UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS DE CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

LUIZ GUSTAVO PIUCO BAZZOTTI: 20230001490
WENDELL NERIS: 2311100035
BRUNO VENDRUSCOLO: 2221100004

TRABALHO INTEGRADOR

GEX613 - PROGRAMAÇÃO II - T02 (2025.1)

CHAPECÓ
13 de julho de 2025

Estrutura geral

O projeto foi dividido em quatro pastas principais: backend, frontend, scripts e documentos Essa separação visa organizar o projeto, facilitando o entendimento do sistema e entendimento dos arquivos.

/backend: Contém a API e a lógica de negócio utilizando FastAPI e SQLAlchemy para integração com o banco de dados. Cada entidade possui um módulo de rotas dedicado.

Rotas: As rotas são incluídas no app principal via include_router.

```python
app.include_router(usuario.router, tags=["Usuario"])
```

Crud: Separação clara entre lógica de acesso a dados (camada crud) e regras de negócio, promovendo reutilização.

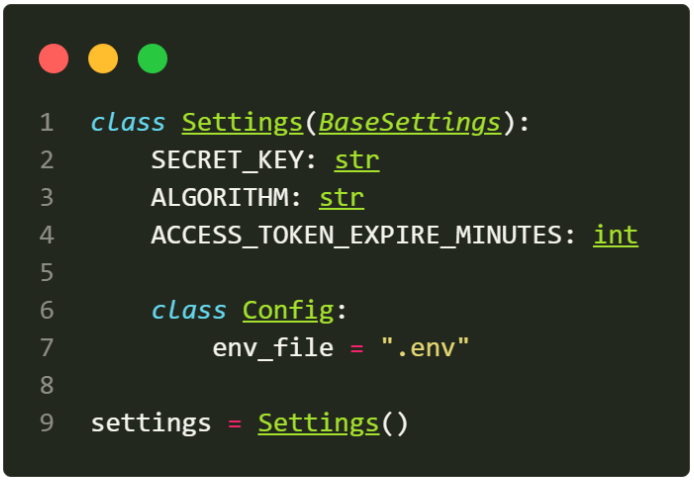
Configuração e Segurança: Centralização das configurações em config.py e implementação de autenticação básica em security.py.

```python
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

def hash_password(password: str) -> str:
    """Criptografa a senha em texto puro."""
    return pwd_context.hash(password)

def verify_password(plain_password: str, hashed_password: str) -> bool:
    """Verifica se a senha em texto puro corresponde à senha criptografada."""
    return pwd_context.verify(plain_password, hashed_password)

def create_access_token(data: dict):
    to_encode = data.copy()
    expire = datetime.utcnow() + timedelta(minutes=settings.ACCESS_TOKEN_EXPIRE_MINUTES)
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, settings.SECRET_KEY, algorithm=settings.ALGORITHM)
    return encoded_jwt
```



```python
class Settings(BaseSettings):
    SECRET_KEY: str
    ALGORITHM: str
    ACCESS_TOKEN_EXPIRE_MINUTES: int

    class Config:
        env_file = ".env"

settings = Settings()
```

Decidimos por utilizar o FastAPI por recomendação de um amigo e por ser um framework fácil de aprender com fácil integração ao frontend

/frontend: Feito em Typescript e React, com Tailwind para estilização, focando em uma interface responsiva.

Hooks e Componentização: Uso de hooks personalizados, para gerenciar estados globais (por exemplo: notificações), promovendo modularidade.

Configuração: Arquivos de configuração separados para ESLint, Tailwind e Typescrpit, garantindo padronização.

Decidimos utilizar Tailwind e Typescript também por recomendação, para a estilização das páginas ficar mais fácil.

Scripts: A pasta scripts contém três arquivos principais que automatizam o gerenciamento do banco de dados do projeto, Create: responsável pela criação do banco de dados e de todas as tabelas necessárias; Insert: utilizado para popular o banco de dados com dados iniciais de exemplo; Select: usado para uma consulta rápida dos dados.

Documentos: Contém a modelagem do banco e requisitos, garantindo rastreabilidade e alinhamento com as necessidades do negócio. Toda a modelagem e requisitos estão documentados.

Facilidades/Dificuldades

Entre as principais dificuldades, podemos destacar a implementação do login com autenticação e autorização. A configuração do fluxo de autenticação e a definição de permissões para diferentes tipos de usuários. Além disso, tivemos problemas com a integração entre o frontend e o backend, especialmente no tratamento de erros.
O uso de Hooks no frontend, por exemplo, trouxe muita praticidade. Como são padronizados no React, depois que criamos os primeiros hooks para lidar com chamadas de API e gerenciamento de estado, ficou muito mais simples replicar o padrão em outras partes da aplicação. Da mesma forma, a configuração de rotas seguiu um padrão bem definido. Após implementar a estrutura inicial com get, post, put e delete as novas rotas foram fáceis de adaptar.