SMART CONTRACT	https://github.com/ivmidable/messages

Comments in blue.
Code Changes in green.

CODE REVIEW

State.rs

```
use schemars::JsonSchema;
use serde::{Deserialize, Serialize};

use cosmwasm_std::{Addr, Uint128};
use cw_storage_plus::{Item, Map};

#[derive(Serialize, Deserialize, Clone, Debug, PartialEq, JsonSchema)]
pub struct Message {
    pub id:Uint128,
    pub owner:Addr,
    pub topic: String,
    pub message: String
}

// CURRENT_ID is an increasing counter per Message struct stored on the MESSAGES Map
pub const CURRENT_ID: Item<u128> = Item::new("current_id");

//Map has a u128 key and Message value
pub const MESSAGES: Map<u128, Message> = Map::new("messages");
```

Msg.rs

```
#[derive(Serialize, Deserialize, Clone, Debug, PartialEq, JsonSchema)]
pub struct InstantiateMsg {
}

#[derive(Serialize, Deserialize, Clone, Debug, PartialEq, JsonSchema)]
#[serde(rename_all = "snake_case")]
pub enum ExecuteMsg {
    AddMessage {topic:String, message:String},
}

#[derive(Serialize, Deserialize, Clone, Debug, PartialEq, JsonSchema)]
#[serde(rename_all = "snake_case")]
pub enum QueryMsg {
    GetCurrentId {},
    GetAllMessage {},
```

```
    GetMessagesByAddr { address:String },
    GetMessagesByTopic { topic:String },
    GetMessagesById { id:Uint128 },
}

#[derive(Serialize, Deserialize, Clone, Debug, PartialEq, JsonSchema)]
#[serde(rename_all = "snake_case")]
pub struct MessagesResponse {
    pub messages: Vec<Message>,
}
```

Contract.rs

```
#[cfg_attr(not(feature = "library"), entry_point)]
pub fn instantiate(
    deps: DepsMut,
    _env: Env,
    _info: MessageInfo,
    _msg: InstantiateMsg,
) -> Result<Response, ContractError> {
    // CURRENT_ID counter starts at 0
    CURRENT_ID.save(deps.storage, &Uint128::zero().u128())?;
    Ok(Response::default())
}

#[cfg_attr(not(feature = "library"), entry_point)]
pub fn execute(
    deps: DepsMut,
    _env: Env,
    info: MessageInfo,
    msg: ExecuteMsg,
) -> Result<Response, ContractError> {
    match msg {
        ExecuteMsg::AddMessage { topic, message } => add_message(deps, info, topic, message),
    }
}

pub fn add_message(
    deps: DepsMut,
    info: MessageInfo,
    topic: String,
    message: String,
) -> Result<Response, ContractError> {
    // Load the current id
    let mut id = CURRENT_ID.load(deps.storage)?;

    // Create message with id and function parameters
    let message = Message {
        id: Uint128::from(id),
        owner: info.sender,
        topic,
        message,
```

```
    };

    id = id.checked_add(1).unwrap();

    // Update message and updated id
    MESSAGES.save(deps.storage, id, &message)?;
    CURRENT_ID.save(deps.storage, &id)?;
    Ok(Response::new()
        .add_attribute("action", "add_message")
        .add_attribute("message_id", message.id.to_string()))
}

#[cfg_attr(not(feature = "library"), entry_point)]
pub fn query(deps: Deps, _env: Env, msg: QueryMsg) -> StdResult<Binary> {
    match msg {
            // Each message will return a binary response, its content will depend on the query
        QueryMsg::GetCurrentId {} => to_binary(&query_current_id(deps)?),     // Uint128
        QueryMsg::GetAllMessage {} => to_binary(&query_all_messages(deps)?),  // MessageResponse {}
        QueryMsg::GetMessagesByAddr { address } => {
            to_binary(&query_messages_by_addr(deps, address)?)                    // MessageResponse {}
        }
        QueryMsg::GetMessagesByTopic { topic } => to_binary(&query_messages_by_topic(deps, topic)?),
                                                                                  // MessageResponse {}
        QueryMsg::GetMessagesById { id } => to_binary(&query_messages_by_id(deps, id)?),
                                                                                  // MessageResponse {}
    }
}

fn query_current_id(deps: Deps) -> StdResult<Uint128> {
    // Load the current id
    let id = CURRENT_ID.load(deps.storage)?;
    Ok(Uint128::from(id))
}

fn query_all_messages(deps: Deps) -> StdResult<MessagesResponse> {
    // Query all  Messages entries (u128, Message), map only Message, no filter
    let messages = MESSAGES
        .range(deps.storage, None, None, Order::Ascending)
        .map(|map_data| map_data.unwrap().1)
        .collect();
    Ok(MessagesResponse { messages })
}

fn query_messages_by_addr(deps: Deps, address: String) -> StdResult<MessagesResponse> {
// range queries all the messages,
// map keeps the Message ( the key (u128 ) is dropped) for each element in the iterator,
// filter filters based on condition on the iterator elements,
// collect turns the iterator into a vector
    let messages = MESSAGES
        .range(deps.storage, None, None, Order::Ascending)
        .map(|map_entry| map_entry.unwrap().1)
        .filter(|message| message.owner == address)
        .collect();
    Ok(MessagesResponse { messages })
```

```
}

fn query_messages_by_topic(deps: Deps, topic: String) -> StdResult<MessagesResponse> {
    // Query all  Messages entries (u128, Message), map only Message, filter messages with topic parameter
    let messages = MESSAGES
        .range(deps.storage, None, None, Order::Ascending)
        .map(|map_entry| map_entry.unwrap().1)
        .filter(|message| message.topic == topic)
        .collect();

    Ok(MessagesResponse { messages })
}

fn query_messages_by_id(deps: Deps, id: Uint128) -> StdResult<MessagesResponse> {
    // Query all  Messages entries (u128, Message), map only Message, filter messages with id parameter
    let messages = MESSAGES
        .range(deps.storage, None, None, Order::Ascending)
        .map(|map_entry| map_entry.unwrap().1)
        .filter(|message| message.id == id)
        .collect();

    Ok(MessagesResponse { messages })
}
```

GENERAL QUESTIONS

1. **What are the concepts (borrowing, ownership, vectors etc)**
   The Concepts in the Code are Structs, Item, Map, Vectors, Tupples, Ownerships, Enum, Result.
   From cw_storage_plus::MAP we work with iterators, making use of range, map, filter and collect.

2. **What is the organization?**
   CosmWasm contract structure: contract.rs, state.rs, message.rs, error.rs, lib.rs

3. **What is the contract doing? What is the mechanism?**
   The contract stores messages, each with an owner, topic, message and id. This id is a counter stored in the contract that starts as zero and is increased after each new message.
   Few queries are implemented to query all messages stored, or get only those with a certain address, topic or owner.

4. **How could it be better? More efficient? Safer?**
   The Item and Map use a u128, whereas the struct uses Uint128 (serialization benefits).
   Not sure if we could use u128 as the keys in Item and Map and keep all the id of the same type.