SuiMover

SuiMover Bootcamp Class#6 - KIOSK

Wayne Kuo

suimover.org

Agenda

Section 1 Sui Kiosk 基礎概念

Section 2 Sui Kiosk 程式碼

Section 3 NFT Project Example With Kiosk

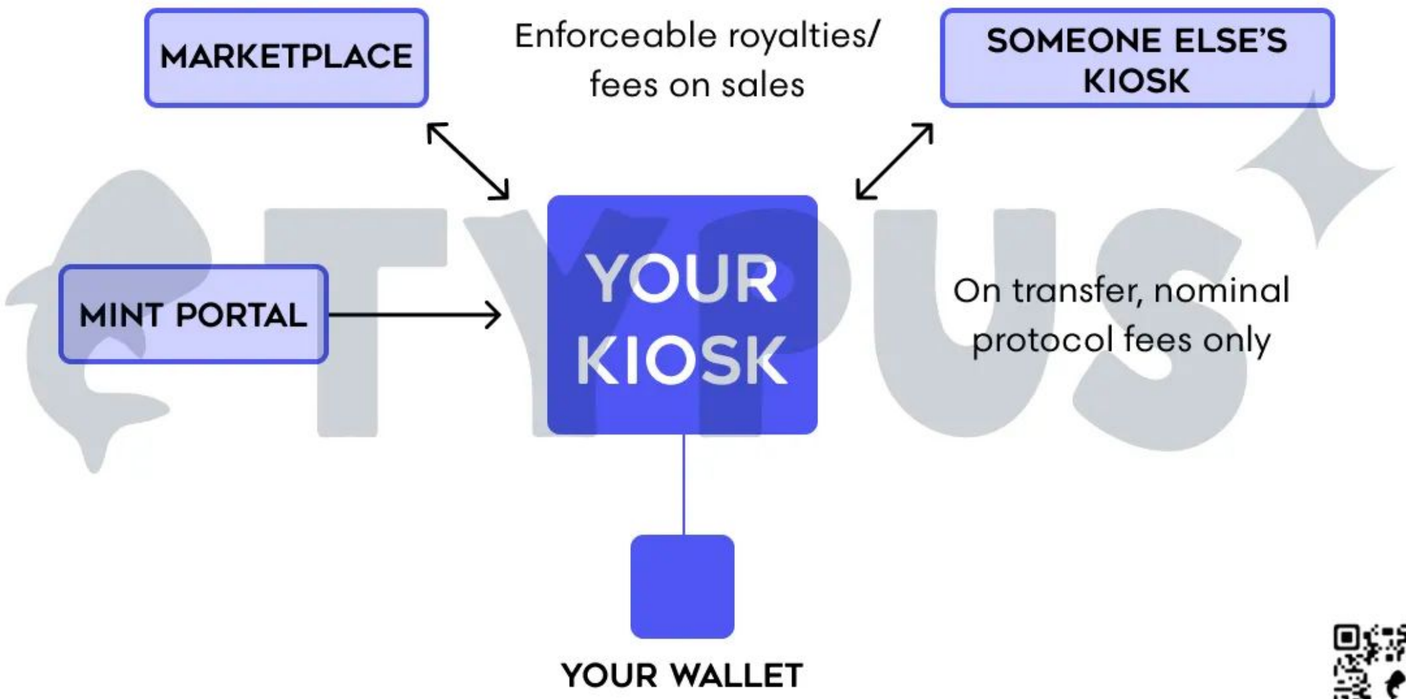
Section 1

Sui Kiosk 基礎概念

Section 1

Sui Object NFT 的交易問題

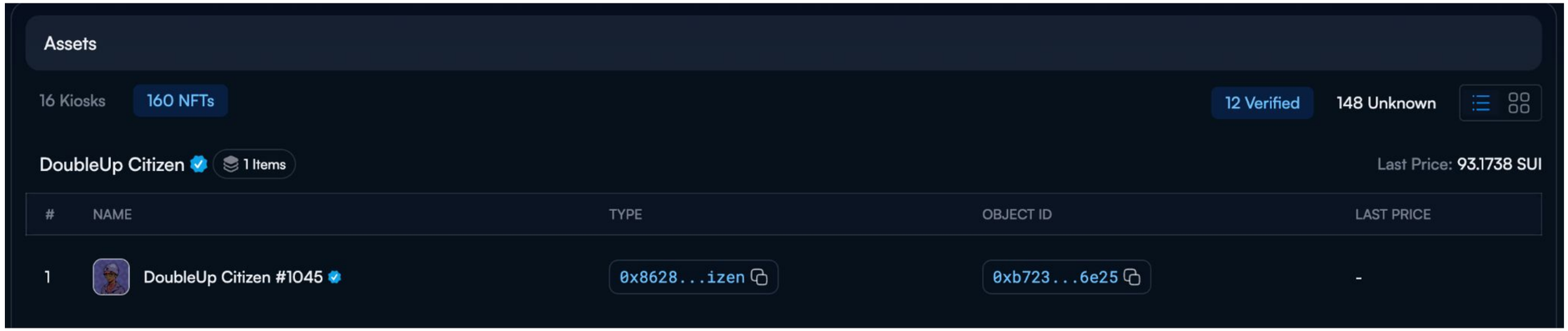
- **market place**
  - 缺少通用的規範
  - 需要第三方託管的風險
- **royalty fee**
  - 發行方無法課徵版稅

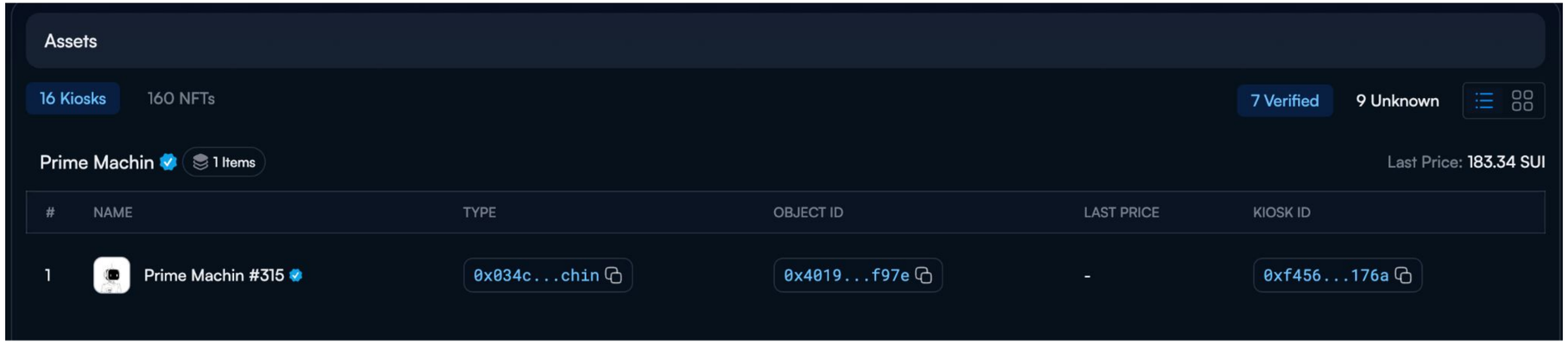


https://www.blocktempo.com/sui-storage-of-digital-assets-changing-the-nft-usage/

Section 1

Object vs Kiosk

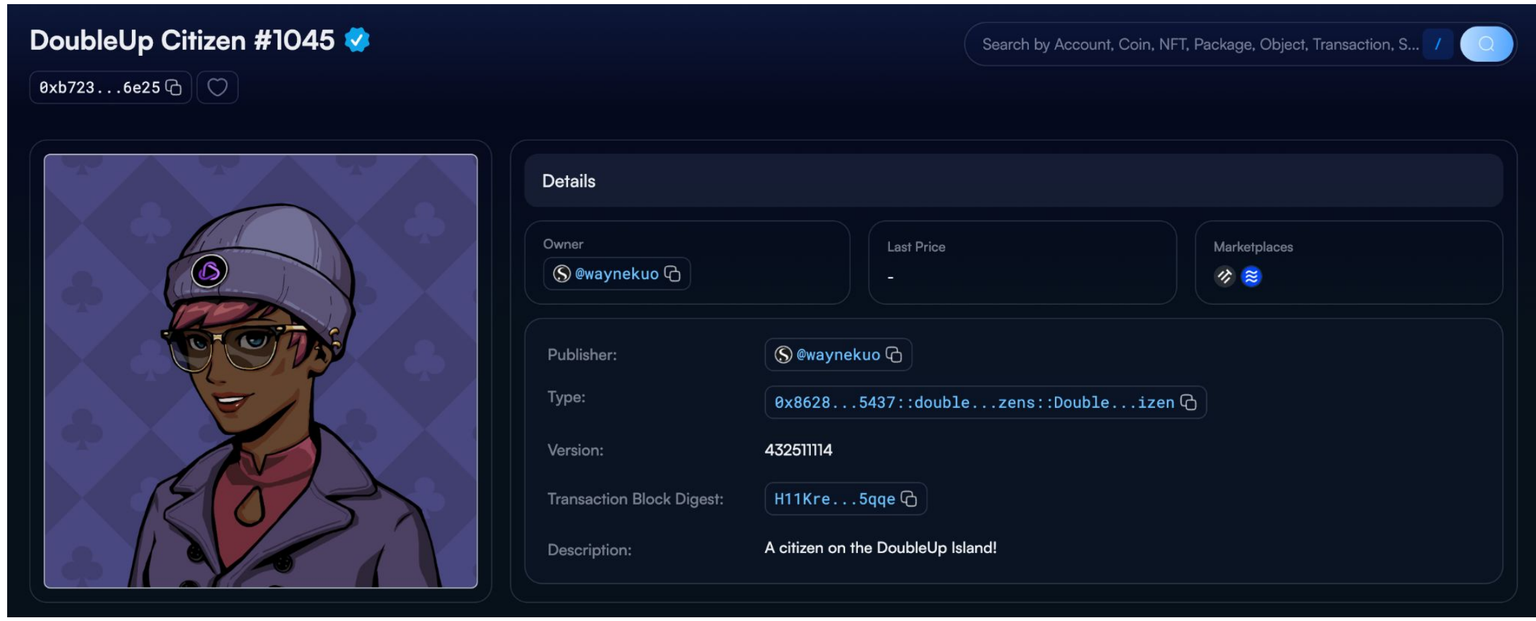


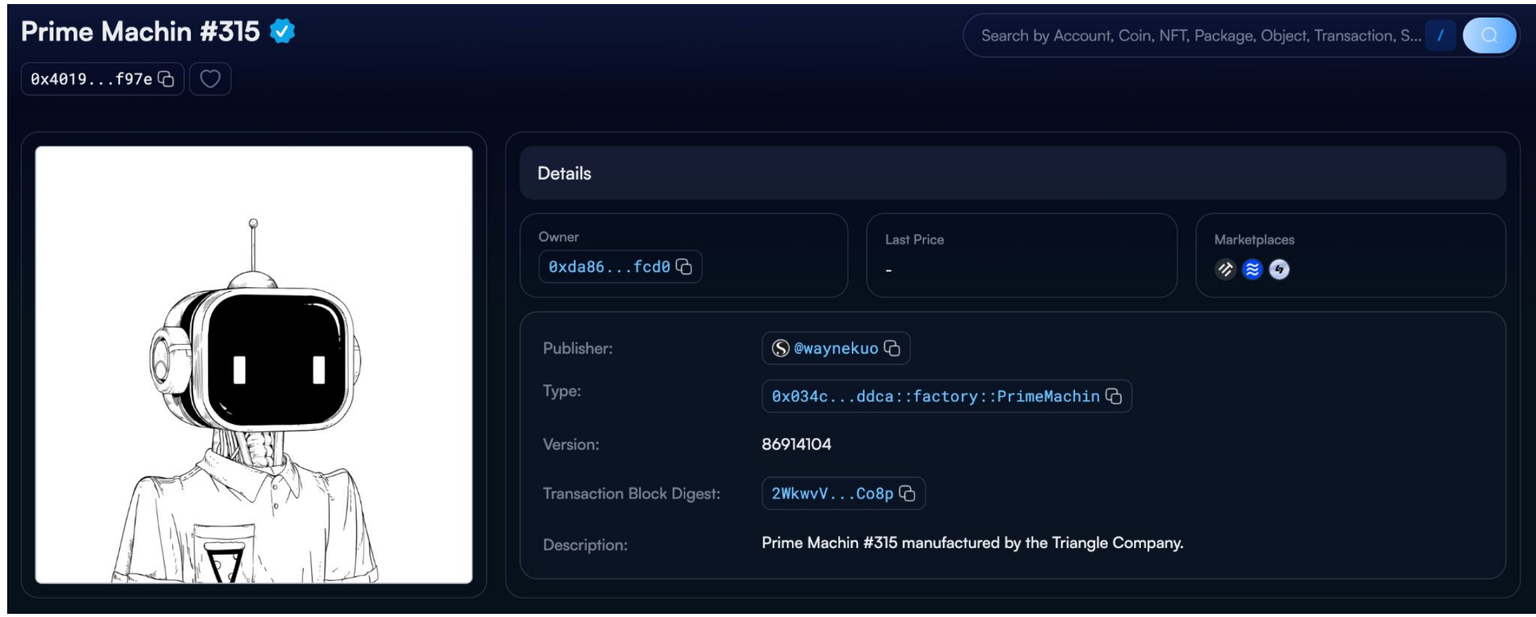


Section 1

Object vs Kiosk

- Object NFT
  - 直接為用戶所持有
  - 可以直接轉移
  - 掛賣時會轉移所有權到市場
- Kiosk NFT
  - 作為 dynamic field 存放 Kiosk 裡
  - 不能直接轉移
  - 成交時才會轉移所有權

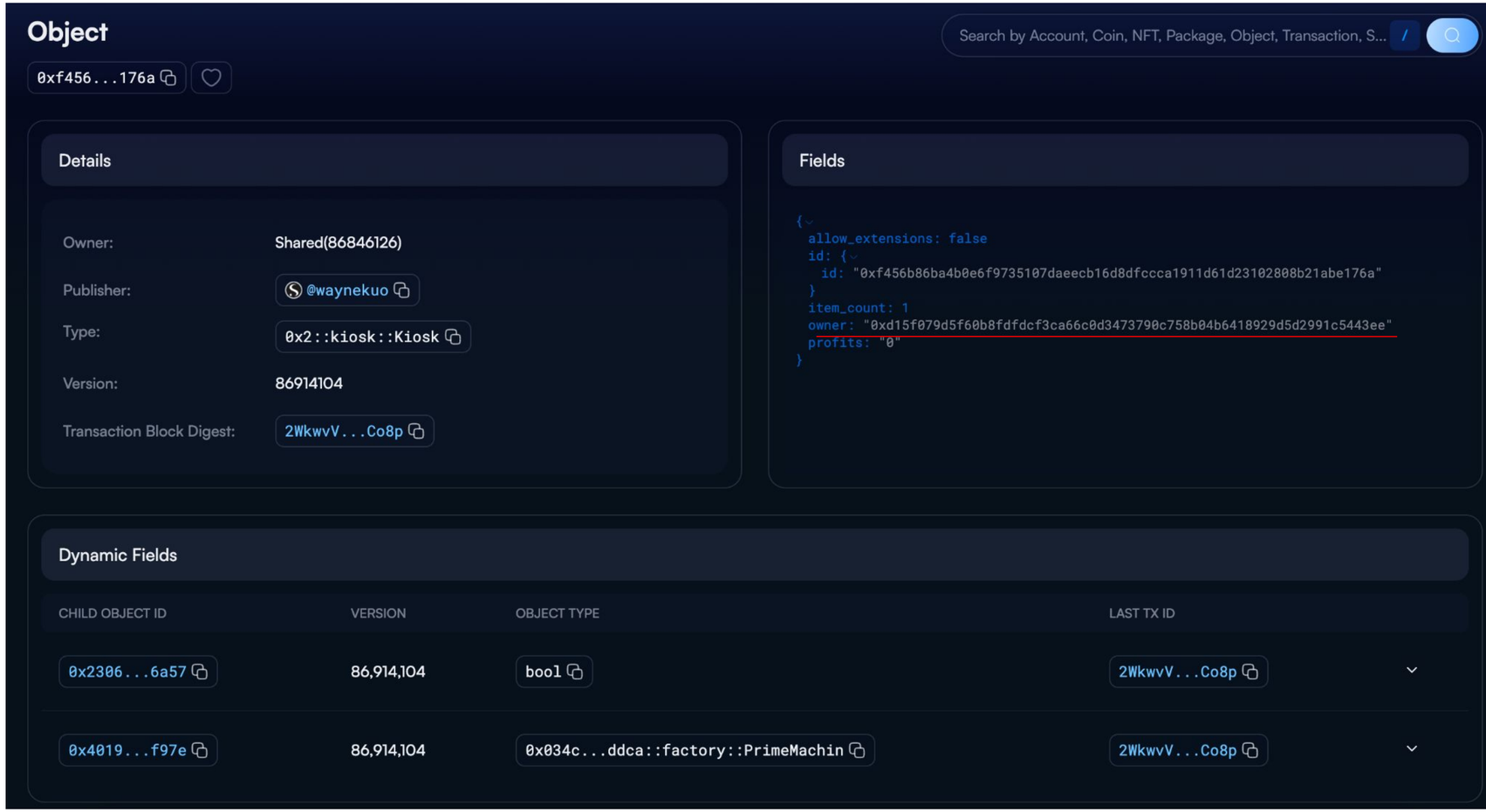




Section 1

Kiosk

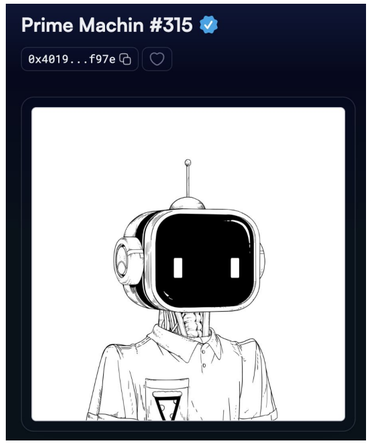
- Shared Object
- NFT 以 dynamic field 存放
- fields 裡的owner 沒有實質作用
- KioskOwnerCap 作為擁有者權限(可轉移)

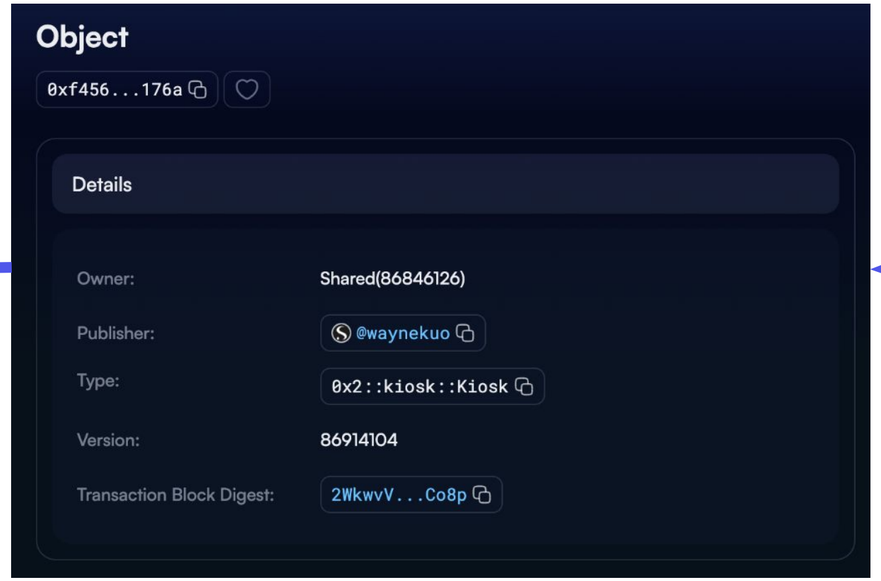


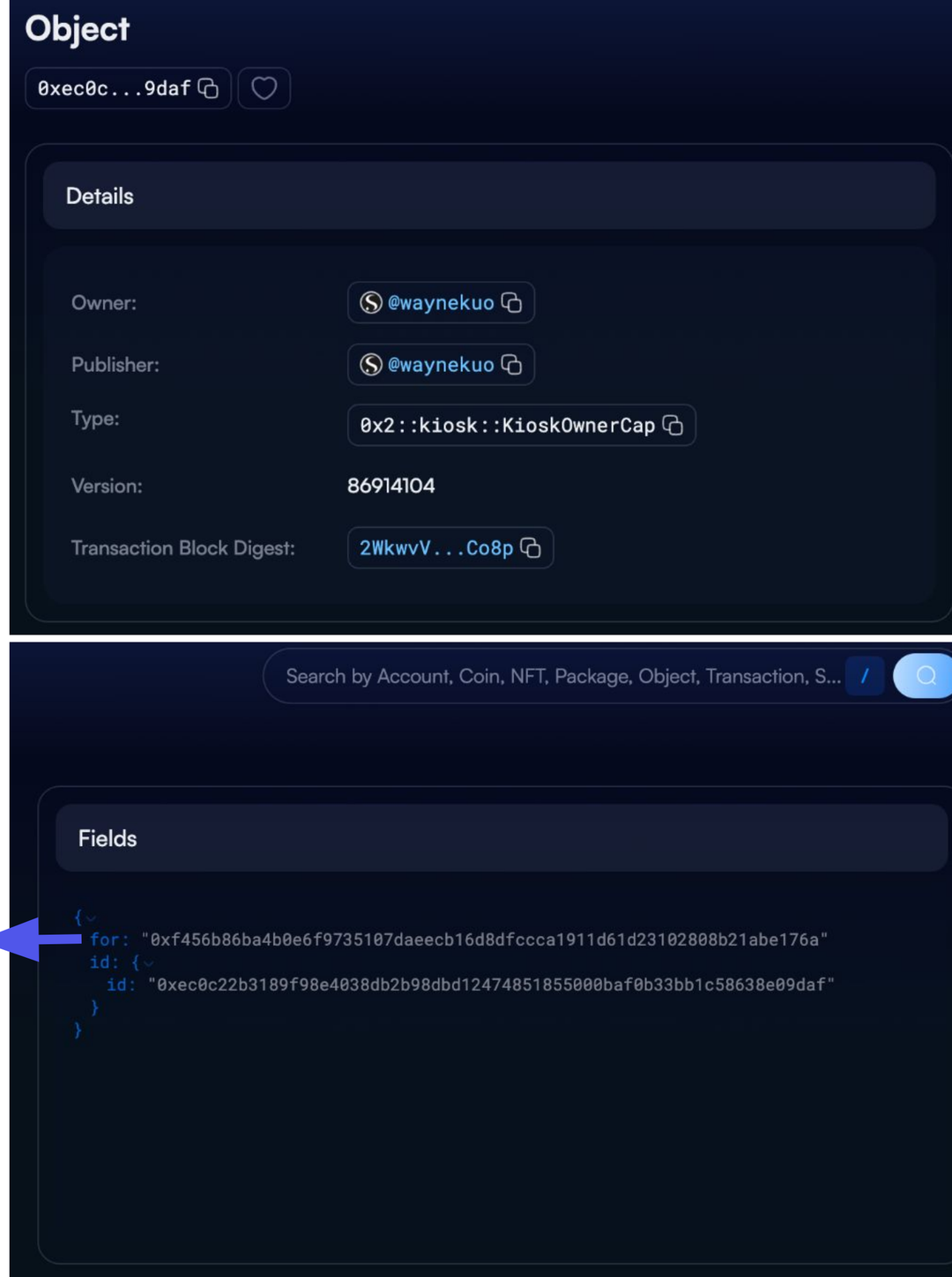
Section 1

KioskOwnerCap

- Owned Object(可轉移)
- 作為Kiosk擁有者的實質權限
- KioskOwnerCap (Owned)
  - -> Kiosk (Shared)
  - -> NFT (dynamic field)



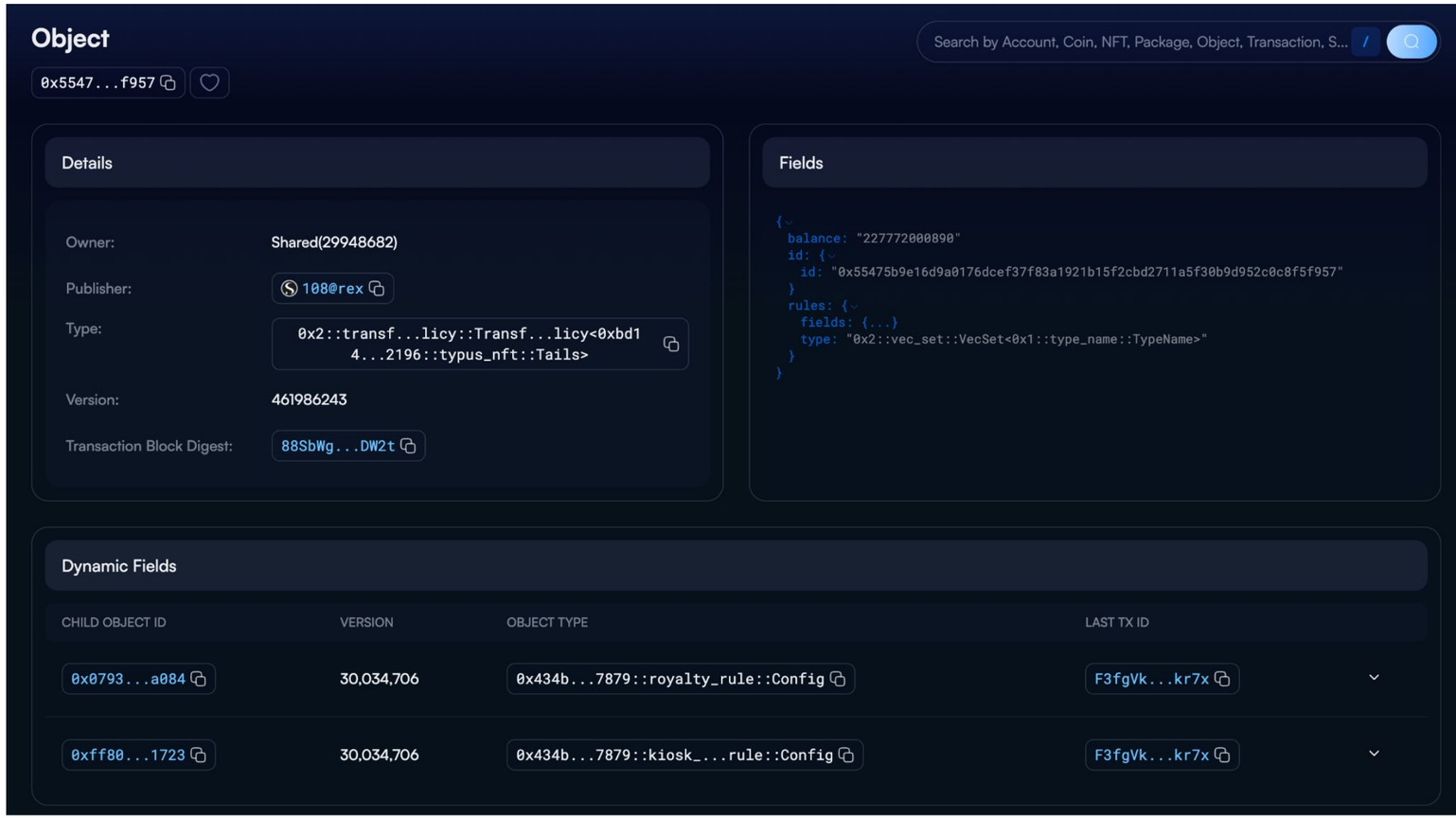




Section 1

TransferPolicy

- 每筆Kisok內的交易都會需要通過TransferPolicy的驗證
- 創作者可以通過設定過TransferPolicy裡面的Rule來制定相關規則
  - royalty_rule 課徵版稅
  - kiosk_lock_rule 將NFT鎖在Kiosk裡
  - personal_kiosk_rule 將NFT鎖在Personal Kiosk裡



https://docs.sui.io/standards/kiosk#sui-kiosk-guarantees

Section 1

TransferRequest

- 在交易時透過new_request生成 TransferRequest
- 根據TransferPolicy內制定的Rule去做相關驗證
  - royalty_rule：通過pay去交版稅
  - kiosk_lock_rule：通過prove驗證是否lock
- Rule 皆通過後由transfer_policy::confirm_request 完成交易

MoveCall 9 pay

Package: 0x434b5bd8f6a7b05fede0ff46c6e511d71ea326ed38056e3bcd681d2d7c2a7879

Module: royalty_rule

Function: pay

Type Arguments: 0xbd147bc7f12f38f175d78947a61364e8e077b9b188b00e7094bc0c3623162196::typus_nft::Tails

Arguments:


```
[
  0: {
    type: "object"
    objectType: "sharedObject"
    objectId: "0x55475b9e16d9a0176dcef37f83a1921b15f2cbd2711a5f30b9d952c0c8f5f957"
    initialSharedVersion: "29948682"
    mutable: true
  }
  1: {
    NestedResult: [
      0: 4
      1: 1
    ]
  }
  2: {
    Result: 8
  }
]
```


MoveCall 10 confirm_request

Package: 0x0000000000000000000000000000000000000000000000000000000000000002

Module: transfer_policy

Function: confirm_request

Type Arguments: 0xbd147bc7f12f38f175d78947a61364e8e077b9b188b00e7094bc0c3623162196::typus_nft::Tails

Arguments:


```
[
  0: {
    type: "object"
    objectType: "sharedObject"
    objectId: "0x55475b9e16d9a0176dcef37f83a1921b15f2cbd2711a5f30b9d952c0c8f5f957"
    initialSharedVersion: "29948682"
    mutable: true
  }
  1: {
    NestedResult: [
      0: 4
      1: 1
    ]
  }
]
```


https://suivision.xyz/txblock/51gZNnPvUWMXP6mWGocg5MNmT71Qg5KG6jVNSJmBXWqm

Section 2

Sui Kiosk 程式碼

Section 2

kiosk borrow vs borrow_val

borrow / borrow_mut

- 只能拿到reference或是mutable reference
- 無需歸還

borrow_val

- borrow_val 可以把 NFT 本體 (T) 拿出來，同時生成一個hot potato Borrow
- 需要透過return_val 去歸還 T，同時把 Borrow 消除

```
#[syntax(index)]
/// Immutably borrow an item from the `Kiosk`. Any item can be `borrow`ed
/// at any time.
public fun borrow<T: key + store>(self: &Kiosk, cap: &KioskOwnerCap, id: ID): &T {
    assert!(object::id(obj: self) == cap.`for`, ENotOwner);
    assert!(self.has_item(id: id), EItemNotFound);

    dof::borrow(object: &self.id, name: Item { id })
}

#[syntax(index)]
/// Mutably borrow an item from the `Kiosk`.
/// Item can be `borrow_mut`ed only if it's not `is_listed`.
public fun borrow_mut<T: key + store>(self: &mut Kiosk, cap: &KioskOwnerCap, id: ID): &mut T {
    assert!(self.has_access(cap: cap), ENotOwner);
    assert!(self.has_item(id: id), EItemNotFound);
    assert!(!self.is_listed(id: id), EItemIsListed);

    dof::borrow_mut(object: &mut self.id, name: Item { id })
}
```

```
/// Take the item from the `Kiosk` with a guarantee that it will be returned.
/// Item can be `borrow_val`-ed only if it's not `is_listed`.
public fun borrow_val<T: key + store>(self: &mut Kiosk, cap: &KioskOwnerCap, id: ID): (T, Borrow) {
    assert!(self.has_access(cap: cap), ENotOwner);
    assert!(self.has_item(id: id), EItemNotFound);
    assert!(!self.is_listed(id: id), EItemIsListed);

    (dof::remove(object: &mut self.id, name: Item { id }), Borrow { kiosk_id: object::id(obj: self), item_id: id })
}

/// Return the borrowed item to the `Kiosk`. This method cannot be avoided
/// if `borrow_val` is used.
public fun return_val<T: key + store>(self: &mut Kiosk, item: T, borrow: Borrow) {
    let Borrow { kiosk_id: ID, item_id: ID } = borrow;

    assert!(object::id(obj: self) == kiosk_id, EWrongKiosk);
    assert!(object::id(obj: &item) == item_id, EItemMismatch);

    dof::add(object: &mut self.id, name: Item { id: item_id }, value: item);
}
```

Section 2

kiosk list

- 把指定的NFT ID 掛賣指定的價格 price
- 此時NFT 本體並不會轉出

```
/// List the item by setting a price and making it available for purchase.
/// Performs an authorization check to make sure only owner can sell.
public fun list<T: key + store>(self: &mut Kiosk, cap: &KioskOwnerCap, id: ID, price: u64) {
    assert!(self.has_access(cap: cap), ENotOwner);
    assert!(self.has_item_with_type<T>(id: id), EItemNotFound);
    assert!(!self.is_listed_exclusively(id: id), EListedExclusively);

    df::add(object: &mut self.id, name: Listing { id, is_exclusive: false }, value: price);
    event::emit(event: ItemListed<T> { kiosk: object::id(obj: self), id, price })
}
```

Section 2

kiosk purchase

- 指定Kisok及交易的NFT ID發起（NFT需list掛賣）
- 同時把售價對應的SUI Coin送入
- 會通過transfer_policy::new_request生成一個需要驗證rule的hot potato
- 以及購買的NFT 本體 (T)

```
/// Make a trade: pay the owner of the item and request a Transfer to the `target`
/// kiosk (to prevent item being taken by the approving party).
///
/// Received `TransferRequest` needs to be handled by the publisher of the T,
/// if they have a method implemented that allows a trade, it is possible to
/// request their approval (by calling some function) so that the trade can be
/// finalized.
public fun purchase<T: key + store>(
    self: &mut Kiosk,
    id: ID,
    payment: Coin<SUI>,
): (T, TransferRequest<T>) {
    let price: u64 = df::remove<Listing, u64>(object: &mut self.id, name: Listing { id, is_exclusive: false });
    let inner: T = dof::remove<Item, T>(object: &mut self.id, name: Item { id });

    self.item_count = self.item_count - 1;
    assert!(price == payment.value(), EIncorrectAmount);
    df::remove_if_exists<Lock, bool>(object: &mut self.id, name: Lock { id });
    coin::put(balance: &mut self.profits, coin: payment);

    event::emit(event: ItemPurchased<T> { kiosk: object::id(obj: self), id, price });

    (inner, transfer_policy::new_request(item: id, paid: price, from: object::id(obj: self)))
}
```

Section 2

TransferRequest

作為Hot Potato (沒有Drop) 需要透過confirm_request去消除

- **item** 被交易的NFT ID
- **paid** 售價
- **from** 存放此 NFT 的 Kiosk
- **receipts** 通過Rule檢驗後的證明

```
/// A "Hot Potato" forcing the buyer to get a transfer permission
/// from the item type (`T`) owner on purchase attempt.
public struct TransferRequest<phantom T> {
    /// The ID of the transferred item. Although the `T` has no
    /// constraints, the main use case for this module is to work
    /// with Objects.
    item: ID,
    /// Amount of SUI paid for the item. Can be used to
    /// calculate the fee / transfer policy enforcement.
    paid: u64,
    /// The ID of the Kiosk / Safe the object is being sold from.
    /// Can be used by the TransferPolicy implementors.
    from: ID,
    /// Collected Receipts. Used to verify that all of the rules
    /// were followed and `TransferRequest` can be confirmed.
    receipts: VecSet<TypeName>,
}
```

```
/// Allow a `TransferRequest` for the type `T`. The call is protected
/// by the type constraint, as only the publisher of the `T` can get
/// `TransferPolicy<T>`.
///
/// Note: unless there's a policy for `T` to allow transfers,
/// Kiosk trades will not be possible.
public fun confirm_request<T>(
    self: &TransferPolicy<T>,
    request: TransferRequest<T>,
): (ID, u64, ID) {
    let TransferRequest { item: ID, paid: u64, from: ID, receipts: VecSet } = request;
    let mut completed: vector<TypeName> = receipts.into_keys();
    let mut total: u64 = completed.length();

    assert!(total == self.rules.size(), EPolicyNotSatisfied);

    while (total > 0) {
        let rule_type: TypeName = completed.pop_back();
        assert!(self.rules.contains(key: &rule_type), EIllegalRule);
        total = total - 1;
    };

    (item, paid, from)
}
```

```
/// Construct a new `TransferRequest` hot potato which requires an
/// approving action from the creator to be destroyed / resolved. Once
/// created, it must be confirmed in the `confirm_request` call otherwise
/// the transaction will fail.
public fun new_request<T>(item: ID, paid: u64, from: ID): TransferRequest<T> {
    TransferRequest { item, paid, from, receipts: vec_set::empty() }
}
```

Section 2

TransferPolicy

- **balance**
  a. 存放收到的版稅
- **rules**
  a. 創作者自由設定的規則
  b. 需全部滿足才能通過confirm_request
- **TransferPolicyCap**
  a. TransferPolicy的權限

```
/// A unique capability that allows the owner of the `T` to authorize
/// transfers. Can only be created with the `Publisher` object. Although
/// there's no limitation to how many policies can be created, for most
/// of the cases there's no need to create more than one since any of the
/// policies can be used to confirm the `TransferRequest`.
public struct TransferPolicy<phantom T> has key, store {
    id: UID,
    /// The Balance of the `TransferPolicy` which collects `SUI`.
    /// By default, transfer policy does not collect anything , and it's
    /// a matter of an implementation of a specific rule - whether to add
    /// to balance and how much.
    balance: Balance<SUI>,
    /// Set of types of attached rules - used to verify `receipts` when
    /// a `TransferRequest` is received in `confirm_request` function.
    ///
    /// Additionally provides a way to look up currently attached Rules.
    rules: VecSet<TypeName>,
}

/// A Capability granting the owner permission to add/remove rules as well
/// as to `withdraw` and `destroy_and_withdraw` the `TransferPolicy`.
public struct TransferPolicyCap<phantom T> has key, store {
    id: UID,
    policy_id: ID,
}
```

Section 2

add_rule

```
/// Add a custom Rule to the `TransferPolicy`. Once set, `TransferRequest` must
/// receive a confirmation of the rule executed so the hot potato can be unpacked.
///
/// - T: the type to which TransferPolicy<T> is applied.
/// - Rule: the witness type for the Custom rule
/// - Config: a custom configuration for the rule
///
/// Config requires `drop` to allow creators to remove any policy at any moment,
/// even if graceful unpacking has not been implemented in a "rule module".
public fun add_rule<T, Rule: drop, Config: store + drop>(
    _: Rule,
    policy: &mut TransferPolicy<T>,
    cap: &TransferPolicyCap<T>,
    cfg: Config,
) {
    assert!(object::id(obj: policy) == cap.policy_id, ENotOwner);
    assert!(!has_rule<T, Rule>(policy: policy), ERuleAlreadySet);
    df::add(object: &mut policy.id, name: RuleKey<Rule> {}, value: cfg);
    policy.rules.insert(key: type_name::get<Rule>())
}
```

main apps / kiosk / sources / rules / Go

damirka [mainnet] PersonalKiosk + Rule (#11)

Name	Last commit message
..	
floor_price_rule.move	[mainnet] PersonalKiosk + Rule (#11)
kiosk_lock_rule.move	adds kiosk
personal_kiosk_rule.move	[mainnet] PersonalKiosk + Rule (#11)
royalty_rule.move	adds kiosk
witness_rule.move	adds kiosk

https://github.com/MystenLabs/apps/tree/main/kiosk/sources/rules

Section 3

NFT Project Example With Kiosk

Section 3

init

Key Points:

- transfer_policy
- add kiosk rule

```
struct Royalty has key {
    id: UID,
    recipient: address,
    policy_cap: TransferPolicyCap<Tails>
}
```

```
let (policy: TransferPolicy, policy_cap: TransferPolicyCap) = transfer_policy::new<Tails>(pub: &publisher, ctx…ctx);
// 1. add royalty_rule
kiosk_royalty_rule::add(policy: &mut policy, cap: &policy_cap, amount_bp: 1_000, min_amount: 1_000_000_000); // MAX(10

// 2. add kiosk_lock_rule
kiosk_lock_rule::add(policy: &mut policy, cap: &policy_cap);

// 3. add personal_kiosk_rule
personal_kiosk_rule::add(policy: &mut policy, cap: &policy_cap);

let royalty: Royalty = Royalty {
    id: object::new(ctx: ctx),
    recipient: @ADMIN,
    policy_cap
};

transfer::public_share_object(obj: policy);
// transfer::public_transfer(policy_cap, sender);
transfer::share_object(obj: royalty);
```

Section 3

new kiosk

```
sui client call \
    --package 0x2 \
    --module kiosk \
    --function default \
    --gas-budget 1000000000
```

```
#[allow(lint(self_transfer))]
/// Creates a new Kiosk in a default configuration: sender receives the
/// `KioskOwnerCap` and becomes the Owner, the `Kiosk` is shared.
entry fun default(ctx: &mut TxContext) {
    let (kiosk: Kiosk, cap: KioskOwnerCap) = new(ctx: ctx);
    sui::transfer::transfer(obj: cap, recipient: ctx.sender());
    sui::transfer::share_object(obj: kiosk);
}

/// Creates a new `Kiosk` with a matching `KioskOwnerCap`.
public fun new(ctx: &mut TxContext): (Kiosk, KioskOwnerCap) {
    let kiosk: Kiosk = Kiosk {
        id: object::new(ctx: ctx),
        profits: balance::zero(),
        owner: ctx.sender(),
        item_count: 0,
        allow_extensions: false,
    };

    let cap: KioskOwnerCap = KioskOwnerCap {
        id: object::new(ctx: ctx),
        `for`: object::id(obj: &kiosk),
    };

    (kiosk, cap)
}
```

Section 3

mint

```
entry fun free_mint_into_kiosk(
    pool: &mut Pool,
    policy: &TransferPolicy<Tails>,
    whitelist_token: Whitelist,
    kiosk: &mut Kiosk,
    kiosk_cap: &KioskOwnerCap,
    random: &Random, // 0x8
    ctx: &mut TxContext,
) {
    let nft: Tails = mint(pool: pool, whitelist_token: whitelist_token, random: random, ctx: ctx);
    kiosk::lock(self: kiosk, cap: kiosk_cap, _policy: policy, item: nft);
}
```

```
sui client ptb \
--assign pool @0xbc583ae6c5a185ae1d74e7f979f0f57b3b579abc54b6d1141bf4f1889d98ec10 \
--assign policy @0x9d7f0ec42b5b1b790c893f2679c4edc1efe1d5f20ca63cf23c2460b0042d74d8 \
--assign whitelist @0xe95ff9dcc835f0320cdb69c2d19f8571c3bad23298f14b3f405caed18b927e4b \
--assign kiosk @0x90df9555659e8d1fe6a57e8c1f1c67a2a093b0ba3ae3de23da2a46d3d3b4b599 \
--assign kiosk_cap @0x4854d1d173f8b13d7449e6081159a543d3dfaa3f466f7c80a0b1f73ac561de00 \
--assign random @0x8 \
--move-call 0x27321bc52766f3ed3f809524ca0149bdbbf01f7f18bdccc261eab2dc5fa14589::mover_nft::free_mint_into_kiosk pool policy whitelist kiosk kiosk_cap random \
--gas-budget 100000000
```

Section 3

Exercise 6

```
public fun goal(
    world: &KapyWorld,
    crew: &mut KapyCrew,
    _simple_nft: &SimpleNFT,
    _tails: &Tails,
    ctx: &mut TxContext
) {
    let pirate: KapyPirate = kapy_pirate::new(
    world: world,
    kind: PIRATE_KIND,
        _rule_witness: NFT_EXERCISE {},
    ctx: ctx,
    );
    crew.recruit(pirate: pirate);
}
```

```
public fun goal_2(
    world: &KapyWorld,
    crew: &mut KapyCrew,
    _simple_nft: &SimpleNFT,
    tails: Tails,
    ctx: &mut TxContext
): Tails {
    let pirate: KapyPirate = kapy_pirate::new(
    world: world,
    kind: PIRATE_KIND,
        _rule_witness: NFT_EXERCISE {},
    ctx: ctx,
    );
    crew.recruit(pirate: pirate);
    tails
}
```

THANK YOU

PRESENTATION TEMPLATE

suimover.org

End Slide