Name: Pranav Raj Sowrirajan Balaji

Date: 3 December, 2024

Home Rental Management System

This system is designed to help property owners manage rental properties and tenants efficiently. It includes features for tracking property listings, tenant applications, lease agreements, payments, maintenance requests and tenant referrals. The Tenant Referral Program adds a layer of community engagement by allowing current tenants to refer friends or colleagues. If the referral leads to a lease, the referring tenant earns rewards, such as rent discounts or cash incentives. This feature encourages tenants to spread the word, improving property occupancy rates for property owners. ***Additionally, we will enhance the system with an in-memory key-value store (Redis) to optimize specific functionalities like frequently accessed data like (tenant referrals, active leases & maintenance requests).***

Rules

- A **property owner** can own multiple properties and list them on the website.
- A **property** can only be rented by one tenant at a time, but over time, it can have multiple tenants.
- A **tenant** can apply to rent multiple properties but can have only one active lease.
- Each **application** is associated with a specific tenant and property.
- A **lease agreement** connects a tenant and a property for a specific duration.
- Tenants must make **payments** for their rent, which are tracked for each lease.
- Tenants can submit **maintenance requests** for properties they rent.
- Tenants can refer other potential tenants and earn rewards if the referral leads to a lease.

Nouns & Verbs:

Entities

1. **Property Owner**: Represents a person or company that owns rental properties.

   Attributes:

   - propertyowner_ID
   - Name
   - Email
   - PhoneNumber
   - Address
   - TaxID

Verbs/actions: List

2. **Property**: Represents a rental property owned by a property owner.
    a. Attributes:
        - property_ID,
        - Address
        - City
        - State
        - ZipCode
        - PropertyType (e.g., Apartment, House)
        - NumberOfRooms
        - RentAmount
        - propertyowner_ID

Verbs/actions: List

3. **Tenant**: Represents a person renting a property.

Attributes:

- tenant_ID,
- Name
- Email
- PhoneNumber
- LeaseStartDate
- LeaseEndDate
- property_id

Verbs: Apply, Rent, submit

4. Application: Represents an application submitted by a tenant to rent a property. (Class serving as medium to split many to many relation from property - tenant UML diagram to the ERD diagram)

Attributes:

- application_ID,
- tenant_ID
- property_ID
- ApplicationDate,
- ApplicationStatus (e.g., Pending, Approved, Rejected).

Verbs: Apply

5. Lease Agreement: Represents the rental agreement between a property owner and a tenant.

Attributes:

- lease_ID
- property_ID
- tenant_ID
- LeaseStartDate
- LeaseEndDate
- RentAmount
- SecurityDepositAmount.

Verbs: connects, active

6. Payment: Represents a payment made by a tenant for rent.

Attributes:

- payment_ID
- tenant_ID
- lease_ID
- PaymentDate
- Amount
- PaymentStatus (e.g., Paid, Late)

7. Maintenance Request: Represents a maintenance request made by a tenant.

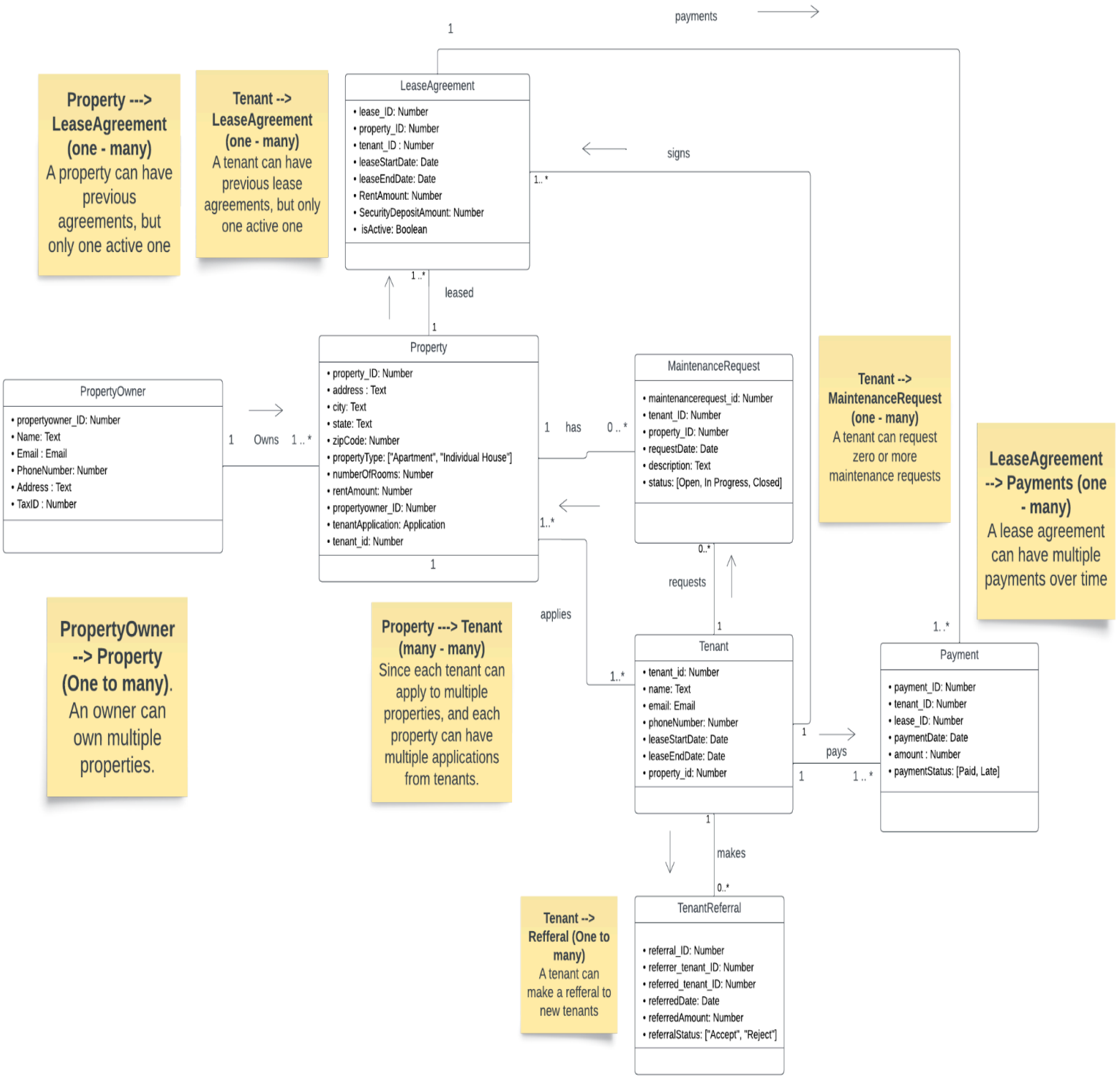
Attributes:

- request_ID
- tenant_ID
- property_ID
- RequestDate
- Description
- Status (e.g., Open, In Progress, Completed)

8. **TenantReferral**: Tenant can give referrals to potential clients / tenants.

- referral_ID
- referrer_tenant_ID
- referred_tenant_ID
- ReferralDate
- RewardAmount
- ReferralStatus (Accept, Reject)

Conceptual Model (UML):



Functionalities Using In-Memory Key-Value Storage

1. Frequently Accessed Data:

- Tenant referral program details (status, rewards).
- Active lease agreements for properties and tenants.

2. Real-Time Data Management:

- Tracking ongoing maintenance requests.

Redis Data Structures for Selected Functionalities

1. Tenant Referral Program

Redis Data Structure: **Hash**

Key Format: **tenantReferral:<referrerTenantID>**

Fields:

- referredTenantID -> String: Stores the ID of the tenant being referred.
- status -> String: Tracks the status of the referral (e.g., Accepted/Rejected).
- reward -> Number: Stores the reward amount (e.g., rent discount or cash incentive).

Reasoning*:*

1. A Hash is optimal because it allows us to store multiple fields (like status and reward) under a single key.
2. It provides efficient access to individual fields within a referral, enabling quick updates or reads (O(1) complexity for both).
3. This structure suits use cases where related data (e.g., status, referred tenant) needs to be grouped and frequently updated.

Example Use Case:

- When a referral is created, it is added as a new hash.
- The reward field can be updated after the referral is accepted, and the status can be changed as needed.

2. Active Lease Agreements

Redis Data Structure: **Sorted Set**

Key Format: **activeLease:<tenantID>**

Value: propertyID (ID of the property leased by the tenant).

Score: Unix timestamp representing the lease end date (e.g., 1704067200 for December 31, 2024).

Reasoning:

1. A Sorted Set is ideal for managing time-sensitive data because the elements can be sorted by their scores.
2. Lease agreements often need to be tracked by their end dates to manage expirations or renewals efficiently. The score (timestamp) allows for chronological queries.
3. Operations like finding expired leases (using ZREMRANGEBYSCORE) or fetching upcoming expirations (using ZRANGE) are efficient and straightforward with this structure.

Example Use Case:

- Add a lease to the sorted set when it becomes active.
- Remove expired leases using the score (timestamp) to ensure the dataset remains up-to-date.

3. Maintenance Requests

Redis Data Structure: **List**

Key Format: **maintenanceRequests:<propertyID>**

Value: JSON string containing request details (e.g., { "tenantID": "tenant789", "description": "Leaky faucet", "status": "Open" }).

Reasoning:

1. A List is chosen because maintenance requests are inherently sequential. Requests are usually processed in the order they are submitted.
2. The List structure supports operations like adding new requests to the end (RPUSH) and reading requests sequentially (LRANGE), which aligns with the typical usage pattern.
3. This structure is suitable for tracking updates (like marking a request as completed) and querying all requests for a specific property.

Example Use Case:

- Add new maintenance requests to the end of the list.
- Fetch all requests for a property to display the maintenance history or pending issues.

Redis Commands for CRUD Operations

1. Tenant Referral Program

Create: Initialize Referral

HSET tenantReferral:tenant789 referredTenantID tenant123 status Pending reward 0

Read: Get Referral Details

HGETALL tenantReferral:tenant789

Update: Change Referral Status and Reward

HSET tenantReferral:tenant789 status Accepted reward 200

Delete: Remove Referral

DEL tenantReferral:tenant789

2. Active Lease Agreements

Create: Add New Lease

ZADD activeLease:tenant789 1704067200 property456

Read: Get Active Leases

ZRANGE activeLease:tenant789 0 -1

Read: Get Active Leases with Scores (Lease End Dates)

ZRANGE activeLease:tenant789 0 -1 WITHSCORES

Update: Extend Lease Duration (Change Lease End Date)

ZADD activeLease:tenant789 1706659200 property456

Delete: Remove a Lease

ZREM activeLease:tenant789 property456

Delete: Remove Expired Leases

ZREMRANGEBYSCORE activeLease:tenant789 -inf 1698969600

3. Maintenance Requests

Create: Add New Maintenance Request

RPUSH maintenanceRequests:property456 '{"tenantID": "tenant789", "description": "Leaky faucet", "status": "Open"}'

Read: Get All Requests for a Property

LRANGE maintenanceRequests:property456 0 -1

Read: Get a Specific Request by Index

LINDEX maintenanceRequests:property456 0

Update: Change Request Status

LSET maintenanceRequests:property456 0 '{"tenantID": "tenant789", "description": "Leaky faucet", "status": "Completed"}'

Delete: Remove a Specific Request by Trimming the List

LTRIM maintenanceRequests:property456 1 -1

Delete: Remove All Requests for a Property

DEL maintenanceRequests:property456