Ordum Phat Contract

Specification and smart contract implementation for internal usage

Main contract parts

- Team & Individual profile management → (Trait CreateProfile)
- Key management for agile profiles (Team as DAOs) → (Trait CreateProfile)
- Project and Milestone management → (Trait Proposer & MilestoneTracker)
- Token and user authentication for off-chain database → (Trait OffchainDbAuth)
- Contract upgradability

Ordum Storage State

```
#[ink(storage)]
    pub struct OrdumState {
        individual_profile: Mapping<AccountId,IndividualProfile>,
        all_individuals: Vec<(String,AccountId)>,

        team_applicant_profile: Mapping<AccountId,TeamApplicantProfile>,
        all_applicant_teams: Vec<(String,AccountId)>,

        manage_keys: Vec<KeyManagement>,

        proposal: Mapping<AccountId,Vec<Project>>,
        db_auth: Mapping<AccountId,(String,Vec<u8>)>

        // **** Not implemented yet for Grant issuer profiles ****
        // Mapping issuer_id to a mapping of  application number to application profile
        // As this will enable specifi grant issuer to have dedicated list of queue application
        // and also teams to have numerous application per one issuer
        //queue_applications:Mapping<u16,Mapping<u32,u32>>,


    }
```

Ordum Phat contract errors

```
/// Error type for Create Profile
#[derive(Debug, PartialEq, Eq, scale::Encode, scale::Decode)]
#[cfg_attr(feature = "std", derive(scale_info::TypeInfo))]
pub enum Error {
    // Profile creation errors
    AccountExists,
    NotAuthorized,
    AccountDontExists,
    ProfileDontExists,
    MaxKeysExceeded,
    AccountExistsOrMaxExceeded,
    // Grant Application errors
    /// Any system related error
    UnexpectedError,
    SecretKeyNotAuthorized,
    SecretKeyAccountDontExists
}

/// Error types for Milestone Tracking
#[derive(Debug, PartialEq, Eq, scale::Encode, scale::Decode)]
#[cfg_attr(feature = "std", derive(scale_info::TypeInfo))]
pub enum MilestoneError {
```

```
    NotAuthorized,
    UnexpectedError,
    StorageExceeded,
    MilestoneNotFound,
    ProjectNotFound
}
```

Ordum Phat contract events

```
/// Event for individual profile creation
        #[ink(event)]
        pub struct IndividualProfileCreated {
            #[ink(topic)]
            name: String,
            account: AccountId,
            time: Timestamp
        }


    /// Event emitted when new Applicant is registered
        #[ink(event)]
        pub struct TeamApplicantCreated {
            #[ink(topic)]
            name: String,
            account: AccountId,
            time:  Timestamp
        }
    /// Event emitted when Applicant updates the profile
        #[ink(event)]
        pub struct TeamApplicantUpdated {
            #[ink(topic)]
            name: String,
            time: Timestamp
        }

    /// Event for setting passcode
        #[ink(event)]
        pub struct PasscodeSet {
            #[ink(topic)]
            account: AccountId
        }

    /// Event for notifying a reference team link has been updated per individual profile
        #[ink(event)]
        pub struct UpdatedTeamMembership {
            #[ink(topic)]
            team_name: String,
            team_id: AccountId,
            individual_id: AccountId,
            time: Timestamp
        }
```

Team & Individual profile management

A team profile has the following functionalities

- Profile creation
- Individual member addition and deletion
- Project application
- Milestone addition and editing
- Individual members roles editing and assignment

Team account implementation specifications

```
#[derive(Clone,Encode, Decode, Debug)]
#[cfg_attr(feature = "std",derive(StorageLayout,scale_info::TypeInfo))]
pub struct TeamApplicantProfile {
    name: String,
    account_id: AccountId,
    description: String,
    mission: String,
    chain: Vec<Chains>,
    members: Vec<(AccountId,MemberRole)>,
    registered_time: Timestamp,
    applications:u8,
    certificates: Vec<(String,u8)>,
    categories: Vec<Categories>,
    links: Vec<String>,

}
```

Account creation

Individual profile has the following functionalities

- Profile creation
- Project application is registered as an applicant
- Milestone addition and editing

Implementation specifications

```
#[derive(Clone,Encode, Decode, Debug)]
#[cfg_attr(feature = "std",derive(StorageLayout,scale_info::TypeInfo))]
pub struct IndividualProfile {
    name: String,
    account_id: AccountId,
    description: String,
    chains: Vec<Chains>,
    pub ref_team: Vec<(AccountId,MemberRole)>,
    applications: u8,
    certificates: Vec<(String,u8)>, // (CID,projectId)
    categories: Vec<Categories>,
    links: Vec<String>,
    role: UserRole
}
```

Account creation

Key management for agile profiles

Each ordum team profile can add members and assign roles to each member.

The roles define the privileges of the members.

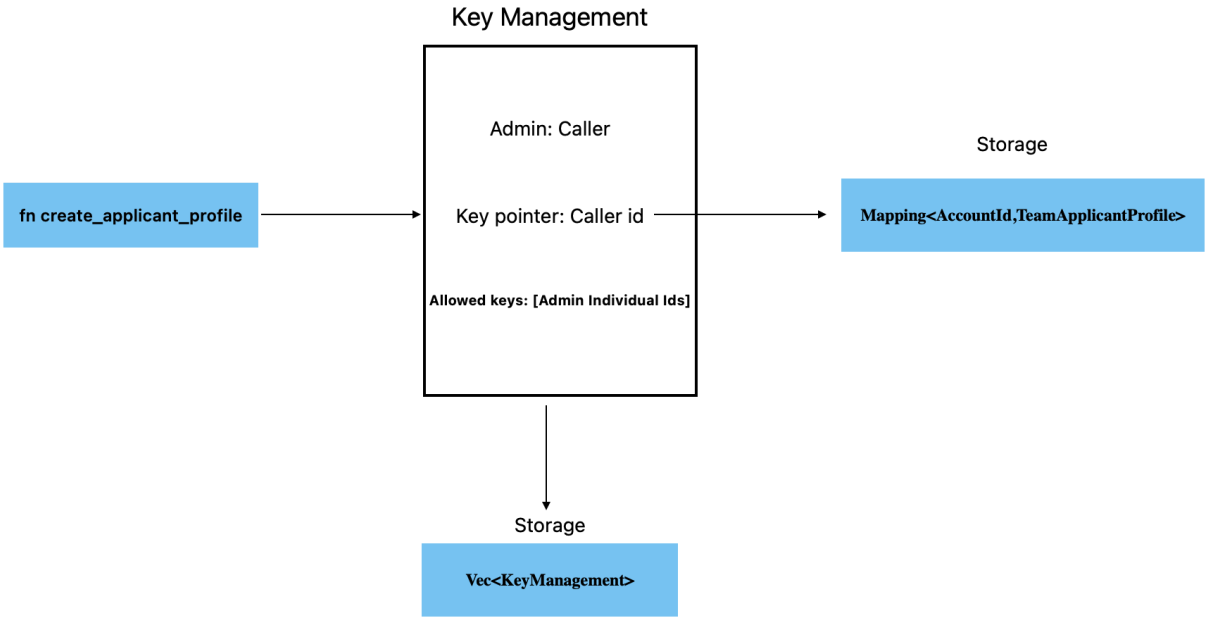
```
/// Team Member Roles
#[derive(Clone, Encode, Decode, Debug,PartialEq)]
#[cfg_attr(feature = "std", derive(StorageLayout, scale_info::TypeInfo))]
pub enum MemberRole {
    Admin,
    Regular
}
```

- Admin → has the privileges to
  - Add new team members
  - Edit the profile
  - Apply for new proposals
  - Submit and edit milestones

Key management is what allows the above functionality

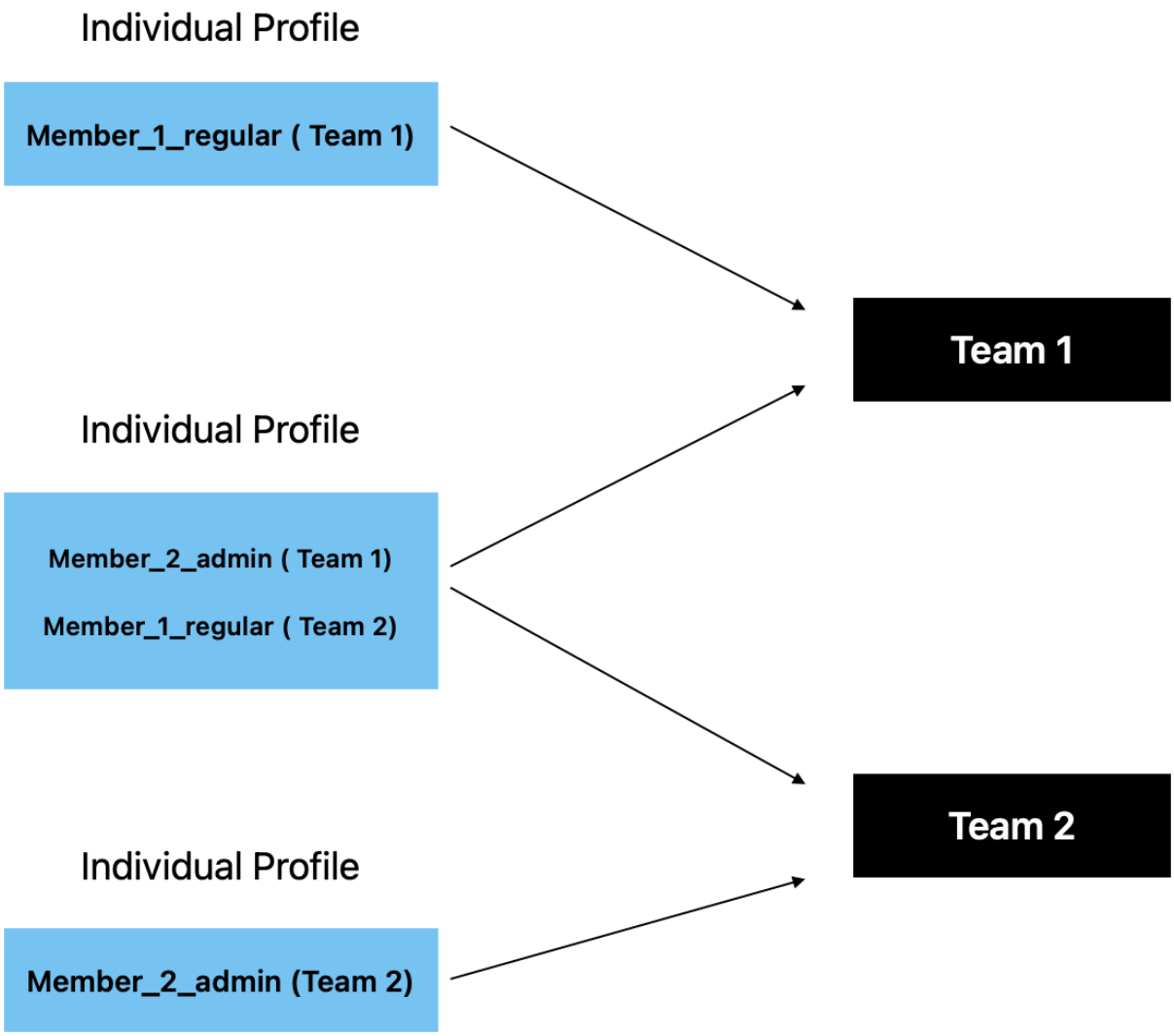
```
/// Key management struct
/// This will allow multiple members in certain organization to manage the account
/// The allowed members will be granted by `admin` key
/// The `key_pointer` is the key used in the key to `TeamApplicantProfile` mapping
#[derive(Clone,Encode,Hash, Decode, Debug)]
#[cfg_attr(feature = "std", derive(StorageLayout,scale_info::TypeInfo))]
pub struct KeyManagement{
    admin: AccountId,
    key_pointer: AccountId, // Account Id for now
    allowed_keys: Vec<AccountId>
}
```

Overall diagram on account creation and interaction



The above implementation enables Individual accounts to be agile in the Applicant Team profiles

- Individual accounts can be in different teams at once while having different roles in each teams



Project and Milestone management

Project management consists of;

- Managing proposal application files
- Managing milestones files per project
- Tracking memory used per proposal and milestone files stored in the ipfs
- All the files are just CID which links the stored files in the Crust/Ipfs

Project specifications

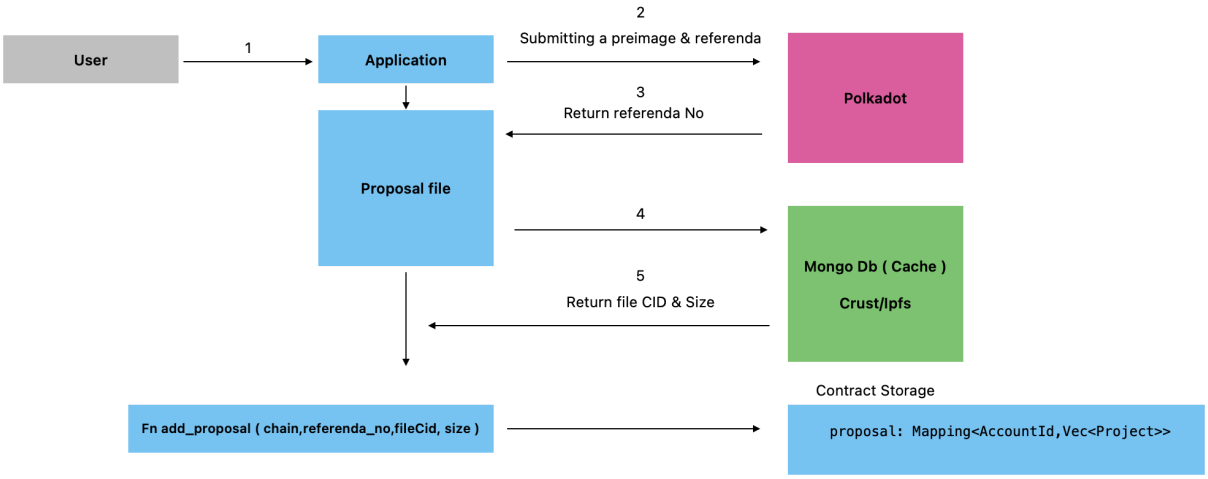
→ Project is to be specified whether it is an on-chain proposal with N referenda number and the chain for which is applied on or a non treasury proposal with None referenda number.

```
#[derive(Encode,Clone,Default, Decode,Debug)]
#[cfg_attr(feature = "std", derive(StorageLayout,scale_info::TypeInfo))]
pub struct InnerProject {
    pub chain: Chains,
    pub file: String,
```

```
    pub referenda_no: Option<u32>
}
```

```
#[derive(Encode,Clone,Default, Decode,Debug)]
#[cfg_attr(feature = "std", derive(StorageLayout,scale_info::TypeInfo))]
pub struct Project{
    id: u8,
    pub data: InnerProject,
    pub edited: Vec<(u8,Vec<EditedMile>)>, // (index == main milestone, value == Vec<EditedMilestonesPerMilestone>)
    pub main: Vec<AddMilestone>,
    pub pivoted: Vec<Vec<AddMilestone>>,
    //Utils
    pub pivot_reason: Option<Vec<String>>,
    pub pivot_index: Option<Vec<u8>>,
    pub total_mem: u32, // total memory used
}
```

Project interaction and storage

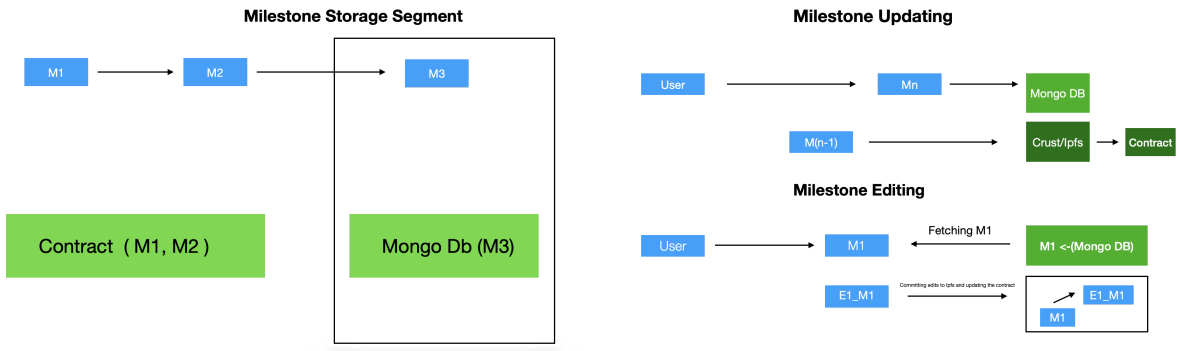


Milestones management

```
//---------------Milestone Tracking-----------------------------------------------------//

//------- Structure-------------//
//
//
//                  - - - - - - - - - - - - - - - - - - - - -> M6_E1 --> M6_E2 --> M6_E3
//             -                                              -
//         -                                                    -
// Edited  ------------> M2_E1 --> M2_E2                 -
//                     -                               -
//                   -                               -
// Main --> M1 --> M2 --> M3 --> M4 --> M5 --> M6 --> M7
//                                                      -
//                                                        -
// Pivoted ------------------------------------------------> P_M7 --> P_M8 --> P_M9
```

Milestone storage and interaction



Milestone specifications

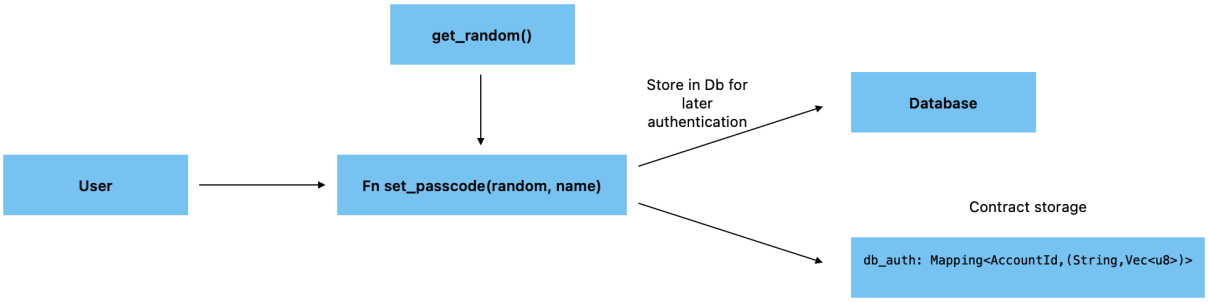
```
#[derive(Encode,Clone,Default, Decode, Debug)]
#[cfg_attr(feature = "std", derive(StorageLayout,scale_info::TypeInfo))]
pub struct EditedMile {
    pub edited_index:u8,
    pub main_index:u8,
    data:String,
    mem:u32 // Storing the byte memory of the stored file pointing to IPFS
}

#[derive(Encode,Clone,Default, Decode, Debug)]
#[cfg_attr(feature = "std", derive(StorageLayout,scale_info::TypeInfo))]
pub struct AddMilestone {
    pub main_index:u8,
    pub no_edits:u8,
    data: String,
    mem: u32
}
```

Token and user authentication for off-chain database

Functionalities

- To authenticate user to access off-chain database data
- Manage same user authentication details in the contract and off-chain database



Contract upgradability

Ordum upgradability will be based on updating the web-assembly blob file.

To be updated and include authentication logic

```
#[ink(message, selector = 0xC0DE2000 )]
        pub fn set_code(&mut self, code_hash: [u8; 32]) {
            ink::env::set_code_hash(&code_hash).unwrap_or_else(|err| {
                panic!(
                    "Failed to `set_code_hash` to {:?} due to {:?}",
                    code_hash, err
                )
            });
            ink::env::debug_println!(" Switched code hash to {:?}.", code_hash);
        }
```