atualização cancelada.")
                    continue
```

```
                status_str = input("Novo Status (aberto, em atendimento,
resolvido, cancelado): ").lower()
                chamado = repo_chamados.buscar_por_id(id_chamado_selecionado)
                if chamado:
                    try:
                        status = StatusChamado(status_str)
                        chamado.status = status
                        if status in [StatusChamado.RESOLVIDO,
StatusChamado.CANCELADO]:
                            chamado.data_fechamento = datetime.now()
                        repo_chamados.atualizar(chamado)
                        print("Status atualizado com sucesso!")
                    except (ValueError, KeyError) as e:
                        print(f"Erro ao atualizar status: {e}")
                else:
                    print("Chamado não encontrado.")
            else: # User option: Ver Status de um Chamado
                chamados_do_usuario = [c for c in repo_chamados.obter_todos() if
c.usuario == user_identity]
                id_chamado_selecionado =
selecionar_opcao_por_lista(chamados_do_usuario, 'id', 'titulo', "Selecione o
Chamado para verificar status (ID): ")
                if id_chamado_selecionado is None:
                    print("Nenhum chamado selecionado.")
                    continue

                chamado = repo_chamados.buscar_por_id(id_chamado_selecionado)
                if chamado and chamado.usuario == user_identity:
                    print(f"Chamado ID: {chamado.id}, Título: {chamado.titulo},
Status: {chamado.status.value}, Prioridade: {chamado.prioridade.value}")
                else:
                    print("Chamado não encontrado ou você não tem permissão para
vê-lo.")
        elif opcao == "4":
            if user_role == "admin": # Admin option: Listar Todos os Chamados
                all_chamados = repo_chamados.obter_todos()
                if all_chamados:
                    for c in all_chamados:
                        print(f"ID: {c.id}, Usuário: {c.usuario}, Título:
{c.titulo}, Status: {c.status.value}, Prioridade: {c.prioridade.value}, Técnico:
{c.id_tecnico if c.id_tecnico else 'N/A'}")
                else:
                    print("Não há chamados cadastrados.")
            else:
                print("Opção inválida para seu perfil.")
        elif opcao == "0":
            break
        else:
            print("Opção inválida.")


def menu_relatorios(repo_chamados: RepositorioChamados, repo_usuarios:
RepositorioUsuarios, repo_tecnicos: RepositorioTecnicos, user_role: str,
user_identity: str | None):
```

```
    while True:
        print("\n--- Menu Relatórios ---")
        if user_role == "admin":
            print("1. Listar Chamados Abertos")
            print("2. Listar Chamados por Usuário")
            print("3. Listar Chamados por Técnico")
        else: # Regular user
            print("1. Listar Meus Chamados Abertos")
            print("2. Listar Todos os Meus Chamados")
        print("0. Voltar")
        opcao = input("Escolha uma opção: ")

        if user_role == "admin":
            if opcao == "1":
                relatorios.listar_chamados_abertos()
            elif opcao == "2":
                usuarios_disponiveis = repo_usuarios.obter_todos()
                cpf_usuario_selecionado =
selecionar_opcao_por_lista(usuarios_disponiveis, 'cpf', 'nome', "Selecione o
Usuário para listar chamados (CPF): ")
                if cpf_usuario_selecionado is None:
                    print("Nenhum usuário selecionado. Operação cancelada.")
                    continue
                relatorios.listar_chamados_por_usuario(cpf_usuario_selecionado)
            elif opcao == "3":
                tecnicos_disponiveis = repo_tecnicos.obter_todos()
                id_tecnico_selecionado =
selecionar_opcao_por_lista(tecnicos_disponiveis, 'id', 'nome', "Selecione o
Técnico para listar chamados (ID): ")
                if id_tecnico_selecionado is None:
                    print("Nenhum técnico selecionado. Operação cancelada.")
                    continue
                relatorios.listar_chamados_por_tecnico(id_tecnico_selecionado)
            elif opcao == "0":
                break
            else:
                print("Opção inválida.")
        else: # Regular user
            if opcao == "1":
                chamados_abertos_do_usuario = [c for c in
repo_chamados.obter_todos() if c.usuario == user_identity and c.status ==
StatusChamado.ABERTO]
                if chamados_abertos_do_usuario:
                    for c in chamados_abertos_do_usuario:
                        print(f"ID: {c.id}, Título: {c.titulo}, Status:
{c.status.value}, Prioridade: {c.prioridade.value}")
                else:
                    print("Você não possui chamados abertos.")
            elif opcao == "2":
                chamados_do_usuario = [c for c in repo_chamados.obter_todos() if
c.usuario == user_identity]
                if chamados_do_usuario:
                    for c in chamados_do_usuario:
                        print(f"ID: {c.id}, Título: {c.titulo}, Status:
{c.status.value}, Prioridade: {c.prioridade.value}")
```

```
                else:
                    print("Você não possui chamados.")
            elif opcao == "0":
                break
            else:
                print("Opção inválida.")



def menu_gerenciar_usuarios_acesso(repo_autenticacao: RepositorioAutenticacao):
    while True:
        print("\n--- Gerenciar Usuários de Acesso ---")
        print("1. Cadastrar Novo Usuário de Acesso")
        print("2. Listar Usuários de Acesso")
        print("0. Voltar")
        opcao = input("Escolha uma opção: ")

        if opcao == "1":
            username = input("Nome de Usuário (login): ")
            password = input("Senha: ")
            role = input("Função (admin/user): ").lower()
            cpf_or_id = None
            if role == "user":
                cpf_or_id = input("CPF associado (para usuário comum): ")

            try:
                repo_autenticacao.adicionar_usuario_acesso(username, password,
role, cpf_or_id)
                print("Usuário de acesso cadastrado com sucesso!")
            except ValueError as e:
                print(f"Erro ao cadastrar usuário de acesso: {e}")
        elif opcao == "2":
            usuarios = repo_autenticacao.obter_todos_usuarios_acesso()
            if usuarios:
                for user in usuarios:
                    print(f"Usuário: {user['username']}, Função: {user['role']},
CPF/ID: {user['cpf_or_id'] if user['cpf_or_id'] else 'N/A'}")
            else:
                print("Nenhum usuário de acesso cadastrado.")
        elif opcao == "0":
            break
        else:
            print("Opção inválida.")


def main():
    repo_usuarios = RepositorioUsuarios()
    repo_tecnicos = RepositorioTecnicos()
    repo_chamados = RepositorioChamados()
    repo_setores = RepositorioSetores()
    repo_autenticacao = RepositorioAutenticacao()

    user_role, user_identity = login(repo_autenticacao)

    if not user_role:
```

```
        print("Falha na autenticação. Encerrando o sistema.")
        return

    while True:
        print("\n--- Sistema de Chamados de Suporte de TI ---")
        if user_role == "admin":
            print("1. Gerenciar Usuários")
            print("2. Gerenciar Técnicos")
            print("3. Gerenciar Setores")
            print("4. Gerenciar Chamados (Admin)")
            print("5. Relatórios (Admin)")
            print("6. Gerenciar Usuários de Acesso")
            print("0. Sair")
            opcao = input("Escolha uma opção: ")

            if opcao == "1":
                menu_usuarios(repo_usuarios, repo_setores)
            elif opcao == "2":
                menu_tecnicos(repo_tecnicos)
            elif opcao == "3":
                menu_setores(repo_setores)
            elif opcao == "4":
                menu_chamados(repo_chamados, repo_usuarios, repo_tecnicos,
repo_setores, user_role, user_identity)
            elif opcao == "5":
                menu_relatorios(repo_chamados, repo_usuarios, repo_tecnicos,
user_role, user_identity)
            elif opcao == "6":
                menu_gerenciar_usuarios_acesso(repo_autenticacao)
            elif opcao == "0":
                print("Saindo do sistema...")
                break
            else:
                print("Opção inválida.")
        else:  # user_role == "user"
            print("1. Gerenciar Meus Chamados")
            print("2. Meus Relatórios")
            print("0. Sair")
            opcao = input("Escolha uma opção: ")

            if opcao == "1":
                menu_chamados(repo_chamados, user_role, user_identity)
            elif opcao == "2":
                menu_relatorios(repo_chamados, user_role, user_identity)
            elif opcao == "0":
                print("Saindo do sistema...")
                break
            else:
                print("Opção inválida.")


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

*** Diretório Models

```python
# chamado.py

from dataclasses import dataclass, field
from datetime import datetime
from enum import Enum
import csv
import io

class Prioridade(Enum):
    BAIXA = "baixa"
    MEDIA = "media"
    ALTA = "alta"

class StatusChamado(Enum):
    ABERTO = "aberto"
    EM_ATENDIMENTO = "em atendimento"
    RESOLVIDO = "resolvido"
    CANCELADO = "cancelado"

@dataclass
class Chamado:
    id: int
    usuario: str # cpf do usuario
    titulo: str
    descricao: str
    id_setor_local: int
    prioridade: Prioridade
    id_tecnico: int | None = None
    status: StatusChamado = StatusChamado.ABERTO
    data_abertura: datetime = field(default_factory=datetime.now)
    data_fechamento: datetime | None = None

    def __post_init__(self):
        if not isinstance(self.id, int) or self.id <= 0:
            raise ValueError(f"Id do chamado é inválido {self.id}")

        if not self.usuario or not self.usuario.strip():
            raise ValueError("Usuário está vazio.")

        if not self.titulo or not self.titulo.strip():
            raise ValueError("Título está vazio.")

        if not self.descricao or not self.descricao.strip():
            raise ValueError("Descrição está vazia.")

        if not isinstance(self.id_setor_local, int) or self.id_setor_local <= 0:
            raise ValueError(f"Id do setor é inválido {self.id_setor_local}")

    @classmethod
    def criar_de_string(cls, linha_csv: str):
        if not linha_csv or not isinstance(linha_csv, str):
            raise ValueError("Linha do arquivo CSV inválida")
        try:
```

```
            reader = csv.reader(io.StringIO(linha_csv))
            dados = next(reader)

            if len(dados) != 10:
                raise ValueError(
                    f"Quantidade de campos errada {len(dados)}, o correto são
10")

            id, usuario, id_tecnico, titulo, descricao, prioridade,
data_abertura, data_fechamento, status, id_setor_local = [item.strip() for item
in dados]

            return cls(
                id=int(id),
                usuario=usuario,
                id_tecnico=int(id_tecnico) if id_tecnico and id_tecnico != 'None'
else None,
                titulo=titulo,
                descricao=descricao,
                prioridade=Prioridade(prioridade),
                data_abertura=datetime.fromisoformat(data_abertura),
                data_fechamento=datetime.fromisoformat(data_fechamento) if
data_fechamento and data_fechamento != 'None' else None,
                status=StatusChamado(status),
                id_setor_local=int(id_setor_local)
            )

        except ValueError as e:
            raise ValueError(f"Erro ao realizar a conversão -> {e}")

    def __eq__(self, other):
        if isinstance(other, Chamado):
            return self.id == other.id
        return False

    def __hash__(self):
        return hash(self.id)
```

```
# setor.py

from dataclasses import dataclass
import csv
import io

@dataclass
class Setor:
    id: int
    nome: str
    id_gestor: int

    def __post_init__(self):
        if not isinstance(self.id, int) or self.id <= 0:
            raise ValueError(f"Id do setor é inválido {self.id}")
```

```
        if not self.nome or not self.nome.strip():
            raise ValueError("Nome está vazio.")

        if not isinstance(self.id_gestor, int) or self.id_gestor <= 0:
            raise ValueError(f"Id do gestor é inválido {self.id_gestor}")

    @classmethod
    def criar_de_string(cls, linha_csv: str):
        if not linha_csv or not isinstance(linha_csv, str):
            raise ValueError("Linha do arquivo CSV inválida")
        try:
            reader = csv.reader(io.StringIO(linha_csv))
            dados = next(reader)
            if len(dados) != 3:
                raise ValueError(
                    f"Quantidade campos errada {len(dados)}, o correto são 3")

            id, nome, id_gestor = dados
            return cls(int(id), nome, int(id_gestor))

        except ValueError as e:
            raise ValueError(f"Erro ao realizar a conversão -> {e}")

    def __eq__(self, other):
        if isinstance(other, Setor):
            return self.id == other.id
        return False

    def __hash__(self):
        return hash(self.id)
```

```
# tecnico.py

from dataclasses import dataclass
import csv
import io

@dataclass
class Tecnico:
    id: int
    nome: str
    email: str
    especialidade: str

    def __post_init__(self):
        if not isinstance(self.id, int) or self.id <= 0:
            raise ValueError(f"Id do técnico é inválido {self.id}")

        if not self.nome or not self.nome.strip():
            raise ValueError("Nome está vazio.")

        if not self.email or "@" not in self.email:
            raise ValueError('Email deve conter o símbolo @.')
```

```
        if not self.especialidade or not self.especialidade.strip():
            raise ValueError("Especialidade está vazia.")

    @classmethod
    def criar_de_string(cls, linha_csv: str):
        if not linha_csv or not isinstance(linha_csv, str):
            raise ValueError("Linha do arquivo CSV inválida")
        try:
            reader = csv.reader(io.StringIO(linha_csv))
            dados = next(reader)
            if len(dados) != 4:
                raise ValueError(
                    f"Quantidade campos errada {len(dados)}, o correto são 4")

            id, nome, email, especialidade = dados
            return cls(int(id), nome, email, especialidade)

        except ValueError as e:
            raise ValueError(f"Erro ao realizar a conversão -> {e}")

    def __eq__(self, other):
        if isinstance(other, Tecnico):
            return self.id == other.id
        return False

    def __hash__(self):
        return hash(self.id)
```

```
# usuario.py

from dataclasses import dataclass
from typing import ClassVar
import csv
import io


@dataclass
class Usuario:
    cpf: str
    nome: str
    email: str
    id_setor: int

    def __post_init__(self):
        if isinstance(self.cpf, str):
            self.cpf = self.cpf.replace(".", "").replace("-", "").strip()

        if not self.cpf.isdigit() or len(self.cpf) != 11:
            raise ValueError(
                f"Id do usuário é inválida {self.cpf}. Use o CPF como
identificador com 11 números")

        if not self.nome or not self.nome.strip():
            raise ValueError("Nome está vazio.")
```

```
        if not self.email or "@" not in self.email:
            raise ValueError('Email deve conter o símbolo @.')

    @classmethod
    def criar_de_string(cls, linha_csv: str):
        if not linha_csv or not isinstance(linha_csv, str):
            raise ValueError("Linha do arquivo CSV inválida")
        try:
            reader = csv.reader(io.StringIO(linha_csv))
            dados = next(reader)
            if len(dados) != 4:
                raise ValueError(
                    f"Quantidade campos errada {len(dados)}, o correto são 4")

            cpf, nome, email, id_setor = dados
            return cls(cpf, nome, email, int(id_setor))

        except ValueError as e:
            raise ValueError(f"Erro ao realizar a conversão -> {e}")

    def __eq__(self, other):
        if isinstance(other, Usuario):
            return self.cpf == other.cpf
        return False

    def __hash__(self):
        return hash(self.cpf)
```

*** Diretório relatório

```
# relatorios.py

from repositorio.repositorio_chamados import RepositorioChamados
from models.chamado import StatusChamado

def listar_chamados_abertos():
    repo_chamados = RepositorioChamados()
    chamados_abertos = [chamado for chamado in repo_chamados.obter_todos() if
chamado.status == StatusChamado.ABERTO]

    if not chamados_abertos:
        print("Não há chamados abertos.")
        return

    for chamado in chamados_abertos:
        print(f"ID: {chamado.id}, Título: {chamado.titulo}, Prioridade:
{chamado.prioridade.value}, Usuário: {chamado.usuario}")

def listar_chamados_por_usuario(cpf_usuario: str):
    repo_chamados = RepositorioChamados()
    chamados_usuario = [chamado for chamado in repo_chamados.obter_todos() if
chamado.usuario == cpf_usuario]
```

```
    if not chamados_usuario:
        print(f"Não há chamados para o usuário com CPF {cpf_usuario}.")
        return

    for chamado in chamados_usuario:
        print(f"ID: {chamado.id}, Título: {chamado.titulo}, Status:
{chamado.status.value}, Técnico: {chamado.id_tecnico or 'N/A'}")

def listar_chamados_por_tecnico(id_tecnico: int):
    repo_chamados = RepositorioChamados()
    chamados_tecnico = [chamado for chamado in repo_chamados.obter_todos() if
chamado.id_tecnico == id_tecnico]

    if not chamados_tecnico:
        print(f"Não há chamados atribuídos ao técnico com ID {id_tecnico}.")
        return

    for chamado in chamados_tecnico:
        print(f"ID: {chamado.id}, Título: {chamado.titulo}, Status:
{chamado.status.value}, Usuário: {chamado.usuario}")
```

*** Diretório repositorio

```
# repositorio_autenticacao.py

from pathlib import Path
import csv

from pathlib import Path
import csv

class RepositorioAutenticacao:
    def __init__(self):
        self.caminho_arquivo = Path(
            __file__).parent.parent / 'data' / 'senhas.csv'

    def _ler_usuarios_acesso(self):
        usuarios_acesso = []
        if not self.caminho_arquivo.exists():
            return []
        with open(self.caminho_arquivo, 'r', encoding='utf-8') as f:
            reader = csv.reader(f)
            for row in reader:
                if len(row) == 4:
                    usuarios_acesso.append({
                        "username": row[0],
                        "password": row[1], # In a real app, this would be hashed
                        "role": row[2],
                        "cpf_or_id": row[3] if row[3] else None
                    })
        return usuarios_acesso

    def _escrever_usuarios_acesso(self, usuarios_acesso):
        with open(self.caminho_arquivo, 'w', encoding='utf-8', newline='') as f:
```

```
            writer = csv.writer(f)
            for user in usuarios_acesso:
                writer.writerow([
                    user["username"],
                    user["password"],
                    user["role"],
                    user["cpf_or_id"] if user["cpf_or_id"] is not None else ''
                ])

    def autenticar(self, username, password):
        usuarios = self._ler_usuarios_acesso()
        for user in usuarios:
            if user["username"] == username and user["password"] == password:
                return user["role"], user["cpf_or_id"]
        return None, None # Authentication failed

    def obter_todos_usuarios_acesso(self):
        return self._ler_usuarios_acesso()

    def adicionar_usuario_acesso(self, username, password, role, cpf_or_id=None):
        usuarios = self._ler_usuarios_acesso()
        if any(u["username"] == username for u in usuarios):
            raise ValueError(f"Usuário '{username}' já existe.")
        usuarios.append({
            "username": username,
            "password": password,
            "role": role,
            "cpf_or_id": cpf_or_id
        })
        self._escrever_usuarios_acesso(usuarios)
```

```
# repositorio_chamados.py

from pathlib import Path
from models.chamado import Chamado, StatusChamado
import csv


class RepositorioChamados:
    def __init__(self):
        self.caminho_arquivo = Path(
            __file__).parent.parent / 'data' / 'chamados.csv'

    def _ler_chamados(self) -> list[Chamado]:
        chamados = []
        if not self.caminho_arquivo.exists():
            return []
        with open(self.caminho_arquivo, 'r', encoding='utf-8') as f:
            for linha in f:
                linha = linha.strip()
                if not linha:
                    continue
                try:
                    chamado = Chamado.criar_de_string(linha)
                    chamados.append(chamado)
```

```
                except ValueError as e:
                    print(f"Erro ao ler linha do chamado: {e}")
        return chamados

    def _escrever_chamados(self, chamados: list[Chamado]):
        with open(self.caminho_arquivo, 'w', encoding='utf-8', newline='') as f:
            writer = csv.writer(f)
            for chamado in chamados:
                row = [
                    str(chamado.id),
                    chamado.usuario,
                    str(chamado.id_tecnico) if chamado.id_tecnico is not None
else 'None',
                    chamado.titulo,
                    chamado.descricao,
                    chamado.prioridade.value,
                    chamado.data_abertura.isoformat(),
                    chamado.data_fechamento.isoformat() if
chamado.data_fechamento else 'None',
                    chamado.status.value,
                    str(chamado.id_setor_local)
                ]
                writer.writerow(row)

    def _gerar_proximo_id(self) -> int:
        chamados = self._ler_chamados()
        if not chamados:
            return 1
        return max(c.id for c in chamados) + 1

    def obter_todos(self) -> list[Chamado]:
        return self._ler_chamados()

    def adicionar(self, usuario: str, titulo: str, descricao: str,
id_setor_local: int, prioridade: StatusChamado):
        novo_id = self._gerar_proximo_id()
        chamado = Chamado(id=novo_id, usuario=usuario, titulo=titulo,
descricao=descricao, id_setor_local=id_setor_local, prioridade=prioridade)
        chamados = self._ler_chamados()
        chamados.append(chamado)
        self._escrever_chamados(chamados)
        return chamado

    def buscar_por_id(self, id: int) -> Chamado | None:
        for chamado in self._ler_chamados():
            if chamado.id == id:
                return chamado
        return None

    def atualizar(self, chamado_atualizado: Chamado):
        chamados = self._ler_chamados()
        chamados = [chamado if chamado.id != chamado_atualizado.id else
chamado_atualizado for chamado in chamados]
        self._escrever_chamados(chamados)
```

```python
# repositorio_setores.py

from pathlib import Path
from models.setor import Setor
import csv

class RepositorioSetores:
    def __init__(self):
        self.caminho_arquivo = Path(__file__).parent.parent / 'data' /
'setores.csv'

    def _ler_setores(self) -> list[Setor]:
        setores = []
        if not self.caminho_arquivo.exists():
            return []
        with open(self.caminho_arquivo, 'r', encoding='utf-8') as f:
            for linha in f:
                linha = linha.strip()
                if not linha:
                    continue
                try:
                    setor = Setor.criar_de_string(linha)
                    setores.append(setor)
                except ValueError as e:
                    print(f"Erro ao ler linha do setor: {e}")
        return setores

    def _escrever_setores(self, setores: list[Setor]):
        with open(self.caminho_arquivo, 'w', encoding='utf-8', newline='') as f:
            writer = csv.writer(f)
            for setor in setores:
                writer.writerow([setor.id, setor.nome, setor.id_gestor])

    def _gerar_proximo_id(self) -> int:
        setores = self._ler_setores()
        if not setores:
            return 1
        return max(s.id for s in setores) + 1

    def obter_todos(self) -> list[Setor]:
        return self._ler_setores()

    def adicionar(self, nome: str, id_gestor: int):
        novo_id = self._gerar_proximo_id()
        setor = Setor(id=novo_id, nome=nome, id_gestor=id_gestor)
        setores = self._ler_setores()
        setores.append(setor)
        self._escrever_setores(setores)
        return setor

    def buscar_por_id(self, id: int) -> Setor | None:
        todos = self.obter_todos()
        for setor in todos:
            if setor.id == id:
                return setor
```

```
        return None
```

```
# repositorio_tecnicos.py

from pathlib import Path
from models.tecnico import Tecnico
import csv

class RepositorioTecnicos:
    def __init__(self):
        self.caminho_arquivo = Path(__file__).parent.parent / 'data' /
'tecnicos.csv'

    def _ler_tecnicos(self) -> list[Tecnico]:
        tecnicos = []
        if not self.caminho_arquivo.exists():
            return []
        with open(self.caminho_arquivo, 'r', encoding='utf-8') as f:
            for linha in f:
                linha = linha.strip()
                if not linha:
                    continue
                try:
                    tecnico = Tecnico.criar_de_string(linha)
                    tecnicos.append(tecnico)
                except ValueError as e:
                    print(f"Erro ao ler linha do tecnico: {e}")
        return tecnicos

    def _escrever_tecnicos(self, tecnicos: list[Tecnico]):
        with open(self.caminho_arquivo, 'w', encoding='utf-8', newline='') as f:
            writer = csv.writer(f)
            for tecnico in tecnicos:
                writer.writerow([tecnico.id, tecnico.nome, tecnico.email,
tecnico.especialidade])

    def _gerar_proximo_id(self) -> int:
        tecnicos = self._ler_tecnicos()
        if not tecnicos:
            return 1
        return max(t.id for t in tecnicos) + 1

    def obter_todos(self) -> list[Tecnico]:
        return self._ler_tecnicos()

    def adicionar(self, nome: str, email: str, especialidade: str):
        novo_id = self._gerar_proximo_id()
        tecnico = Tecnico(id=novo_id, nome=nome, email=email,
especialidade=especialidade)
        tecnicos = self._ler_tecnicos()
        tecnicos.append(tecnico)
        self._escrever_tecnicos(tecnicos)
        return tecnico

    def buscar_por_id(self, id: int) -> Tecnico | None:
```

```
        todos = self.obter_todos()
        for tecnico in todos:
            if tecnico.id == id:
                return tecnico
        return None
```

```
# repositorio_usuarios.py

from pathlib import Path
from models.usuario import Usuario
import csv

class RepositorioUsuarios:
    def __init__(self):
        self.caminho_arquivo = Path(__file__).parent.parent / 'data' /
'usuarios.csv'

    def _ler_usuarios(self) -> list[Usuario]:
        usuarios = []
        if not self.caminho_arquivo.exists():
            return []
        with open(self.caminho_arquivo, 'r', encoding='utf-8') as f:
            for linha in f:
                linha = linha.strip()
                if not linha:
                    continue
                try:
                    usuario = Usuario.criar_de_string(linha)
                    usuarios.append(usuario)
                except ValueError as e:
                    print(f"Erro ao ler linha do usuario: {e}")
        return usuarios

    def _escrever_usuarios(self, usuarios: list[Usuario]):
        with open(self.caminho_arquivo, 'w', encoding='utf-8', newline='') as f:
            writer = csv.writer(f)
            for usuario in usuarios:
                writer.writerow([usuario.cpf, usuario.nome, usuario.email,
usuario.id_setor])

    def obter_todos(self) -> list[Usuario]:
        return self._ler_usuarios()

    def adicionar(self, usuario: Usuario):
        usuarios = self._ler_usuarios()
        usuarios.append(usuario)
        self._escrever_usuarios(usuarios)

    def buscar_por_cpf(self, cpf: str) -> Usuario | None:
        todos = self.obter_todos()
        cpf_limpo = cpf.replace(".", "").replace("-", "").strip()

        for usuario in todos:
            if usuario.cpf == cpf_limpo:
                return usuario
```

```
        return None
```

Os dados foram criados no diretório data como demonstrado no início do pdf, foi utilizado o sistema, em cada diretório existe um arquivo init.py vazio para referenciação das bibliotecas pelo python.