**REQUIREMENTS:**

# **Functional Requirements:**

## **User Management Module (UMM)**

1. **US-UMM-001 - User Registration and Authentication**

* As a **property owner**, I want to register and authenticate securely using multi-factor authentication, so that I can manage my properties safely.
* As a **renter or buyer**, I want to create an account and verify my identity securely, so that I can rent or buy properties with confidence.

1. **US-UMM-002 - Role-Based Access Control (RBAC)**

* As a **system administrator**, I want to enforce role-based access control, so that users have access only to the functionalities relevant to their role.

1. **US-UMM-003 - Identity Verification for Property Owners**
   * As a **property owner**, I want to verify my identity using government-issued documents, so that I can list properties without fraudulent concerns.
   * As a **renter or buyer**, I want to be assured that the property owners are verified, so that I can trust the listings.\*
2. **US-UMM-004 - Client Reputation and Review System**

* As a **property owner**, I want to view the rental history and reputation score of potential tenants, so that I can make informed decisions.
* As a **renter or buyer**, I want to leave and receive reviews after transactions, so that I can build a trustworthy profile over time.

## **Property Management Module (PMM)**

1. **US-PMM-001 - Property Ownership Registration on Blockchain**

* As a **property owner**, I want to register my property on a blockchain ledger, so that ownership rights are immutable and transparent.
* As a **buyer**, I want to verify property ownership on the blockchain before making a purchase, so that I can avoid fraudulent transactions.

1. **US-PMM-002 - Property Listing for Sale and Rental**

* As a **property owner**, I want to list my property for rent or sale, so that I can attract potential renters or buyers.
* As a **renter or buyer**, I want to browse listed properties with complete details, so that I can make informed decisions.

1. **US-PMM-003 - Ownership Verification via Smart Contracts**

* As a **buyer**, I want to verify the ownership of a property via smart contracts, so that I can ensure the seller is legitimate.

1. **US-PMM-004 - Property Ownership Transfer**

* As a **property owner**, I want to transfer ownership rights securely via the blockchain when selling my property, so that the records remain immutable.
* As a **buyer**, I want to receive proof of ownership transfer on the blockchain, so that I can confirm my legal rights over the property.

## **Rental Management Module (RMM)**

1. **US-RMM-001 - Rental Agreement via Smart Contracts**

* As a **property owner**, I want rental agreements to be automatically created and enforced via smart contracts, so that both parties follow the agreed terms.\*(future implementation\*legal validity)
* As a **renter**, I want my rental agreement to be securely stored on the blockchain, so that I have proof of my lease terms.

1. **US-RMM-002 - Tenant Rental History Tracking**

* As a **property owner**, I want to access the rental history of a property, so that I can verify the previous tenants.
* As a **renter**, I want my rental history to be recorded, so that I can prove my past rental experiences for future transactions.

1. **US-RMM-003 - Consensus-Based Rental Verification**

* As a **renter**, I want to verify the legitimacy of a new rental claim using feedback from previous tenants, so that I can avoid rental scams.
* As a **previous renter**, I want to confirm whether I rented a property from the listed owner, so that I can help prevent fraud.

1. **US-RMM-004 - Property Return and Condition Verification**

* As a **renter**, I want to document the property's condition upon moving in and moving out, so that I can avoid disputes over damages.
* As a **property owner**, I want to verify the condition of my property after a tenant moves out, so that I can assess any damages fairly.

\*assumption of property validity and no land commission issues or litigation issues( recommended path) or verification from Lands commission(complications of political interactions, future works implementation)

|  |  |
| --- | --- |
| **User Story** | **Priority** |
| **US-UMM-001 - User Registration and Authentication** | 1 |
| **US-UMM-002 - Role-Based Access Control (RBAC)** | 2 |
| **US-PMM-001 - Property Ownership Registration on Blockchain** | 3 |
| **US-PMM-003 - Ownership Verification via Smart Contracts** | 4 |
| **US-PMM-002 - Property Listing for Sale and Rental** | 5 |
| **US-PMM-004 - Property Ownership Transfer** | 6 |
| **US-RMM-001 - Rental Agreement via Smart Contracts** | 7 |
| **US-RMM-003 - Consensus-Based Rental Verification** | 8 |
| **US-RMM-004 - Property Return and Condition Verification** | 9 |
| **US-RMM-002 - Tenant Rental History Tracking** | 10 |
| **US-UMM-003 - Identity Verification for Property Owners** | 11 |
| **US-UMM-004 - Client Reputation and Review System** | 12 |

# **Non-Functional Requirements:**

1. NFR1: The system shall implement end-to-end encryption for data transmission and storage.
2. NFR2: The system shall support at least 100 concurrent users without performance degradation
3. NFR3: The system shall maintain a 99% uptime during peak usage hours.
4. NFR4: Execution of smart contracts shall be completed within 3(or 5) seconds on average.
5. NFR5: The user interface (UI) shall be intuitive and accessible to users with basic literacy skills.
6. NFR6: All rental and ownership records stored on the blockchain shall be immutable and tamper-proof.
7. NFR7: The system should comply with the necessary data protection laws. \*
8. NFR8: The User Interface shall respond to user input with 500ms under normal working conditions
9. NFR9: The system shall provide a log of all transactions and changes for audit purposes
10. NFR10: The system shall maintain a modular codebase for easier scalability.

|  |  |
| --- | --- |
| **Non-Functional Requirement** | **Availability Percentage** |
|  |  |
|  |  |

**SYSTEM DESIGN:**

**DB Models:**

User:

- firstname: str

- lastname: str

- email: str

- phone\_number: str

- password\_hash: str

- role: str [choices -> property\_owner, property\_buyer, land\_commission\_rep, sys\_admin]

- id\_type: str

- id\_value: str

- is\_verified: bool

- created\_at: datetime

- \* last\_login

-\*is\_active

Property:

-property\_id ( primary key..)

- title: str

-\*imageThis T

-\*property\_type( can be enum?\*)

- description: text

- location: str

- price: decimal

- status: str [choices -> available, rented, unlisted]

- created\_at: datetime

UserProperty:

- owner: User

- property: Property

- is\_verified: bool

- is\_active: bool

- transaction\_hash: text

PropertyDocument:

- user\_property: UserProperty

- attachment: file

-\*type( enum?)

-uploaded\_at : datetime

PropertyListing:

-\*created\_at

-\*expires\_at

- user\_property: UserProperty

- listing\_type: str [choices -> sale, rent]

- is\_active: bool

PropertyReviewRequest:

- property\_listing: PropertyListing

- requester: User

- status: str [choices -> requested, accepted, declined, reviewed]

- comment: text

PurchaseAgreement:

- user\_property: UserProperty

- renter: User

- start\_date: datetime

- end\_date: str

- transaction\_id: text

RentalReview:

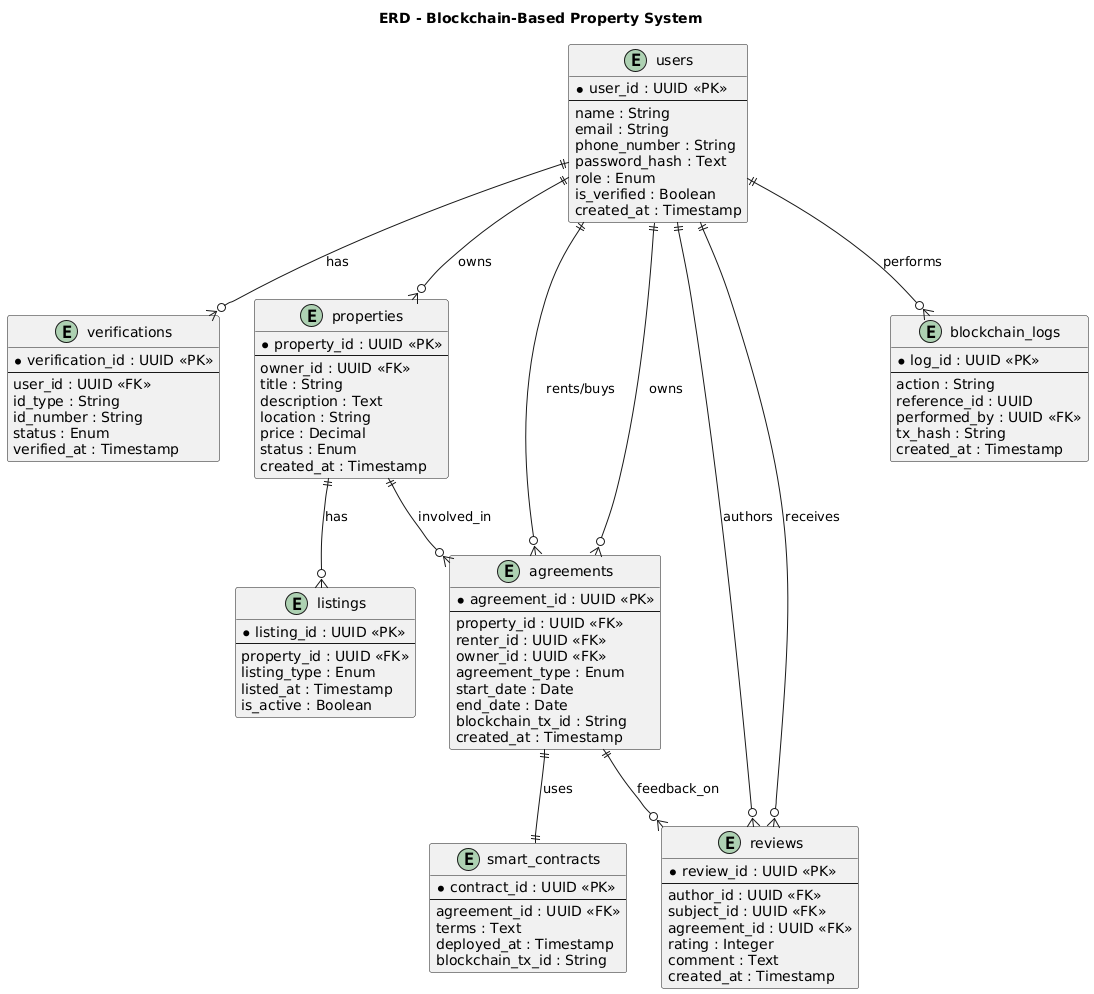
-created\_at

- agreement: PurchaseAgreement

- tenant\_review: text

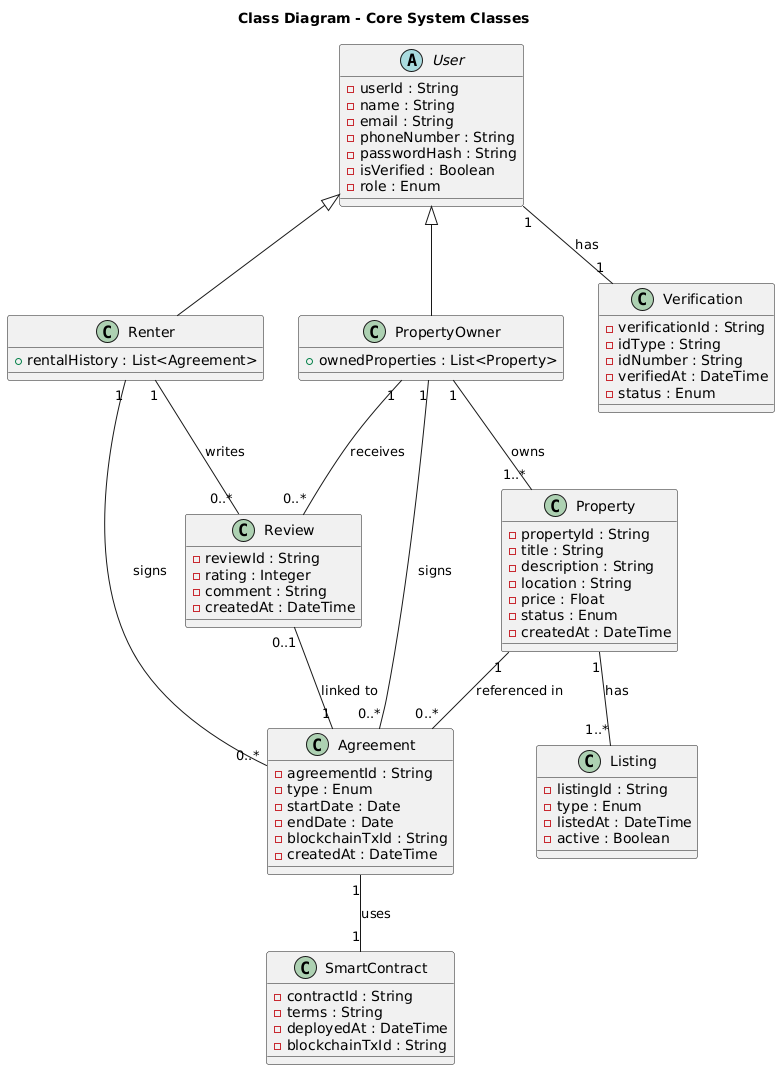
- landlord\_review: text

PS: A tenant can review a property within one month before or after their tenure ends.



* First name, last name atomicity for users table\*
* Simulate land commission rep for mock verification \* (single button to mark property verified)\*
* Renters are also buyers in the system
* Property status: available and unlisted
* Include a user-property table to prevent multiplicity between users and properties
* Includes is verified, is active Boolean in user-property table
* Include id type, id number and verified Boolean in the user table
* Property inspection review immediately after inspection
* Property table should also store picture of property

**Class Diagram:**

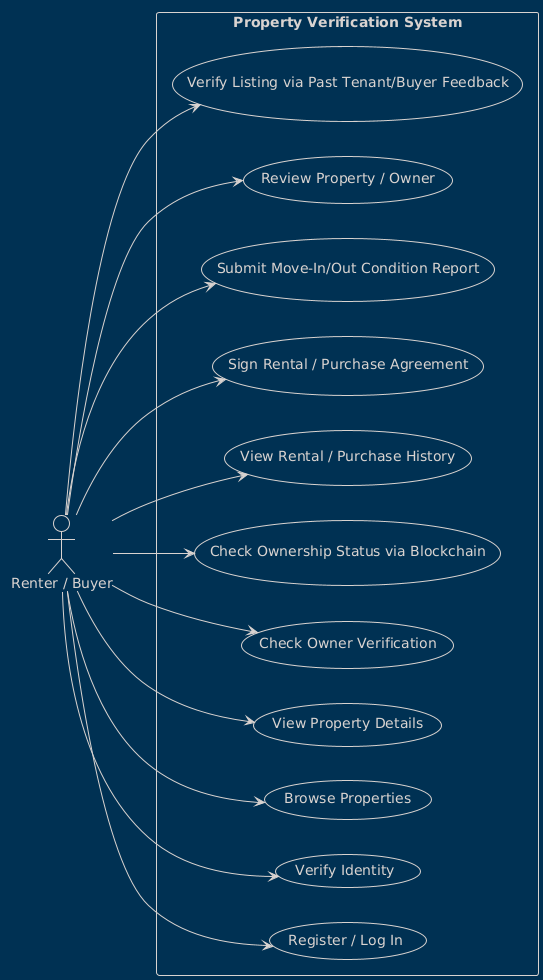
****

* + **Id type: dropdown of supported ids**
  + **Sysadmin – regex validation of id card type**
  + **\* future work: integration into actual ghana card API system**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Use Case Diagrams:**

**A diagram of property owner

AI-generated content may be incorrect.**

* + Evaluate the use case more to figure out what happens during the moving out process for renters\*

**PS:** Revisit post rental activities.

* Monolithic Architecture with db\*

-backend setup: PostgreSQL, Django framework for web

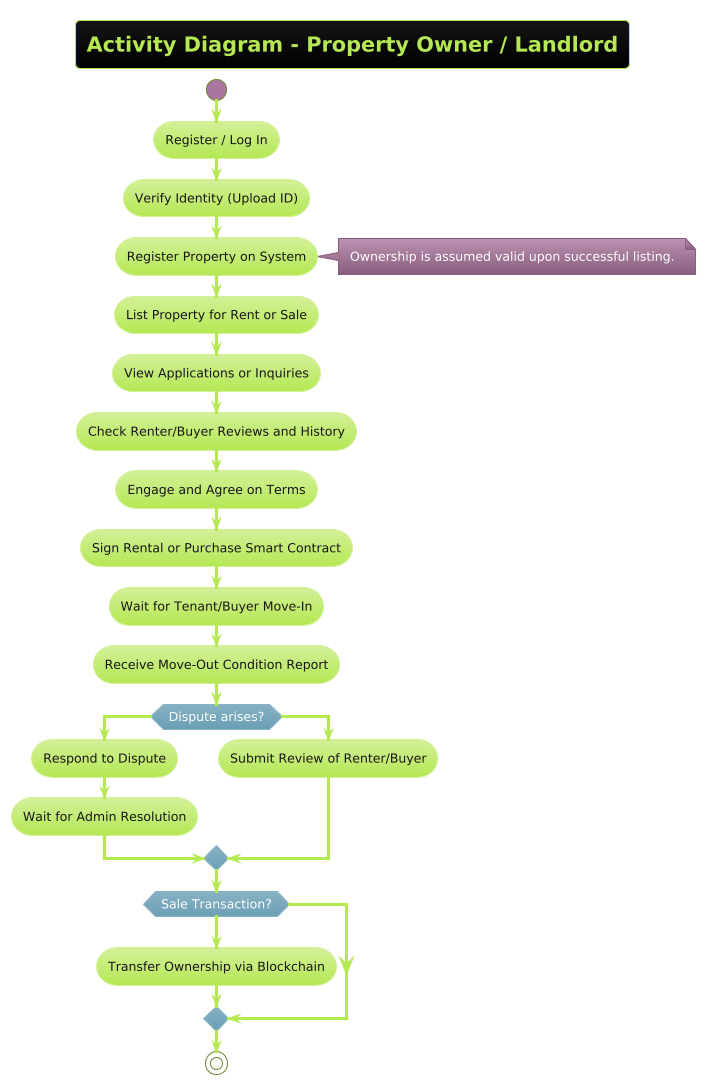
\frontend: Figma ( by tomorrow), react js for front

Add use case for sysadmin( verifying users ) and lands commission ( verify properties)

A screen shot of a chat

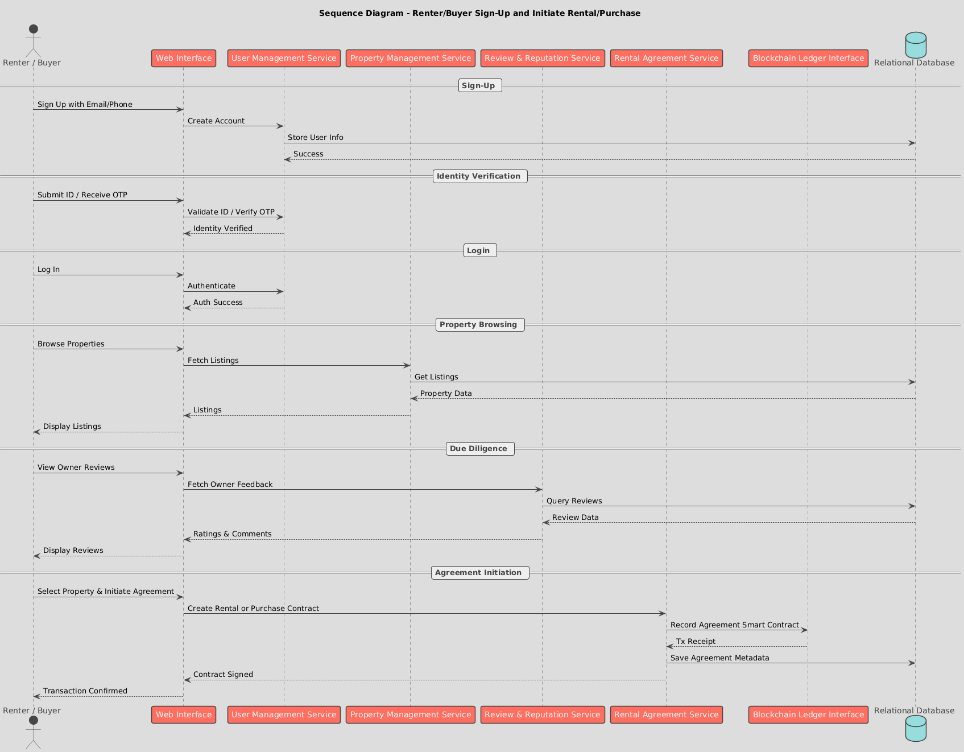
AI-generated content may be incorrect.

**Activity Diagrams**

**A flowchart of a company

AI-generated content may be incorrect.**

**Sequence Diagram:**



A diagram of a company

AI-generated content may be incorrect.

A screenshot of a diagram

AI-generated content may be incorrect.

Monolithic architecture with multi dbs –( multiple databases, spread out tables)

Agile development ( document sprints)

from django.http import JsonResponse

from django.views.decorators.csrf import csrf\_exempt

from django.core.exceptions import ValidationError

from django.contrib.auth.hashers import make\_password

import json

from .models import User

@csrf\_exempt

def register\_user(request):

    if request.method != 'POST':

        return JsonResponse({'error': 'Method not allowed'}, status=405)

    try:

        data = json.loads(request.body)

        # Validate required fields

        required\_fields = ['firstname', 'lastname', 'email', 'phone\_number', 'password', 'role', 'id\_type', 'id\_value']

        for field in required\_fields:

            if field not in data:

                return JsonResponse({'error': f'Missing required field: {field}'}, status=400)

        # Validate role

        if data['role'] not in dict(User.ROLE\_CHOICES):

            return JsonResponse({'error': 'Invalid role'}, status=400)

        # Check if email already exists

        if User.objects.filter(email=data['email']).exists():

            return JsonResponse({'error': 'Email already exists'}, status=400)

        # Create user with hashed password

        user = User.objects.create(

            firstname=data['firstname'],

            lastname=data['lastname'],

            email=data['email'],

            phone\_number=data['phone\_number'],

            password\_hash=make\_password(data['password']),  # Properly hash the password

            role=data['role'],

            id\_type=data['id\_type'],

            id\_value=data['id\_value'],

            is\_verified=False

        )

        return JsonResponse({

            'message': 'User registered successfully',

            'user\_id': user.id

        }, status=201)

    except json.JSONDecodeError:

        return JsonResponse({'error': 'Invalid JSON data'}, status=400)

    except ValidationError as e:

        return JsonResponse({'error': str(e)}, status=400)

    except Exception as e:

        return JsonResponse({'error': 'An unexpected error occurred'}, status=500)