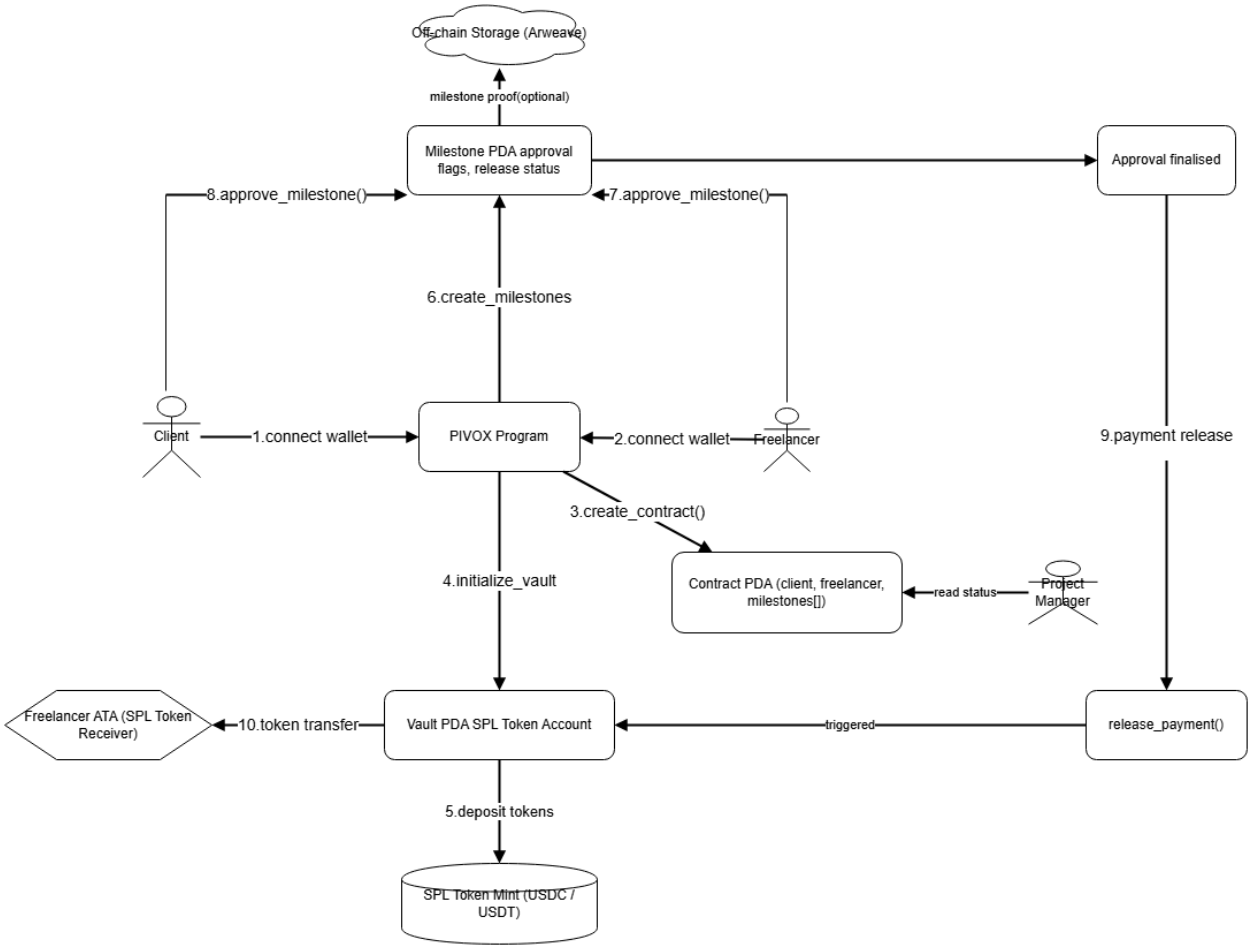
PIVOX REQUIREMENTS

- The protocol shall allow a **client** to initiate contract creation via a signed wallet transaction.
- The protocol shall allow a **freelancer** to join a contract and view milestones using their wallet.
- The protocol shall initialize a **Contract PDA**, storing the client, freelancer, and associated milestones.
- The protocol shall initialize a **Vault PDA** to hold client-deposited tokens for milestone-based payouts.
- The protocol shall allow deposits only from a **whitelisted list of SPL tokens** (e.g., USDC / USDT).
- The protocol shall derive a **Milestone PDA** per contract to track approval flags and release status.
- The protocol shall enable a **project manager** to view contract and milestone status in **read-only mode**.

Overview



- Freelancers and clients coordinate off-chain to define milestones and payment terms.
- Wallet-based approval and signing is required from both participants using Phantom.
- After dual approval:
    - A **Contract PDA** is initialized using seeds: `[b"contract", client_pubkey, freelancer_pubkey, contract_id]`.
    - This PDA stores: `client`, `freelancer`, `milestones[]`, `released[]`, `status`, etc.
- Additional derived accounts:
    - **Vault PDA**: stores locked USDC/USDT
    - **Milestone PDA(s)**: track per-milestone approval status
- Once on-chain setup completes, the protocol transitions to milestone approval and execution flow.

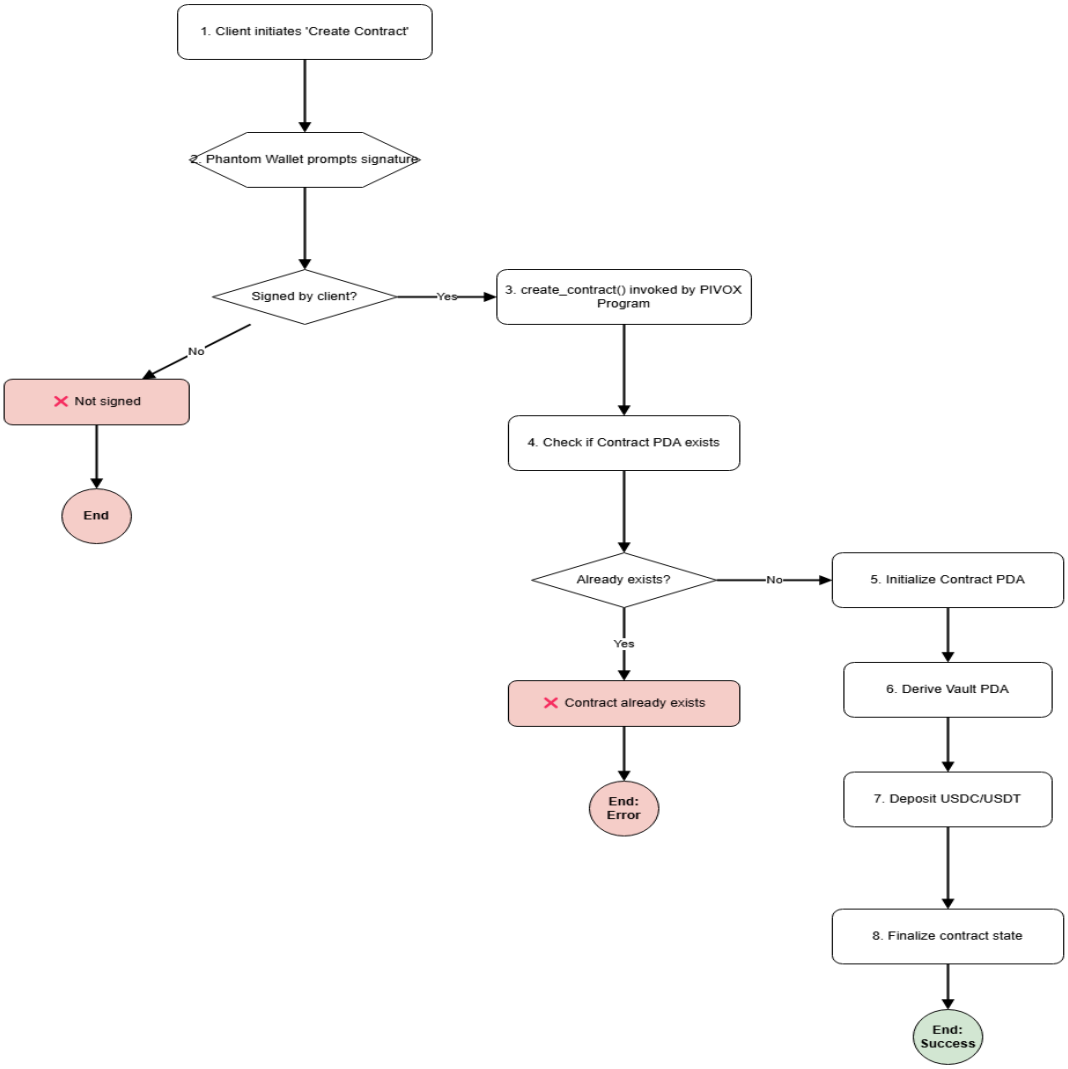
Core Accounts

```
#[account]
pub struct Contract {
    client: Pubkey,                 // Client wallet address
    freelancer: Pubkey,             // Freelancer wallet address
    pm: Option<Pubkey>,             // Project Manager (read-only observer)
    token_mint: Pubkey,             // Mint used for payments (e.g., USDC)
    amount_agreed_on: u64,          // Total contract value
    amount_released: u64,           // Total paid so far
    milestone_count: u8,            // Number of milestones
    status: u8,                     // 0: Draft, 1: Active, 2: Completed, 3: Cancelled
    start_ts: i64,                  // Contract creation time
    end_ts: Option<i64>,            // Optional termination time
    bump: u8,                       // PDA bump
}
```

```
#[account]
pub struct Vault {
    authority: Pubkey,              // Contract PDA or program authority
    token_account: Pubkey,          // Actual SPL Token Account (ATA)
    token_mint: Pubkey,             // USDC / USDT
    total_deposited: u64,           // Total tokens received
    total_withdrawn: u64,           // Total tokens sent to freelancer
    bump: u8,
}
```

```
#[account]
pub struct Milestone {
    contract: Pubkey,               // Parent Contract PDA
    milestone_id: u64,              // Unique per contract
    title: [u8; 64],                // Short description (fixed length)
    amount: u64,                    // Payment amount
    freelancer_approved: bool,
    client_approved: bool,
    is_released: bool,
    bump: u8,                       // PDA bump
}
```

Contract Creation Flow

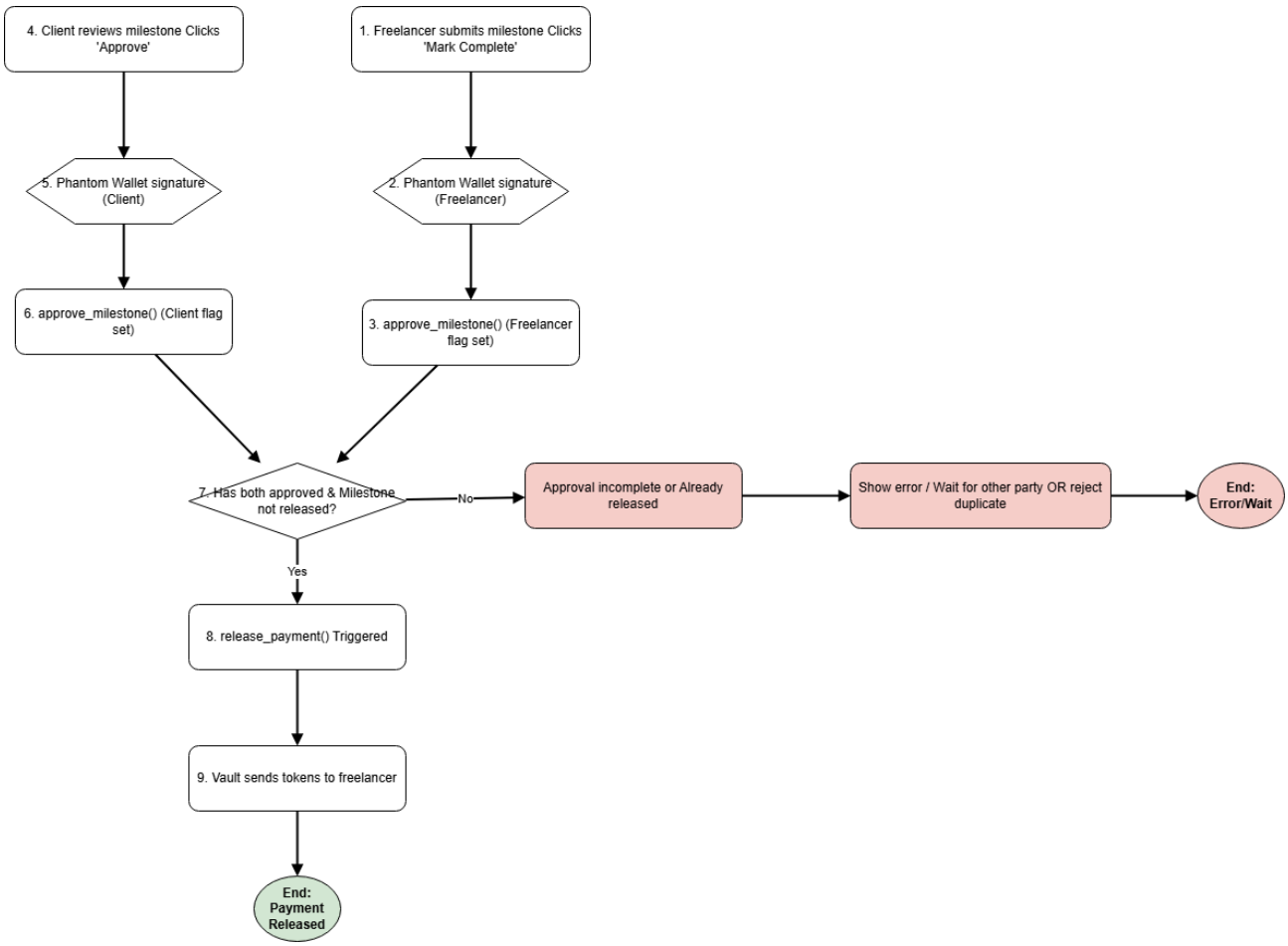


- The client initiates contract creation via a signed wallet transaction.
- The program:
    - Checks for duplicate contracts via PDA derivation.
    - Validates client signature before proceeding.
- Upon success:
    - **Contract PDA** is created and initialized with milestone metadata.
    - **Vault PDA** is derived and initialized as a program-owned SPL token account.
- Client deposits tokens (e.g., USDC/USDT) into Vault PDA using `spl_token::transfer`.
- If any of the validations fail (e.g., wrong signer, unsupported mint), the instruction is aborted.

Vault Logic

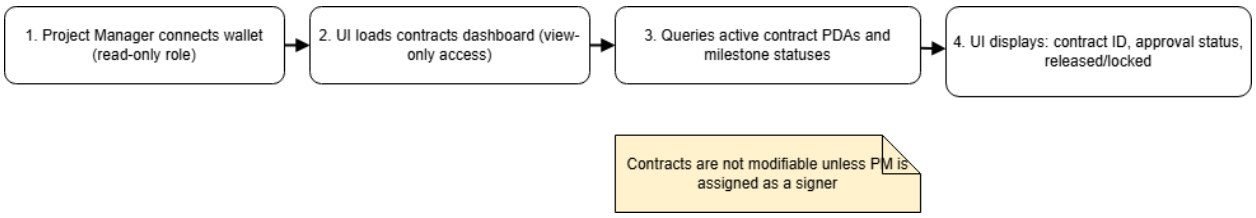
- Token movement:
  - `spl_token::transfer_checked()` is used for secure payout
  - Destination is the freelancer's wallet-linked ATA
- Vault only releases tokens when both milestone approvals are complete and vault ownership is verified.
- Failure cases include:
  - Invalid token mint
  - Mismatch in authority or destination address
  - Re-entry attempt

Milestone Approval Flow



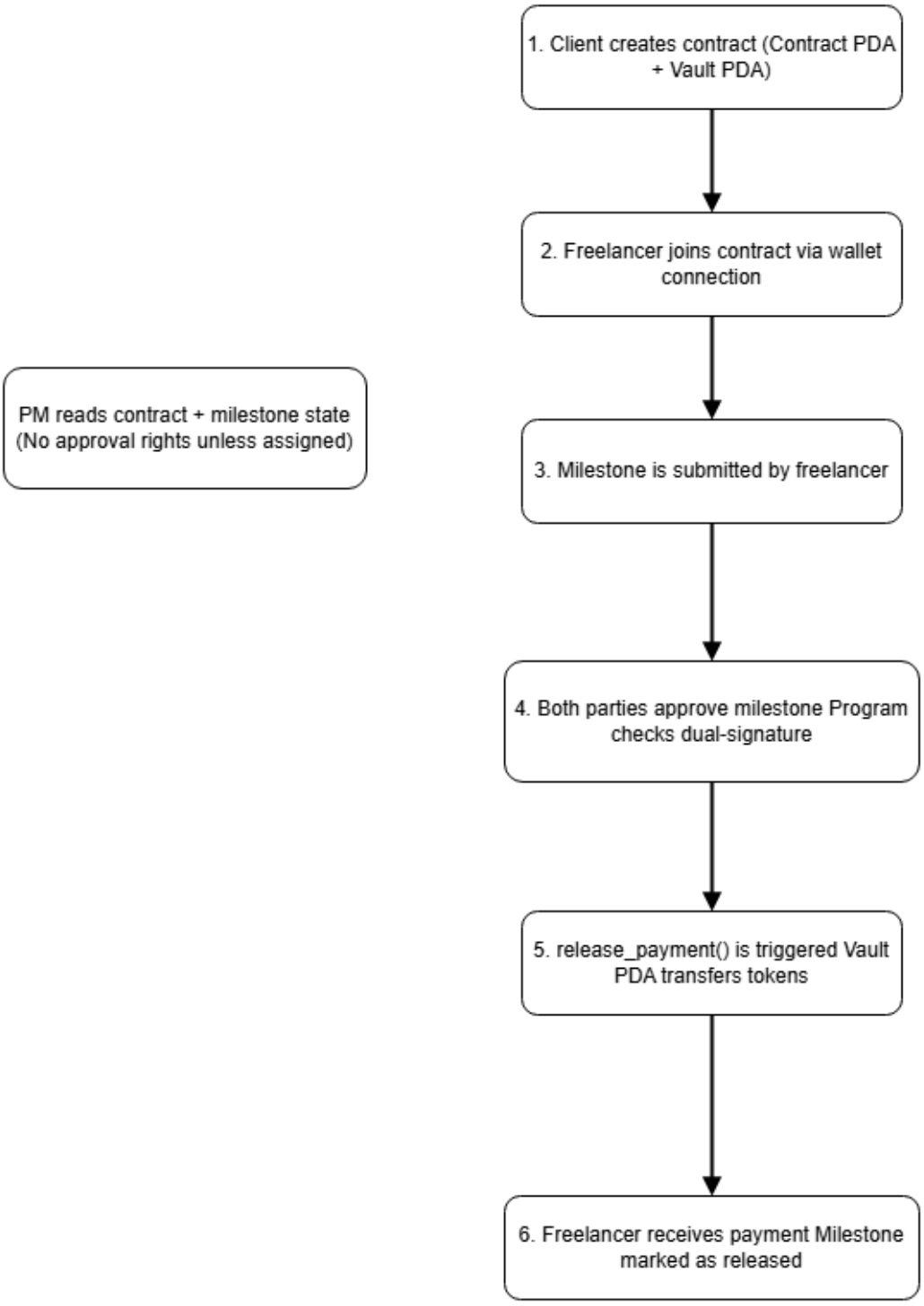
- Freelancer marks a milestone as completed using `approve_milestone()` from their wallet.
- The same instruction is later executed by the client to complete dual approval.
- **Milestone PDA** is used to track:
  - `freelancer_approved`, `client_approved`, `is_released`
  - Derived as `[b"milestone", contract_pda, milestone_id]`
- Once both parties have approved:
  - Program triggers `release_payment()`
  - Tokens are transferred from **Vault PDA** to freelancer's **Associated Token Account (ATA)**
- If already released or partially approved, the transaction is blocked to prevent double spend.

PM Dashboard View



- PM wallet connects via frontend with **read-only permissions**.
- Contracts and milestones are indexed using `getProgramAccounts()` or Anchor's `Program.fetchAll()`.
- The dashboard shows:
    - Active contracts
    - Milestone states: `Pending`, `Approved`, `Released`
    - Signer status for both client and freelancer
- PM cannot approve or release unless explicitly listed in the contract metadata (`pm: Option<Pubkey>`).
- All read operations are executed via deserialized PDA state from the blockchain.

Full User Flow Summary



- Entire PIVOX lifecycle:
    - Client creates contract → Contract + Vault PDA initialized.
    - Freelancer joins, milestones tracked.
    - Both parties approve → triggers release_payment().
    - Funds transferred from Vault PDA to freelancer.
    - Project manager optionally monitors flow via read-only dashboard.
- All operations are performed via Anchor instructions and follow strict PDA derivation constraints.
- State transitions are enforced:
    - ContractState: Draft → Confirmed → Completed
    - MilestoneState: Pending → Approved → Released