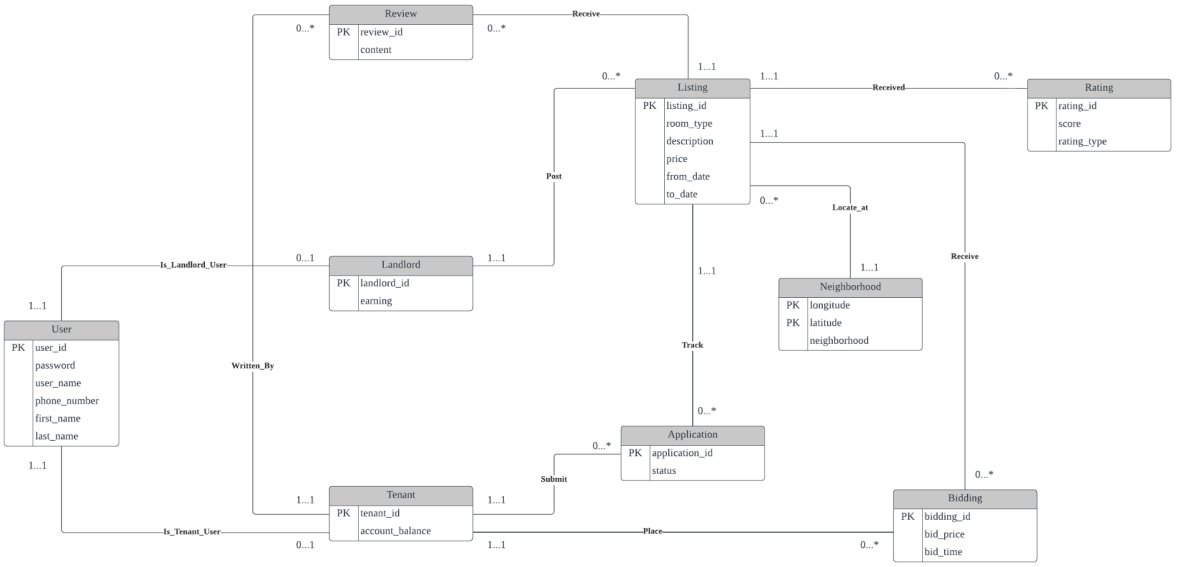
1 UML Diagram



2 Explanation of UML

2.1 Assumptions and Modeling Decisions:

- **User**: Represents individuals with unique identities. Each user is either a landlord or tenant, not both, to simplify user management.
- **Landlord**: This entity might have unique financial attributes such as earnings and functions related to property management. Having Landlord as a separate entity allows the system to provide specialized services and operations for landlord users without affecting other types of users.
- **Tenant**: Similarly, the Tenant entity has its financial attributes like account balance and functions such as searching for listings, applying for rentals, and bidding on properties. By separating this into a distinct entity, the system can cater to tenant-specific interactions.
- **Listing**: Independent entity due to its complex attributes and to ensure listings persist independently of landlord status.
- **Application**: Stands alone to track the status of rental applications, which involves multiple stages and outcomes.
- **Bidding**: Isolated entity to manage bid amounts and enforce rules such as increasing bid values and a maximum of three bids per listing.
- **Review and Rating**: Separated from listings to allow multiple instances per listing and detailed tenant feedback.

2.2 Relationships

User ↔ Landlord/Tenant

- **User to Landlord/Tenant**: A user is assumed to either be a landlord or a tenant (1-to-1).

- **Landlord/Tenant to User**: Each landlord or tenant is also a user, completing the reciprocal relationship (1-to-1).

Landlord ↔ Listing

- **Landlord to Listing**: Landlords are capable of posting multiple listings (1-to-many).
- **Listing to Landlord**: Each listing is tied to a single landlord (many-to-1).

Tenant ↔ Application

- **Tenant to Application**: A tenant can submit multiple applications for different listings (0-to-many).
- **Application to Tenant**: Each application is submitted by one tenant (many-to-1).

Tenant ↔ Bidding

- **Tenant to Bidding**: A tenant can place multiple bids or no bid, possibly on different listings (0-to-many).
- **Bidding to Tenant**: Each bid is placed by one tenant (many-to-1).

Listing ↔ Review

- **Listing to Review:** A listing can have multiple reviews or no review from different tenants (0-to-many).
- **Review to Listing:** Each review is associated with one listing (many-to-1).

Listing ↔ Rating

- **Listing to Rating:** A listing can have multiple ratings or no rating of different types from tenants (0-to-many).
- **Rating to Listing:** Each rating is associated with one listing (many-to-1).

Listing ↔ Application

- **Listing to Application:** A listing can have multiple applications from various tenants (1-to-many).
- **Application to Listing:** Each application is for one listing (many-to-1).

Listing ↔ Bidding

- **Listing to Bidding:** A listing can receive multiple bids or no bid (0-to-many).
- **Bidding to Listing:** Each bid is for one listing (many-to-1).

2.3 Additional Bidding Assumptions:

- The first bid must exceed the landlord's price
- subsequent bids must be higher than the last.

This competitive approach is typical in auction-like environments and ensures fair market value is achieved for listings.

4 Normalization with 3NF

We chose 3NF instead of BCNF decomposition to normalize our database because the cardinality of our main table (listing) is huge (8950) and the requirement of query performance is critical. At the same time, 3NF will help us end up with fewer tables compared to BCNF while maintaining manageable data integrity risks, so we decided to choose 3NF to normalize our database schema.

For tables other than Listing, all the non-key attributes directly depend on the primary key of that table, and there does not exist any transitive dependency among non-key attributes. Thus, we can say that these tables (User, Review, Rating, Bidding, Application) satisfy the Third Normal Form (3NF) requirements.
As for the Listing table, the address can determine the neighborhood, and the neighborhood can also be determined by the combination of longitude and latitude. The transitive dependency will cause data redundancy in our table. Therefore, we conduct the 3NF decomposition in this table as follows:

Step 1: Use character to represent each attribute in the original Listing table shown as follow. To simplify the question, we use one character “B” to represent all the attributes that do not have any transitive dependency with other attributes.

Listing	
A	listing_id
B	room_type
C	neighborhood
D	longitude
E	latitude
B	description
B	price
B	from_date
B	to_date

Step 2: Find out all the functional dependencies in this table.
FDs = {A -> BCDE, DE -> C}

Step 3: Find out the candidate key for the FDs.

LEFT	MIDDLE	RIGHT	NONE
A	D, E	B, C	

Since A+ = {A, B, C, D, E}, A (listing_id) is the candidate key for this table.

Step 4: Find the minimal basis for FDs.
(1) A -> BCDE ==> A -> B, A -> C, A -> D, A -> E
(2) DE -> C. No unnecessary attributes in LHS. Keep this relationship.
(3) FDs = {A -> B, A -> C, A -> D, A -> E, DE -> C}
FDs’ = {A -> B, A -> D, A -> E, DE -> C}

Step 5: Get normalized tables that conform to 3NF:
(A, B, D, E), (D, E, C)

According to the 3NF decomposition, we split the original Listing table into two tables. One is Listing (listing_id, room_type, description, price, from_data, to_data) and the other is Neighborhood (longitude, latitude, neighborhood).

5 Logical Design (Relational Schema)

1. User(

user_id: int [PK],

password: varchar(255),

user_name: varchar(255),

phone_number: int,

first_name: varchar(255),

last_name: varchar(255)

)

Table: User

Field	Type	Key	Constraint	Extra
user_id	int	PK	Not Null	
password	varchar(255)		Not Null	
user_name	varchar(255)		Not Null	
phone_number	int		Not Null	
first_name	varchar(255)			
last_name	varchar(255)			

2. Landlord(

landlord_id: int [PK],

earning: int,

user_id: int [FK to User.user_id]

)

Table: Landlord

Field	Type	Key	Constraint	Extra
landlord_id	int	PK	Not Null	

earning	int		Default 0	
user_id	int	FK	Not Null	

3. Tenant(

tenant_id: int [PK],

account_balance: int,

user_id: int [FK to User.user_id]

)

Table: Tenant

Field	Type	Key	Constraint	Extra
tenant_id	int	PK	Not Null	
account_balance	int		Default 0	
user_id	int	FK	Not Null	

4.Listing(

listing_id: int [PK],

room_type: int,

description: varchar(255),

price: int,

from_date: date,

to_date: date,

landlord_id: int [FK to Landlord.landlord_id],

longitude: decimal [FK to Neighborhood.longitude],

latitude: decimal [FK to Neighborhood.latitude],

)

Table: Listing

Field	Type	Key	Constraint	Extra
listing_id	int	PK	Not Null	
room_type	int			0-Shared room; 1-Entire home/apt
description	varchar(255)			
price	int		Not Null	
from_date	date			
to_date	date			
landlord_id	int	FK	Not Null	
longitude	decimal	FK		
latitude	decimal	FK		

5. Application(

application_id: int [PK],

status: varchar(255),

listing_id: int [FK to Listing.listing_id],

tenant_id: int [FK to Tenant.tenant_id]

)

Table: Application

Field	Type	Key	Constraint	Extra
application_id	int	PK	Not Null	
status	varchar(255)			
listing_id	int	FK	Not Null	
tenant_id	int	FK	Not Null	

6. Bidding(

bidding_id: int [PK],

bid_price: int,

listing_id: int [FK to Listing.listing_id],

tenant_id: int [FK to Tenant.tenant_id]

)

Table: Bidding

Field	Type	Key	Constraint	Extra
bidding_id	int	PK	Not Null	
bid_price	int			
bid_time	DateTime			
listing_id	int	FK	Not Null	
tenant_id	int	FK	Not Null	

7. Rating(

rating_id: int [PK],

score: decimal,

rating_type: int,

listing_id: int [FK to Listing.listing_id]

)

Table: Rating

Field	Type	Key	Constraint	Extra
rating_id	int	PK	Not Null	
score	decimal		Default 0	
rating_type	int		Not Null	0-overall rating ;1-accuracy

				rating; 2-cleanliness rating; 3-checkin rating; 4-communication rating; 5-location rating; 6-value rating
listing_id	int	FK	Not Null	

8.Neighborhood(

longitude: decimal [PK],

latitude: decimal [PK],

neighborhood: varchar(255)

)

Table: Neighborhood

Field	Type	Key	Constraint	Extra
longitude	decimal	PK		
latitude	decimal	PK		
neighborhood	varchar(255)			

9. Review(

review_id: int [PK],

content: TEXT,

tenant_id: int [FK to Tenant.tenant_id] ,

listing_id: int [FK to Listing.listing_id]

)

Table: Review

Field	Type	Key	Constraint	Extra
review_id	int	PK	Not Null	
content	TEXT			
tenant_id	int	FK	Not Null	
listing_id	int	FK	Not Null	