SMART CONTRACT	https://github.com/blasmorkai/cw-plus-bm/tree/main/contracts/cw4-stake

Scope: state.rs, message.rs, contract.rs (instantiate entry point)

CODE REVIEW

state.rs

```
// A Claim allows a given address to claim an amount of tokens after a release date.
// When a claim is created: an address, amount and expiration are given.
// Claims(Map<&Addr, Vec<Claim>>)      struct Claim {amount: Uint128,release_at:
Expiration,}
pub const CLAIMS: Claims = Claims::new("claims");

// Duration is a delta of time.
#[derive(Serialize, Deserialize, Clone, PartialEq, JsonSchema, Debug)]
pub struct Config {
   /// denom of the token to stake
   pub denom: Denom,      //enum Denom {Native(String), Cw20(Addr),}
                          // We can specify a String (coin denom) or an addr
(contract address) for the Denomination
   pub tokens_per_weight: Uint128,     // Constant, will not change as we stake/bond
new tokens.
   pub min_bond: Uint128,
   pub unbonding_period: Duration,
}

// ADMIN: Item< Option<Addr> >      struct Admin(Item<Option<Addr>>)
pub const ADMIN: Admin = Admin::new("admin");

// HOOKS: Item< Vec<Addr> >         struct Hooks(Item< Vec<Addr> >)
pub const HOOKS: Hooks = Hooks::new("cw4-hooks");

pub const CONFIG: Item<Config> = Item::new("config");

// TOTAL gets updated when tokens are bonded or unbonded. Claiming tokens has no
effect over total.
pub const TOTAL: Item<u64> = Item::new(TOTAL_KEY);

// SnapshotMap => Map that maintains a snapshots of one or more checkpoints.
// We can query historical data as well as current state. What data is snapshotted
depends on the Strategy.
// The Stragegy can be EveryBlock, Never or Selected
// MEMBERS is a ledger where we register weights at a certain height, or remove the
address entry for a certain height.
```

```rust
pub const MEMBERS: SnapshotMap<&Addr, u64> = SnapshotMap::new(
    cw4::MEMBERS_KEY,
    cw4::MEMBERS_CHECKPOINTS,
    cw4::MEMBERS_CHANGELOG,
    Strategy::EveryBlock,
);

// Bonded tokens per address
pub const STAKE: Map<&Addr, Uint128> = Map::new("stake");
```

msg.rs

```rust
pub struct InstantiateMsg {
    /// denom of the token to stake
    pub denom: Denom,
    pub tokens_per_weight: Uint128,
    pub min_bond: Uint128,
    pub unbonding_period: Duration,

    // admin can only add/remove hooks, not change other parameters
    pub admin: Option<String>,
}

#[derive(Serialize, Deserialize, Clone, PartialEq, JsonSchema, Debug)]
#[serde(rename_all = "snake_case")]
pub enum ExecuteMsg {
    /// Bond will bond all staking tokens sent with the message and update membership
weight
    Bond {},
    /// Unbond will start the unbonding process for the given number of tokens.
    /// The sender immediately loses weight from these tokens, and can claim them
    /// back to his wallet after `unbonding_period`
    Unbond { tokens: Uint128 },
    /// Claim is used to claim your native tokens that you previously "unbonded"
    /// after the contract-defined waiting period (eg. 1 week)
    Claim {},

    /// Change the admin
    UpdateAdmin { admin: Option<String> },
    /// Add a new hook to be informed of all membership changes. Must be called by
Admin
    AddHook { addr: String },
    /// Remove a hook. Must be called by Admin
    RemoveHook { addr: String },
```

```rust
    /// This accepts a properly-encoded ReceiveMsg from a cw20 contract
    Receive(Cw20ReceiveMsg),
}

#[derive(Serialize, Deserialize, Clone, Debug, PartialEq, JsonSchema)]
#[serde(rename_all = "snake_case")]
pub enum ReceiveMsg {
    /// Only valid cw20 message is to bond the tokens
    Bond {},
}

#[derive(Serialize, Deserialize, Clone, PartialEq, JsonSchema, Debug)]
#[serde(rename_all = "snake_case")]
pub enum QueryMsg {
    /// Claims shows the tokens in process of unbonding for this address
    //  struct ClaimsResponse { pub claims: Vec<Claim>,}
    Claims {
        address: String,
    },
    // Show the number of tokens currently staked by this address.
    // struct StakedResponse { pub stake: Uint128, pub denom: Denom,}
    Staked {
        address: String,
    },

    /// Return AdminResponse            struct AdminResponse {pub admin:
Option<String>,}
    Admin {},
    /// Return TotalWeightResponse      struct TotalWeightResponse {weight: u64,}
    TotalWeight {},
    /// Returns MembersListResponse     struct MemberListResponse {pub members:
Vec<Member>,}
    ListMembers {
        start_after: Option<String>,
        limit: Option<u32>,
    },
    /// Returns MemberResponse          struct MemberResponse { pub weight:
Option<u64>,}
    Member {
        addr: String,
        at_height: Option<u64>,
    },
    /// Shows all registered hooks. Returns HooksResponse.  struct HooksResponse { pub
hooks: Vec<String>,}
    Hooks {},
}
```

```rust
#[derive(Serialize, Deserialize, Clone, Debug, PartialEq, JsonSchema)]
pub struct StakedResponse {
    pub stake: Uint128,
    pub denom: Denom,
}
```

contract.rs

```rust
/ Note, you can use StdResult in some functions where you do not
// make use of the custom errors
#[cfg_attr(not(feature = "library"), entry_point)]
pub fn instantiate(
    mut deps: DepsMut,
    _env: Env,
    _info: MessageInfo,
    msg: InstantiateMsg,
) -> Result<Response, ContractError> {
    set_contract_version(deps.storage, CONTRACT_NAME, CONTRACT_VERSION)?;
    let api = deps.api;
    // Admin struct helper function => set = save
    // Validation of the admin address if it is Some.
    ADMIN.set(deps.branch(), maybe_addr(api, msg.admin)?)?;

    // min_bond is at least 1, so 0 stake -> non-membership
    let min_bond = std::cmp::max(msg.min_bond, Uint128::new(1));

    let config = Config {
        denom: msg.denom,
        tokens_per_weight: msg.tokens_per_weight,
        min_bond,
        unbonding_period: msg.unbonding_period,
    };
    CONFIG.save(deps.storage, &config)?;

    // So far, no tokens have been staked
    TOTAL.save(deps.storage, &0)?;

    Ok(Response::default())
}
```

GENERAL QUESTIONS

1. **What are the concepts (borrowing, ownership, vectors etc)**
   The Concepts in the code are Structs, Item, Map, Vectors, Tupples, Ownerships, Enum (Options, Results), Coins.
   When setting up the state, a few structures like Admin, Hooks, Claims are based on common Items and Maps.

2. **What is the organization?**
   Entry points and the functions taking care of its messages.

3. **What is the contract doing? What is the mechanism?**
   On instantiation, the token info is set up.
   The parameter Denom can either be set up as a token denomination or as an address. The first one will deal with the staking of funds sent by senders via info.funds. The latter will stake amounts sent by a contract (on behalf of a user) via Cw20ReceiveMsg.

4. **How could it be better? More efficient? Safer?**
   Nothing that I can think of.