WORKSHOP 1

AUTHOR:

LUIS ALEJANDRO MORALES

20222020175

DATABASE FOUNDATIONS

PROFESSOR:

CARLOS ANDRÉS SIERRA VIRGÜEZ

UNIVERSIDAD DISTRITAL FRANCISCO JOSE DE CALDAS

BOGOTA DC

SEPTIEMBRE 2024

WORKSHOP 1

Survey of 15 students Question: What features would you like to see in an app for an apartment complex?

1. **Student responses:**

- Access to the list of blocks and apartments.
- Possibility of making online payments for management services.
- Reserve common areas such as the gym, pool and event room.
- Receive notifications about events or maintenance in the building.
- System for reporting problems or maintenance requests.
- View payment history and outstanding debts.
- Controlled access to security cameras in common areas.
- View announcements from the administrator or the board of directors.
- Visitor management and access request.
- Access to mail or package delivery services.
- Calendar of complex activities, such as yoga classes or meetings.
- A chat to communicate with management or between neighbors.
- Utility consumption report.
- Information on recycling and rules of coexistence.
- Option to rent common areas to other neighbors.

2. **User Histories:**

- As a <resident>, I want to <see a list of all blocks and apartments>, so <I can identify my home and the surrounding areas>.
- As a <resident>, I want to <make payments online for administration services>, so <I can stay up to date with my fees in a convenient way>.
- As a <resident>, I want to <reserve common areas like the gym or the pool>, so <I can ensure their availability when I need them>.
- As a <resident>, I want to <receive notifications about events and maintenance>, so <I can stay informed about what's happening in the apartment complex>.
- As a <resident>, I want to <report maintenance issues>, so <the administration can address them promptly>.
- As a <resident>, I want to <view my payment history and outstanding debts>, so <I can keep track of my financial obligations>.
- As a <resident>, I want to <access security camera footage in common areas>, so <I can monitor the surroundings and feel more secure>.

- As a <resident>, I want to <manage visitor access and receive notifications when they arrive>, so <I can ensure safe and easy entry for my guests>.
- As a <resident>, I want to <chat with other neighbors or the administration through the app>, so <I can communicate more easily with the community>.
- As a <resident>, I want to <receive and manage my mail and packages>, so <I can be notified when they arrive and pick them up on time>. By

3. Technical and Design Considerations

- Scalability: The database must be able to handle large volumes of data from residents, payments, and reservations.
- Security: Implement encryption for personal data and financial transactions. Access to certain functionalities should be restricted by roles.
- Availability: The application should be accessible from any device, ensuring a seamless experience for residents.
- Integration: Integrate with external services to process payments and send notifications in real time.
- Performance: Optimize database queries, especially for handling long lists of apartments, transactions, and reservations.

4. Database Design

- **Define Components:**

The main components of the system include resident management, apartments, payments, common area reservations, maintenance, notifications and visits. These components are the key functionalities that must be covered by the database.

- **Define Entities:**

The main entities needed to cover the functionalities are:

- Resident
- Apartment
- Block
- Payment
- Reservation
- Common Area
- Maintenance
- Notification
- Visitor
- Correspondence

- **Define Attributes per Entity**

Each entity has the following attributes:

  - Resident: ID, first name, last name, apartment number, block, email, phone.
  - Apartment: ID, number, block, size, current occupancy.
  - Block: ID, block name or number.
  - Payment: ID, amount, date, resident, payment method.
  - Reservation: ID, common area, date, time, resident.
  - Common Area: ID, name, maximum capacity.
  - Maintenance: ID, problem description, status (pending, in process, resolved), report date, resident.
  - Notification: ID, message, date, type (event, maintenance), resident.
  - Visitor: ID, name, associated resident, visit date.
  - Correspondence: ID, package type, arrival date, resident.

- **Define Relationships**

The relationships between entities in the database are defined as follows:

- Resident - Apartment: A Resident is associated with an Apartment. This is a one-to-one relationship since a resident can only live in one apartment at a time, and each apartment can be occupied by one resident (though over time, the same apartment may be occupied by different residents).
- Resident - Block: A Resident belongs to an Apartment, which is part of a Block. This is a many-to-one relationship, where many residents live in the same block.
- Resident - Payment: A Resident makes multiple Payments. This is a one-to-many relationship since a resident can make multiple payments over time.
- Resident - Reservation: A Resident can make multiple Reservations for Common Areas (like gym, pool, etc.). This is also a one-to-many relationship since each resident can make many reservations.
- Resident - Maintenance: A Resident can report multiple Maintenance issues. This is another one-to-many relationship since a resident may report several issues.
- Resident - Notification: A Resident can receive multiple Notifications. This is a one-to-many relationship, where a resident may get notifications regarding events, maintenance, or other updates.
- Resident - Visitor: A Resident can host multiple Visitors. This is a one-to-many relationship because one resident can have various visitors over time.

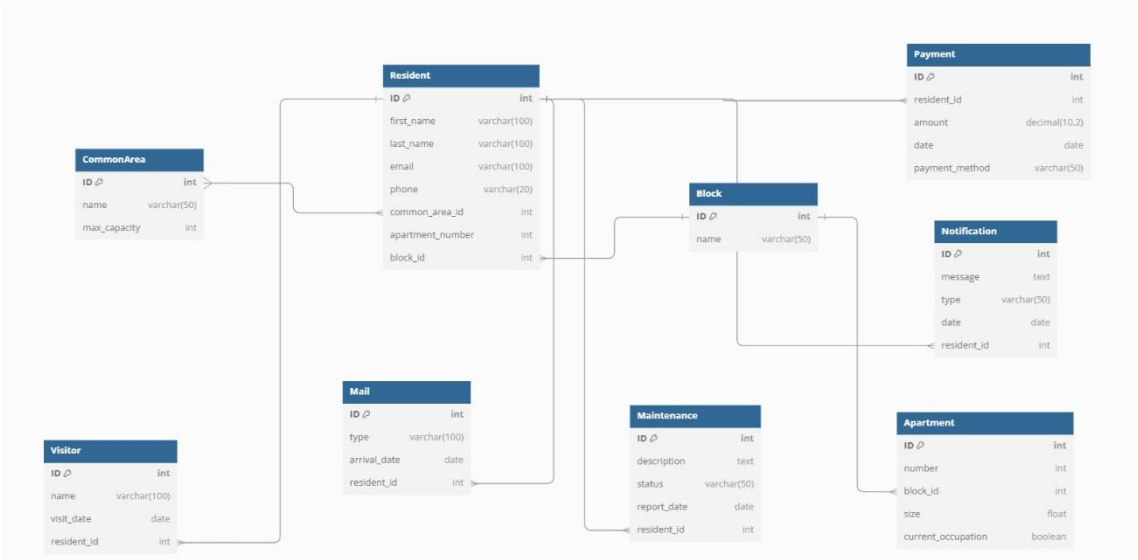
- Resident - Mail: A Resident can receive multiple pieces of Mail (or packages). This is a one-to-many relationship.
- Block - Apartment: Each Block has multiple Apartments. This is a one-to-many relationship since one block contains many apartments.
- Common Area - Reservation: A Common Area (such as a gym or pool) can be reserved by multiple residents, and each resident can make multiple reservations. This is a many-to-many relationship

- **Define Relationship Types**

The relationship types based on cardinality are:

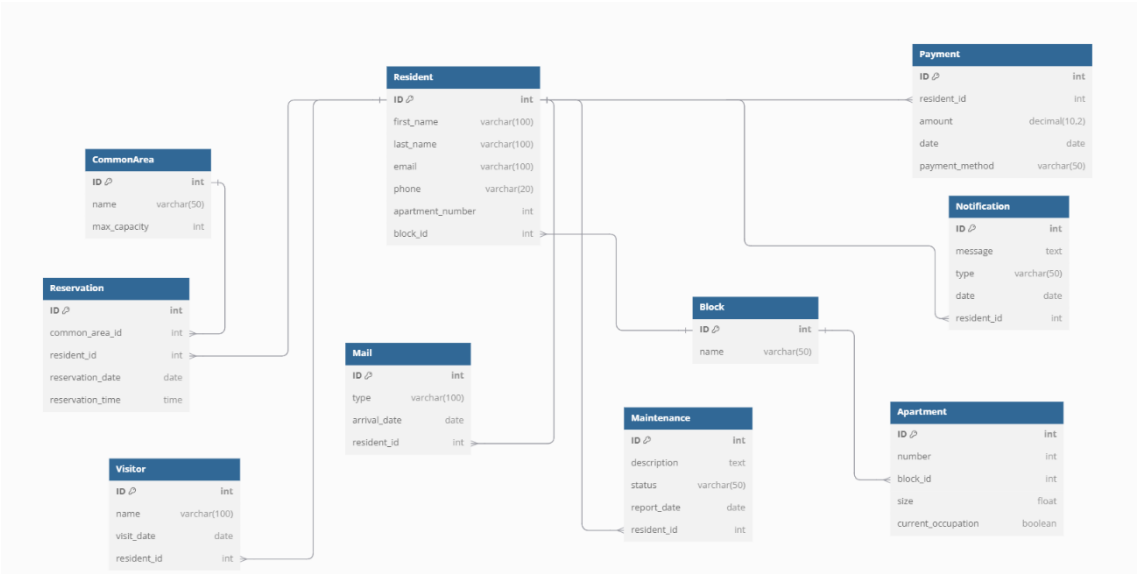
- One-to-One (1:1):
  - Resident - Apartment: Each Resident is linked to exactly one Apartment, and each Apartment is occupied by exactly one Resident at any given time.
- One-to-Many (1:N):
  - Block - Apartment: One Block can contain multiple Apartments.
  - Resident - Payment: One Resident can make many Payments.
  - Resident - Reservation: One Resident can make many Reservations.
  - Resident - Maintenance: One Resident can report multiple Maintenance issues.
  - Resident - Notification: One Resident can receive many Notifications.
  - Resident - Visitor: One Resident can host many Visitors.
  - Resident - Mail: One Resident can receive multiple pieces of Mail.
  - Many-to-One (N:1):
  - Apartment - Block: Multiple Apartments are part of a single Block.
- Many-to-Many (M:N):
  - Common Area - Reservation - Resident: Many Residents can reserve many Common Areas.

- **First Entity-Relationship Draw**



- **First Split Many-to-Many Relationship**
  - Many-to-Many (M:N):
    - Common Area - Reservation - Resident: Many Residents can reserve many Common Areas. This many-to-many relationship is handled using the Reservation table as a junction between Resident and Common Area.

- **Second Entity-Relationship Draw / E-R-M**



- **Constraints and Properties of Data**
  - Resident Table
    - ID: Primary Key, auto-increment, non-null.
    - first_name: Non-null, maximum length of 100 characters.
    - last_name: Non-null, maximum length of 100 characters.

    - email: Non-null, unique, must follow a valid email format.
    - phone: Non-null, maximum length of 20 characters, should follow a standard phone number format.
    - apartment_number: Non-null, integer that must exist in the Apartment table.
    - block_id: Foreign Key, references Block.ID, non-null.
- Apartment Table
    - ID: Primary Key, auto-increment, non-null.
    - number: Non-null, unique within the same block_id.
    - block_id: Foreign Key, references Block.ID, non-null.
    - size: Non-null, positive float value representing the size of the apartment in square meters.
    - current_occupation: Non-null, boolean value indicating whether the apartment is currently occupied.

- Block Table
    - ID: Primary Key, auto-increment, non-null.
    - name: Non-null, unique, maximum length of 50 characters.
- Payment Table
    - ID: Primary Key, auto-increment, non-null.
    - resident_id: Foreign Key, references Resident.ID, non-null.
    - amount: Non-null, positive decimal value representing the payment amount.
    - date: Non-null, valid date when the payment was made.
    - payment_method: Non-null, maximum length of 50 characters, should be one of the predefined methods (e.g., 'credit card', 'cash').
- Reservation Table
    - ID: Primary Key, auto-increment, non-null.
    - common_area_id: Foreign Key, references CommonArea.ID, non-null.
    - resident_id: Foreign Key, references Resident.ID, non-null.
    - reservation_date: Non-null, valid date for the reservation.
    - reservation_time: Non-null, valid time for the reservation.
- Common Area Table
    - ID: Primary Key, auto-increment, non-null.
    - name: Non-null, unique, maximum length of 50 characters.
    - max_capacity: Non-null, positive integer representing the maximum number of people allowed.
- Maintenance Table
    - ID: Primary Key, auto-increment, non-null.
    - description: Non-null, text field describing the maintenance issue.
    - status: Non-null, should be one of the predefined statuses (e.g., 'pending', 'in progress', 'resolved').
    - report_date: Non-null, valid date when the issue was reported.

    - resident_id: Foreign Key, references Resident.ID, non-null.
- Notification Table
    - ID: Primary Key, auto-increment, non-null.
    - message: Non-null, text field with a description of the notification.
    - type: Non-null, should be one of the predefined types (e.g., 'event', 'maintenance').
    - date: Non-null, valid date when the notification was issued.
    - resident_id: Foreign Key, references Resident.ID, non-null.
- Visitor Table
    - ID: Primary Key, auto-increment, non-null.
    - name: Non-null, maximum length of 100 characters.
    - visit_date: Non-null, valid date of the visit.
    - resident_id: Foreign Key, references Resident.ID, non-null.

- Mail Table
    - ID: Primary Key, auto-increment, non-null.
    - type: Non-null, maximum length of 100 characters describing the type of mail or package.
    - arrival_date: Non-null, valid date when the mail arrived.
    - resident_id: Foreign Key, references Resident.ID, non-null.