Projeto 1 - Python Puro

Acesse diretamente pelo Notion!

Copie os códigos facilmente pelo link abaixo.

```
https://grizzly-amaranthus-f6a.notion.site/Projeto-1-Python-Puro-15b6cf8ea89f80c9a3e9d2882eed45e6?pvs=4
```

▼ Configurações iniciais

Primeiro devemos criar o ambiente virtual:

```
# Criar
    # Linux
        python3 -m venv venv
    # Windows
        python -m venv venv
```

Após a criação do venv vamos ativa-lo:

```
#Ativar
    # Linux
        source venv/bin/activate
    # Windows
        venv\Scripts\Activate

# Caso algum comando retorne um erro de permissão execute o código e tente novamente:

Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned
```

Faça a instalaçao do SQLModel:

```
pip install sqlmodel
```

▼ Banco de dados

Vamos criar as tabelas no Banco de dados, em models/model.py:

```
from sqlmodel import Field, SQLModel, create_engine, Relationship
from typing import Optional
from datetime import date, datetime
from decimal import Decimal

class Subscription(SQLModel, table=True):
    id: int = Field(primary_key=True)
    empresa: str
    site: Optional[str] = None
    data_assinatura: date
    valor: Decimal

class Payments(SQLModel, table=True):
    id: int = Field(primary_key=True)
    subscription_id: int = Field(foreign_key="subscription.id")
    subscription: Subscription = Relationship()
    date: date
```

Para efetivar as mudanças no DB, em models/database.py crie:

```
from sqlmodel import SQLModel, create_engine
from .model import *

sqlite_file_name = "database.db"
sqlite_url = f"sqlite:///{sqlite_file_name}"

engine = create_engine(sqlite_url, echo=False)
```

```
def create_db_and_tables():
    SQLModel.metadata.create_all(engine)


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

Efetue a migração com:

```
python3 models/database.py
```

▼ views

Aqui nas views vamos trabalhar com a lógica da nossa aplicação.

Em `__init__.py` permita importações:

```
import sys
import os

sys.path.append(os.path.abspath(os.curdir))
```

Vamos criar uma classe para processar as assinaturas:

```
class SubscriptionService:
    def __init__(self, engine):
        self.engine = engine
```

Nela adicione o método para salvar assinaturas no Banco:

```
from sqlmodel import Session
from models.model import Subscription

def create(self, subscription: Subscription):
    with Session(self.engine) as session:
        session.add(subscription)
        session.commit()
        session.refresh(subscription)
        return subscription
```

Agora, vamos trabalhar com a listagem de todos os dados salvos no BD:

```
def list_all(self):
    with Session(self.engine) as session:
        statement = select(Subscription)
        results = session.exec(statement).all()
    return results
```

No próximo método, vamos receber uma assinatura e marcar o seu pagamento no mês atual:

```
def _has_pay(self, results):
    for result in results:
        if result.date.month == date.today().month:
            return True
    return False

def pay(self, subscription: Subscription):
    with Session(self.engine) as session:
        statement = select(Payments).join(Subscription).where(Subscription.empresa==subscription.empresa)
        results = session.exec(statement).all()

        if self._has_pay(results):
            question = input('Essa conta já foi paga esse mês, deseja pagar novamente ? Y ou N')
```

```
            if not question.upper() == 'Y':
                return

        pay = Payments(subscription_id=subscription.id)
        session.add(pay)
        session.commit()
```

Nessa etapa, desejamos calcular qual o custo mensal de todas as assinaturas ativas do usuário:

```
def total_value(self):
    with Session(self.engine) as session:
        statement = select(Subscription)
        results = session.exec(statement).all()

    total = 0
    for result in results:
        total += result.valor

    return float(total)
```

Finalmente chegamos aos gráficos, aqui criaremos 3 métodos.

- Criar uma lista dos últimos 12 meses
- Criar uma lista dos gastos no últimos 12 meses
- gerar o gráticos

```
def _get_last_12_months_native(self):
    today = datetime.now()
    year = today.year
    month = today.month
    last_12_months = []
    for _ in range(12):
        last_12_months.append((month, year))
        month -= 1
        if month == 0:
            month = 12
            year -= 1
    return last_12_months[::-1]

def _get_values_for_months(self, last_12_months):
    with Session(self.engine) as session:
        statement = select(Payments)
        results = session.exec(statement).all()

        value_for_months = []
        for i in last_12_months:
            value = 0
            for result in results:
                if result.date.month == i[0] and result.date.year == i[1]:
                    value += float(result.subscription.valor)

            value_for_months.append(value)
    return value_for_months

def gen_chart(self):
    last_12_months = self._get_last_12_months_native()
    values_for_months = self._get_values_for_months(last_12_months)
    last_12_months = list(map(lambda x: x[0], self._get_last_12_months_native()))

    import matplotlib.pyplot as plt
    plt.plot(last_12_months, values_for_months)
    plt.show()
```

▼ Templates

Nessa etapa criaremos a ‘Interface’ de nossa aplicação.

Em templates/app.py crie a classe UI:

```
import __init__
from views.view import SubscriptionService
from models.database import engine

class UI:
    def __init__(self):
        self.subscription_service = SubscriptionService(engine)
```

Crie os métodos para realizar as ações.

- Adicionar assinatura
- Remover assinatura
- Valor total
- Gastos últimos 12 meses
- Sair

```
def add_subscription(self):
    empresa = input('Empresa: ')
    site = input('Site: ')
    data_assinatura = datetime.strptime(input('Data de assinatura: '), '%d/%m/%Y')
    valor = Decimal(input('Valor: '))

    subscription = Subscription(empresa=empresa, site=site, data_assinatura=data_assinatura, valor=valor)
    self.subscription_service.create(subscription)
    print('Assinatura adicionada com sucesso.')

def delete_subscription(self):
    subscriptions = self.subscription_service.list_all()
    print('Escolha qual assinatura deseja excluir')

    for i in subscriptions:
        print(f'[{i.id}] -> {i.empresa}')

    choice = int(input('Escolha a assinatura: '))
    self.subscription_service.delete(choice)
    print('Assinatura excluída com sucesso.')

def total_value(self):
    print(f'Seu valor total mensal em assinaturas: {self.subscription_service.total_value()}')
```

Para finalizar crie o método start:

```
def start(self):
    while True:
        print('''
        [1] -> Adicionar assinatura
        [2] -> Remover assinatura
        [3] -> Valor total
        [4] -> Gastos últimos 12 meses
        [5] -> Sair
        ''')

        choice = int(input('Escolha uma opção: '))

        if choice == 1:
            self.add_subscription()
        elif choice == 2:
            self.delete_subscription()
        elif choice == 3:
            self.total_value()
        elif choice == 4:
            self.subscription_service.gen_chart()
        else:
            break
```

E o chame no IF:

```
if __name__ == '__main__':
    UI().start()
```