Io inizierei da USDK.sol
È il contratto **meno dipendente dagli altri**, ma da cui quasi tutti gli altri *dipendono*.

🔷 Perché partire da USDK.sol?

USDK è “banalmente”:

- un **ERC20** con:
    - mint
    - burn
    - ruoli (minter/burner: vault, pool, eventualmente treasury)
- niente logica di vault, niente logica di leverage, niente farming.

È il pezzo:

- più semplice da implementare,
- più facile da testare (unit test puri),
- super riutilizzabile anche mentre il resto è ancora in bozza,
- richiesto da:
    - Vault USDT (che paga interessi in USDK),
    - Vault WBTC/XAUT/SPYON (che pagano in USDK),
    - Pool USDK/USDT,
    - PointsManager (se vuole leggere saldo USDK),
    - eventuali future integrazioni esterne (CEX/DEX, prestiti, ecc).

In pratica: **nessuno dei contratti di base è necessario per far funzionare USDK**,
ma quasi tutti gli altri devono interfacciarsi con USDK.

🔷 Dipendenza minima: grafo mentale dei contratti

Se guardiamo il “grafo di dipendenze”:

- **USDK.sol**
    - NON dipende da: vault, strategy, pool, treasury, points, risk manager
    - Solo da librerie standard (OpenZeppelin: ERC20, AccessControl, ecc.)
- **Treasury.sol**
    - può esistere anche solo come wallet/multisig all’inizio
    - dipende solo da: USDK (eventuale read), USDT (ERC20)
- **BaseVault.sol**
    - dipende da:
        - asset depositato (USDT, WBTC, ecc)
        - token di share interno (può essere un ERC20)
        - StrategyContracts (interfaccia)
- **StrategyStableFarm.sol**
    - dipende da:
        - BaseVault interface
        - protocolli esterni (Curve, Aave, Pendle, ecc)
        - USDK (per sapere cosa pagare)
- **StrategyLeveragedLendFarm.sol**
    - dipende da:
        - Aave
        - farm esterna
        - BaseVault

        - RiskManager

- **PoolManager.sol**
    - dipende da:
        - USDK
        - USDT
        - AMM (Uniswap, Curve, ecc.)
- **PointsManager.sol**
    - dipende da:
        - Vault (per sapere depositi / tempo)
        - magari USDK/KTG per conversione/claim
- **RiskManager.sol**
    - dipende da:
        - StrategyLeveraged
        - oracoli
        - parametri di vault

👉 L'unico veramente "isolato" è **USDK.sol**.

🔷 Ordine consigliato di sviluppo

Se dovessi impostare il progetto da zero, farei così:

1️⃣ USDK.sol

- ERC20 base
- mint/burn con AccessControl (ROLE_MINTER, ROLE_BURNER)
- pause/emergency (opzionale)

- eventi chiari (Minted, Burned, MinterAdded, ecc.)

Test:

- mint/burn corretti
- gestione ruoli
- nessuna dipendenza esterna

2️⃣ BaseVault.sol + MockStrategy

Prima di implementare la vera farm:

- scrivi un `BaseVault` generico per asset ERC20
- scrivi una **MockStrategy** semplicissima che:
    - tiene solo conto di “profitto finto”
    - ti permette di testare: deposit, withdraw, harvest, distribuzione

Così validi:

- la logica di shares
- la logica di calcolo NAV
- l’API del vault

3️⃣ StrategyStableFarm.sol (reale)

Una volta che il vault è solido con la mock:

- integri una **prima vera farm** (es. su un protocollo di test, magari una sola pool su una testnet)
- qui inizi a interfacciarti con protocolli esterni (Curve, Pendle, ecc.)

4️⃣ PoolManager.sol

Quando hai:

- USDK
- un vault che produce USDK per l’utente
- una strategia che genera USDT

allora ha senso aggiungere la **pool USDK/USDT** + manager che:

- spinge USDT in LP
- gestisce burn/mint equilibrando supply

5️⃣ PointsManager + KTG (in un secondo momento)

Una volta che:

- i vault girano
- USDK si muove
- la pool funziona

aggiungi:

- contatore punti per KTG
- logica di vesting
- emissione KTG vera e propria

6️⃣ StrategyLeveragedLendFarm + RiskManager (ultimo step)

Sono i più complessi e rischiosi:

- integrazione Aave (o simili)
- gestione LTV, HF, liquidation threshold
- logica di deleverage automatica
- interaction col RiskManager

Da fare solo quando:

- il core (USDK + vault + pool) è **ultra stabile e testato**.

🔷 TL;DR

👉 **Contratto da cui partire: `USDK.sol`**
Perché:

- è **il meno dipendente** dagli altri
- è la base monetaria del sistema
- lo puoi scrivere, testare e auditare **subito**
- mentre gli altri componenti sono ancora in fase di design.

Subito dopo: `BaseVault.sol` + `MockStrategy.sol`.
