Apartment Leasing Management System

Final Report: 05.03.2024

Umar Abdul Aziz | Shriya Jaknalli | Aashlesha Voditelwar
CSCI 5333 Database Management System
Section-1, Spring 2024
University of Houston-Clear Lake, Houston, TX 77058

Table of Contents

Overview

The Apartment Management System is a robust, secure and user-friendly Database Management System (DBMS) project that will be developed to automate and make it easier for users and landlords to manage various apartment management activities such as leasing, renewing, etc. Tenants, owner, administrators, and employees of the apartment administration will all benefit from our project. Tasks like viewing tenant and owner information, making new profiles for owner, visitors, and residents, allocating parking spaces to both parties, and handling complaints and raising tickets for any kind of problem may all be handled effectively by administrators. Tenants, on the other hand, benefit from functionalities like viewing allotted parking slots, making maintenance fee payments, raising, and tracking complaints, and accessing personal details.

Additionally, the system guarantees a smooth experience for staff members, who can easily view all complaints and provide a combined count of all complaints. The system's focus on security and user authentication is essential in guaranteeing that all parties involved—Admins, Owner, Tenants, and Employees—have personalized and safe access to the platform. In general, the Apartment Management System acts as a single point of contact for managing complaints, retrieving information, and facilitating user interaction, all of which improve the effectiveness and transparency of apartment management procedures.

It is like a one stop solution for all the parties involved in apartment leasing, administrating, and renting the apartments.

Introduction/The detailed description of the system

Apartment Leasing Management System (ALMS) is built to cater to the specific data requirements essential for effective apartment rental management. Below is a detailed description of the key components and functionalities of the system:

Owners Management:

- **Identity and Contact Information: ALMS stores comprehensive details of apartment owners, including their identities and contact information.**
- **Owned Apartments: The system maintains a record of apartments owned by each landlord, facilitating efficient tracking and management.**

Tenants Management:

- **Identity and Contact Details: ALMS stores personal information and contact details of tenants for communication and identification purposes.**
- **Leasing Details: The system records the leased apartment details, including lease durations, rental amounts, and payment details.**
- **Age: Tenant age information is stored to ensure compliance with legal regulations.**

Apartments Management:

- **Unit Details: ALMS maintains detailed information about apartment units, including area, floor (1st floor, 2nd Floor), type (e.g., studio, one-bedroom), and block where the apartment is located (e.g. Block A, Block B).**

Payments Management:

- **Financial Transactions: ALMS keeps a comprehensive record of all financial transactions related to rent payments, security deposits, and other fees.**

Employees Management:

- **Staff Information: The system stores details of employees involved in apartment management, including their roles (eg. Admin, Maintenance Staff), contact information, and working hours.**

Parking Management:

- **Parking Spot Details: ALMS maintains information about parking spots within the complex and their allocation to specific tenants.**

Complaints Management:

- **Tracking and Resolution: The system provides functionality for tenants to log complaints, which are then tracked by Admin and Maintenance staff and resolved by the Maintenance staff.**

Apartment Applications Management:

- **Rental Application Handling: ALMS facilitates the management of rental applications from prospective tenants, including application details applied by the tenants.**

Blocks Management:

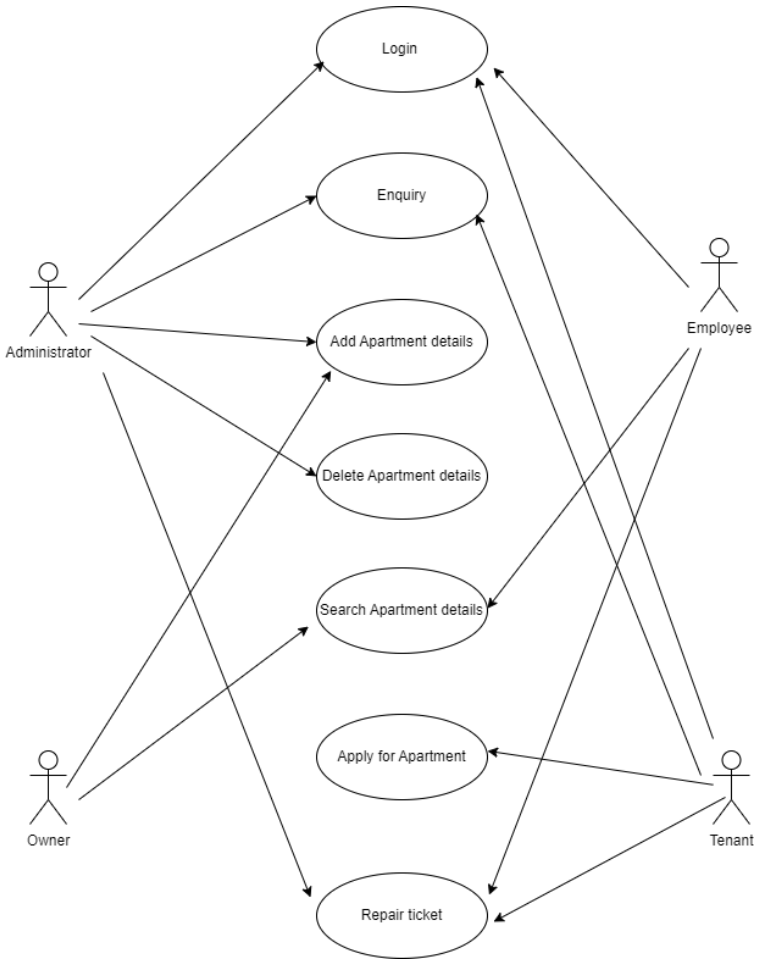
- **Organizational Structure: Information regarding different blocks within the apartment complex is stored for organizational purposes and efficient management.**

Access Control:

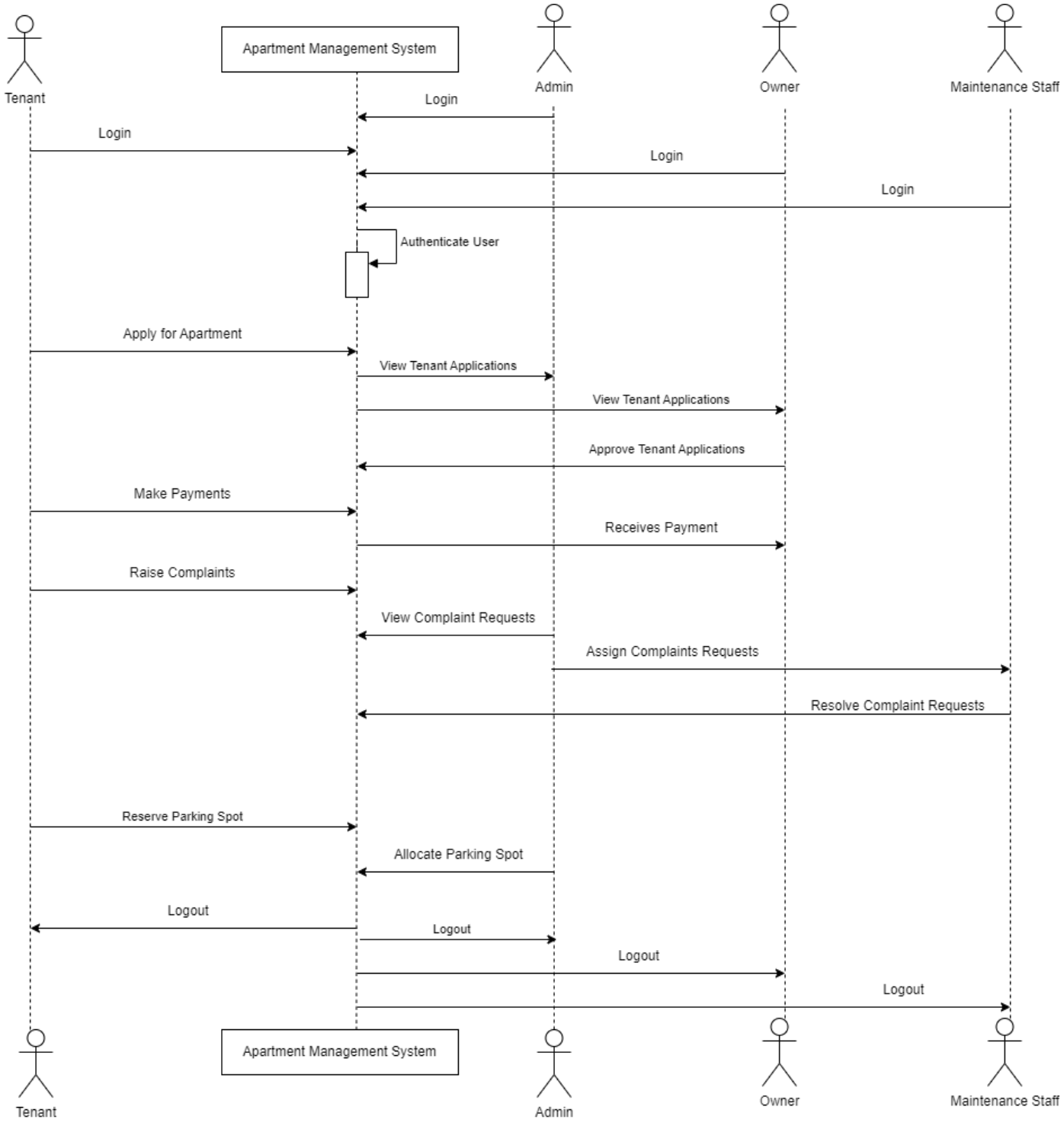
- **User Login Information: ALMS implements robust access control mechanisms, ensuring secure system access for owners, admin, tenants and employees by means of registration and login credentials.**

UML use-case/sequence/activity diagrams of the system

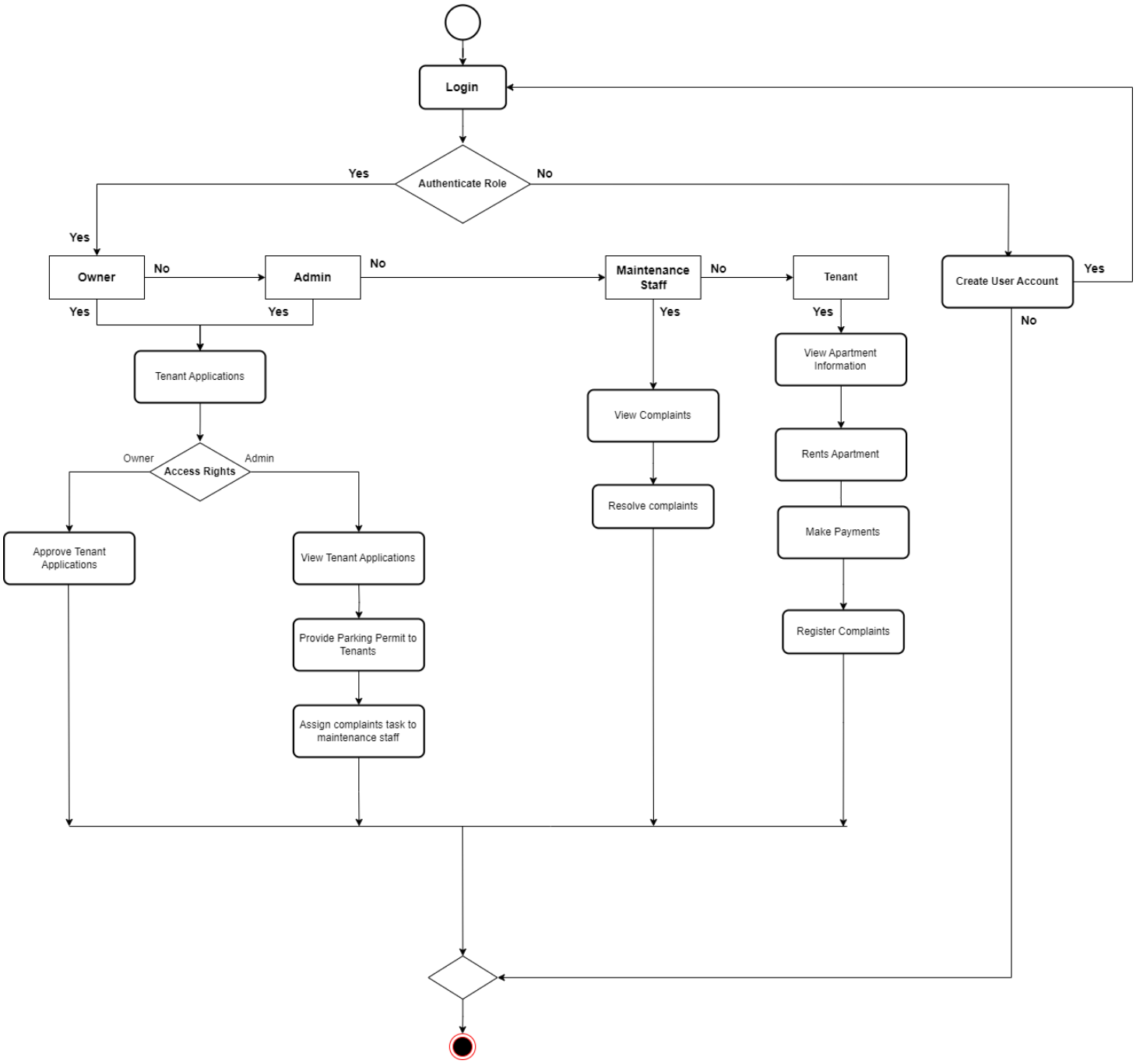
UML Use case diagram



UML Sequence diagram



UML Activity Diagram



Description of the overall system's data requirements

The system is designed to manage apartment rentals, which involves several key operations such as tracking apartment ownership, tenant leasing, payments, and maintenance activities. To facilitate these operations, the system requires data related to:

- Owners: Identity, contact information, and apartments owned.
- Tenants: Identity, contact details, age, and leasing details including payments.
- Apartments: Details about apartment units, including size, type, and location.
- Payments: Details of financial transactions related to rent and other fees.
- Employees: Information on staff members, including their roles and working hours.
- Parking: Details on parking spots and their allocation to tenants.
- Complaints: Tracking and resolution of tenant complaints.
- Apartment Applications: Management of rental applications from prospective tenants.
- Blocks: Information regarding the different blocks within the complex for organizational purposes.
- Access Control: User login information for secure system access by tenants and employees.

Entity-set designation

- Owner

  Attributes: Owner ID, Name, SSN, Address, Phone Number, Address, Email.

  Description: Represents the individuals or entities that own apartments.
- Apartment

  Attributes: Apartment Number (Apt No), Block ID, Bedrooms, Type, Area, Floor, Address, Owner_id

  Description: Represents individual apartment units available for rent.
- Tenant

  Attributes: Tenant ID, Name, SSN, Age, Permanent Address, Phone, Apt No (leased apartment), Email

  Description: Represents individuals who rent or are seeking to rent apartments.

- **Admin**

  Attributes: Employee ID (Emp ID), Name, Phone, Shift Timings, Authorization Type, Email

  Description: Represents administrative employees working for the apartment complex

- **Maintenance Staff**

  Attributes: Employee ID (Emp ID), Name, Phone, Shift Timings, Contract Length, Role (e.g. Technician), Email

  Description: Represents administrative employees working for the apartment complex

- **Apartment Application**

  Attributes: Name, Email, Phone Number, Apt No, Owner Id

  Description: Represents applications submitted by potential tenants for renting apartments.

- **Complaint**

  Attributes: Complaint ID, Complaint Description, Date, Emp_ID (representing the maintenance staff assigned), Tenant_id (tenant who filed the complaint)

  Description: Represents complaints made by tenants and complaint details.

- **Block**

  Attributes: Block ID, Block Name, Address

  Description: Represents a group or building within the apartment complex such as Block A, Block B.

- **Parking**

  Attributes: Spot Number, Type, Block Id, Tenant_id

  Description: Represents parking spots for the respective block available for tenants or visitors.

- **Login**

  Attributes: Email, Password

  Description: Represents the credentials required for users to access the system.

Relationship-set designation

- **Receives (between Owner and Payment)**
  Cardinality: One-to-Many
  Description: Indicates that an owner can receive many payments.

- **Approves (between Owner and Apartment Application)**
  Cardinality: Many-to-Many
  Description: Indicates that an owner can approve many applications and an application can be approved by many owners.

- **Makes (between Tenant and Payment, and between Tenant and Complaint)**
  Cardinality: Many-to-Many
  Description: Indicates that a tenant can make many payments and many complaints.

- **Owns (between Owner and Apartment)**
  Cardinality: Many-to-Many
  Description: Indicates that an owner can own many apartments and an apartment can have many owners.

- **Rents (between Tenant and Apartment)**
  Cardinality: Many-to-Many
  Description: Indicates that a tenant can rent many apartments and an apartment can be rented by many tenants.

- **Leases (between Tenant and Parking)**
  Cardinality: One-to-Many
  Description: Indicates that a tenant leases one or more parking spots.

- **Manages (between Apartment Employee and Apartment)**
  Cardinality: Many-to-Many
  Description: Indicates that an employee can manage many apartments and an apartment can be managed by many employees.

- **Handles (between Apartment Employee and Complaint)**
  Cardinality: Many-to-Many
  Description: Indicates that an employee can handle many complaints and a complaint can be handled by many employees.

- **Consist of (between Apartment and Block)**
  Cardinality: Many-to-One
  Description: Indicates that many apartments make up a block.

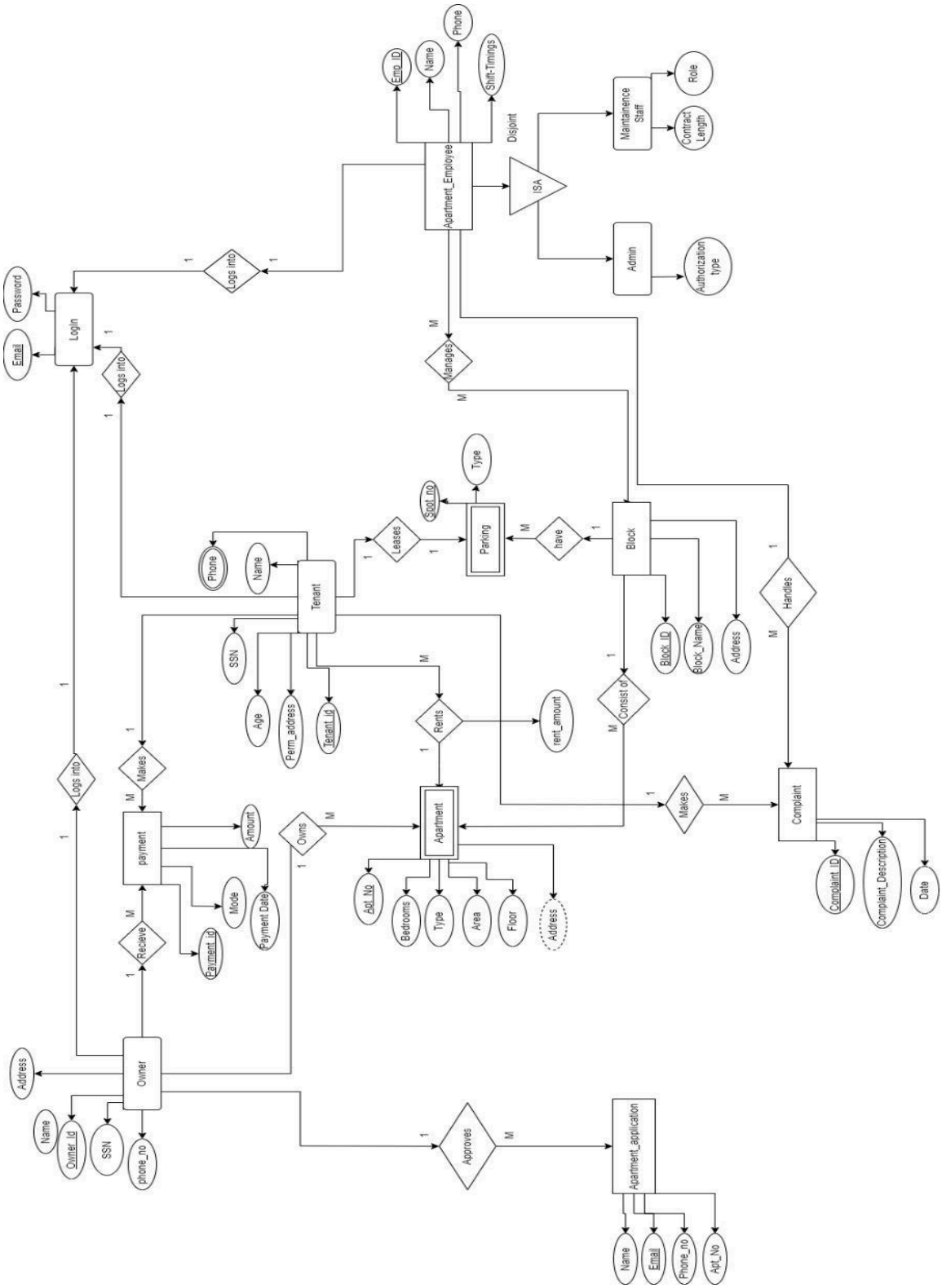
- **Logs into (between Tenant, Apartment Employee and Login)**
  Cardinality: One-to-One
  Description: Indicates that a tenant or an employee log into the system using one set of login credentials.

- **Have (between Parking and Block)**
  Cardinality: Many-to-Many
  Description: Indicates that a block can have many parking spots and a parking spot can belong to many blocks.

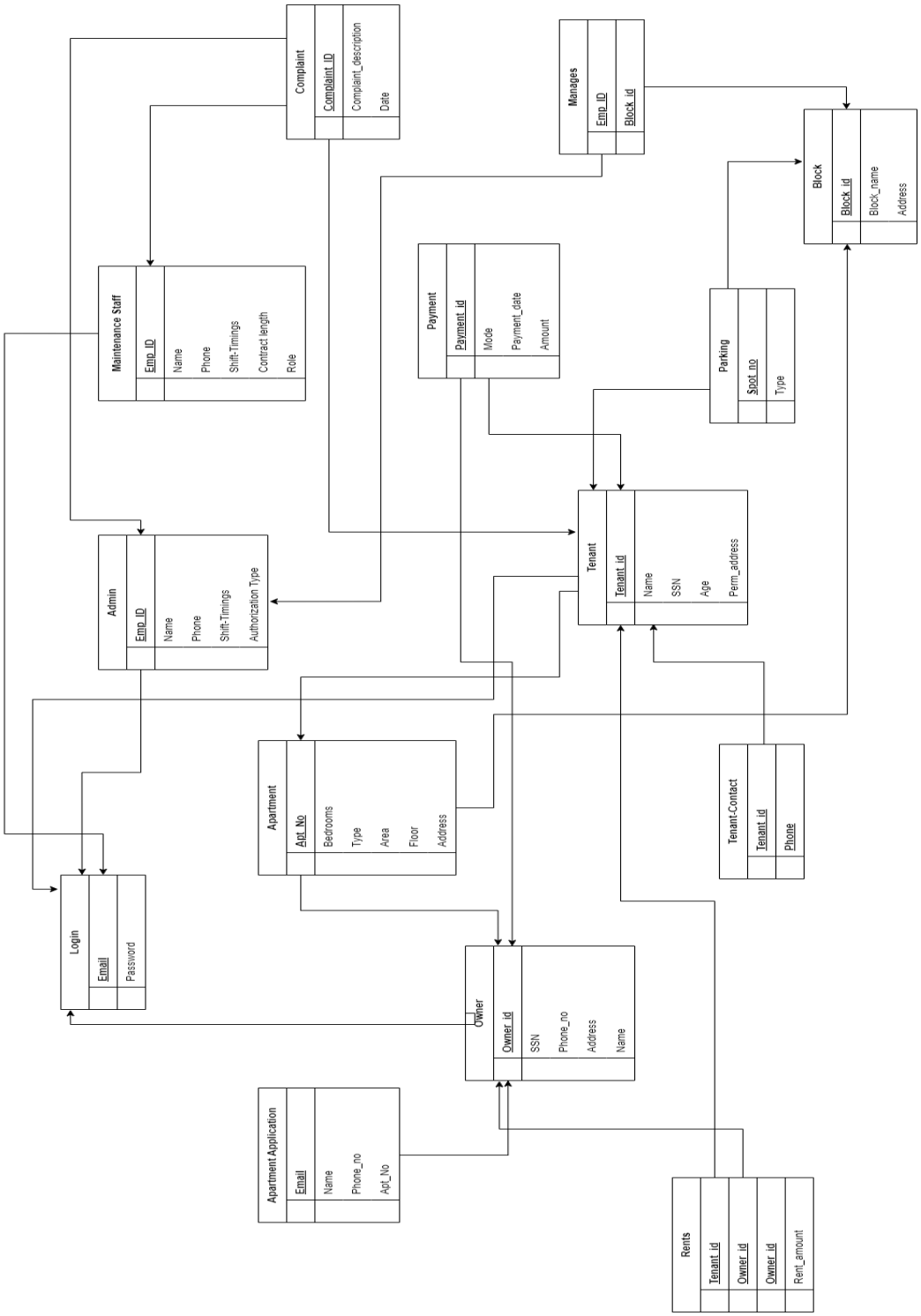
E-R Diagram with relevant assumptions

Assumptions

- **All entities have total participation.**
- **Owner's phone number is a single-valued attribute since we want to avoid giving personal numbers.**
- **Multiple tenants can share a single apartment.**
- **One login per person only**
- **Owners can receive multiple payments for multiple apartments.**
- **1 tenant can reserve 1 parking spot only.**
- **One apartment can accommodate multiple tenants hence we need multiple phone numbers.**


Emp_ID
Name
Phone
Shift-Timings
Apartment_Employee
Disjoint
ISA
Maintainence Staff
Role
Contract Length
Admin
Authorization type
Logs into
Password
Email
LogIn
Manages
Type
Slot_no
Parking
Leases
have
Block
Phone
Name
Tenant
Block_ID
Block_Name
Address
Handles
SSN
Age
Perm_address
Tenant_Id
Rents
rent_amount
Consist of
Makes
payment
Amount
Mode
Payment Date
Payment_id
Receive
Owns
Apartment
Apt_No
Bedrooms
Type
Area
Floor
Address
Complaint
Complaint_ID
Complaint_Description
Date
Owner
Name
Owner_Id
SSN
phone_no
Approves
Apartment_application
Email
Phone_no
Apt_No

Reduce the ER-Diagram to tables and Eliminating Redundancy



Relational Schema

*Note: The attributes underlined are primary keys and the ones which are **bold** are foreign keys and **bold-underlined** are composite primary keys.*

- Owner

  (Owner_id, SSN, Phone_no, Address, Name, **Email**)

- Payment

  (Payment_id, Mode, Payment_date, Amount, **Owner_id**, **Tenant_id**)

- Login

  (**Email**, Password,Role)

- Admin

  (Emp_ID, Name, Phone, Shift-Timings, Authorization Type, **Email**)

- Maintenance Staff

  (Emp_ID, Name, Phone, Shift-Timings, Contract Length, Role, **Email**)

- Complaint

  (Complaint_ID, Complaint_description, Complaint_Date, **Emp_ID**, **Tenant_id**)

- Tenant

  (Tenant_id, Age, SSN, Name, Perm_address, **Apt_No**, **Email**)

- Tenant-Contact

  (**Tenant_id**, **Phone**)

- Apartment

  (**Apt_No**, **Block_id**, Bedrooms, Type, Area, Floor, Address, **Owner_ID**)

- Block

  (Block_id, Block_name, Address)

- Parking

  (Spot_no, Type, **Block_id**, **Tenant_id**)

- ❖ Manages

  (**Emp_ID**, **Block_id**)

- ❖ Rents

  (**Tenant_id**, **Owner_id**, Rent_amount)

- ❖ Apartment_application

  (Email, Phone, Name, Apt_no, **Owner_id**)

Normalization of the schema

Step-by-Step Normalization

1. Identify Primary Keys

First, ensure each table has a primary key that uniquely identifies each row:

- **Owner - Owner_id**
- **Payment - Payment_id**
- **Login - Email (assuming each email is unique per user)**
- **Admin - Emp_ID**
- **Maintenance Staff - Emp_ID**
- **Complaint - Complaint_ID**
- **Tenant - Tenant_id**
- **Tenant-Contact - Tenant_id, Phone (Tenant_id and Phone form a composite primary key)**
- **Apartment - Apt_No**
- **Block - Block_id**
- **Parking - Spot_no**
- **Manages - Composite key (Emp_ID, Block_id)**
- **Rents - Composite key (Tenant_id, Owner_id)**
- **Apartment_application - Email**

2. Move Towards First Normal Form (1NF)

- ✔ **Remove repeating groups or arrays.**
- ✔ **All attributes have atomic values (no groups or arrays).**

3. Move Towards Second Normal Form (2NF)

- ✔ **Ensure that all non-key attributes are fully functional on the primary key.**
- ✔ **Remove partial dependencies (a dependency on part of the composite key).**

4. Move Towards Third Normal Form (3NF)

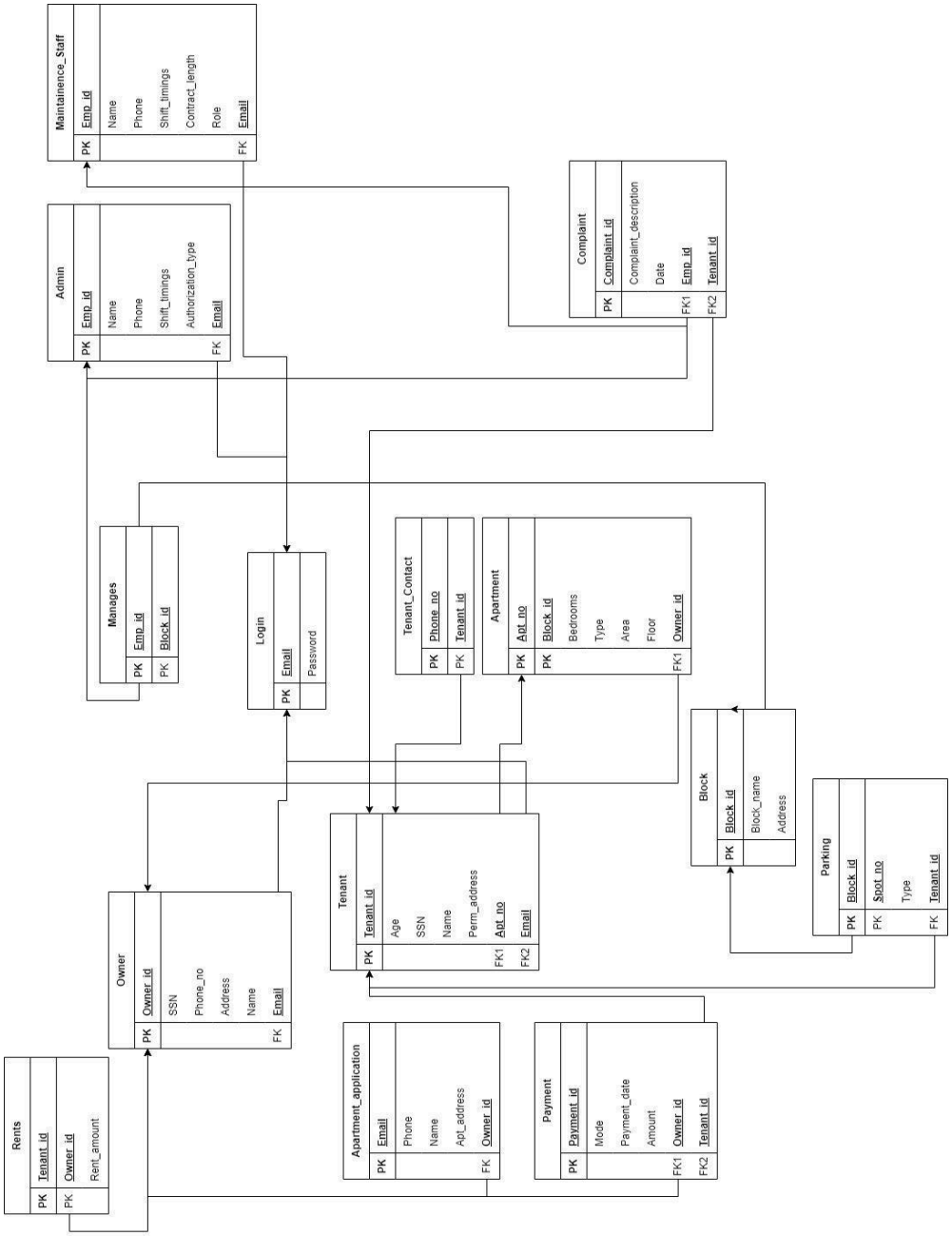
- ✔ **Remove transitive dependencies (a non-key column depends on another non-key column).**
- ✔ **Let's discuss potential issues:**
  - o **Owner and Tenant tables both have SSN, Address and Email. This can be a point of redundancy if not handled correctly.**
  - o **Admin and Maintenance Staff tables both have common attributes Emp_Id, Name, Phone, Shift-Timings and Email. This can be a point of redundancy of repeating data if not handled correctly.**
  - o **This suggests an opportunity to centralize Tenant-Owner data in one and Admin-Staff data in one table.**

Proposed Normalized Schema in 3NF

- ✔ Login (**Email**, Password)
- ✔ Tenant_Owner_Details (User_id, SSN, Name, Address, **Email**)
- ✔ Owner (**Owner_User_id,** Phone)
- ✔ Tenant (**Tenant_User_id**, Age, **Apt_No**)
- ✔ Tenant-Contact (**User_id,** Phone)
- ✔ Employee (Emp_Id, Name, Phone, Shift-Timings, **Email**)
- ✔ Admin (**Admin_id**, Authorization_Type) – Admin Id references Emp_Id
- ✔ Maintenance Staff (**Staff_id**, Contract_Length, Role) – Staff_Id references Emp_Id
- ✔ Apartment_application (Application_no, **Tenant_uid, Owner_uid**)
- ✔ Complaint (Complaint_ID, Complaint_description, Complaint_Date, **Emp_User_ID**, **Tenant_User_id**)
- ✔ Payment (Payment_id, Mode, Payment_date, Amount, **Owner_User_id**, **Tenant_id**)
- ✔ Apartment (**Apt_No**, **Block_id**, Bedrooms, Type, Area, Floor, Address, **Owner_ID**)
- ✔ Block (Block_id, Block_name, Address)
- ✔ Parking (Spot_no, Type, **Block_id**, **Tenant_id**)
- ✔ Manages (**Emp_ID, Block_id**): Emp_ID references Emp_Id in the Employee Table
- ✔ Rents (**Tenant_id**, **Owner_id**, Rent_amount)

This approach simplifies the schema by consolidating user-related data and reducing redundancy, potentially improving data integrity and query efficiency.

Schema Diagram



User Interface Design

A User Interface (UI) Design Diagram for the Apartment Leasing Management System (ALMS) can visually represent the key components and functionalities of the system. Here's a simplified UI design diagram focusing on the main features:

1. **Login Page:**

   - Username

   - Password

   - Login button
2. **Dashboard:**

   - Overview of key metrics (e.g., total tenants, total payments, total complaints)

   - Quick access to important features (e.g., manage tenants, manage apartments, view payments)
3. **Tenants Management:**

   - List of tenants with basic details (name, contact)

   - Ability to add, edit, and delete tenants

   - View tenant details, including leasing information and complaints history
4. **Apartments Management:**

   - List of apartments with details (unit number, size, type)

   - Ability to add, edit, and delete apartments

   - View apartment details, including owner information and current tenant

5. **Payments Management:**

   - List of payments with details (payment date, amount, tenant)

   - Ability to add, edit, and delete payments

   - Filter payments by date, tenant, or payment type.

6. **Employees Management:**

   - List of employees with details (name, role, contact)

   - Ability to add, edit, and delete employees

   - View employee details, including working hours and assigned tasks

7. **Complaints Management:**

   - List of complaints with details (complaint date, description, status)

   - Ability to add, edit, and resolve complaints

   - Filter complaints by date, tenant, or status

8. **Parking Management:**

   - List of parking spots with details (spot number, type, tenant)

   - Ability to assign or unassign parking spots

   - View parking spot details, including tenant information

9. **Blocks Management:**

   - List of blocks with details (block name, address)

   - Ability to add, edit, and delete blocks

   - View block details, including apartment count and current occupancy

10. **Settings:**

    - User profile management (change password, update contact information)

    - System settings (manage user roles, configure email notifications)

This diagram provides a high-level overview of the user interface design for the ALMS, focusing on the main functionalities and interactions. Detailed wireframes and mockups can be created for each feature to further refine the user experience.

Data type and their description

Listing the data types and descriptions for each attribute, along with proposed domain constraints for each attribute. We'll also organize the table attributes in tabular format for clarity:

- **Proposed Domains for attributes:**

Attribute	Proposed Domain	Description
Email	**EmailDomain**	**Domain for email attribute with specific format.**
Password	**PasswordDomain**	**Domain for password attribute with specific criteria.**
Phone_no	**PhoneNoDomain**	**Domain for phone number attribute with specific format.**
Emp_ID	**EmpIdDomain**	**Domain for employee ID attribute with specific constraints.**
Age	**AgeDomain**	**Domain for age attribute with specific constraints.**
Complaint_date	**ComplaintDateDomain**	**Domain for complaint date attribute with specific constraints.**
Rent_amount	**RentAmountDomain**	**Domain for rent amount attribute with specific constraints.**

Description of table attributed in tabular format:

Table Name	Attribute	Data Type/Domain	Description
Owner	Owner_id	VARCHAR(10)	Unique identifier for owners.
	Name	VARCHAR(20)	Name of the owner.
	SSN	VARCHAR(9)	Social Security Number of the owner.
	Phone_no	VARCHAR(10)	Phone number of the owner.
	Address	VARCHAR(30)	Address of the owner.
	Email	Email Domain	Email address of the owner.
Payment	Payment_id	VARCHAR(20)	Unique identifier for payments.
	Mode	VARCHAR(10)	Payment mode (e.g., cash, credit card).
	Payment_date	DATE	Date of the payment.
	Amount	NUMERIC(5,2)	Amount of the payment.
	Owner_id	VARCHAR(10)	Foreign key referencing the owner.
	Tenant_id	VARCHAR(10)	Foreign key referencing the tenant.
Login	Email	Email Domain	Email address used for login.
	Password	Password Domain	Password for login.

Admin	Emp_ID	Emp_ID Domain	Unique identifier for admins.
	Name	CHAR(10)	Name of the admin.
	Phone	VARCHAR(10)	Phone number of the admin.
	Shift_Timings	VARCHAR(50)	Shift timings of the admin.
	Authorization_Type	VARCHAR(15)	Type of authorization for the admin (e.g., manager).
	Email	Email Domain	Email address of the admin.
Maintenance_Staff	Emp_ID	Emp_ID Domain	Unique identifier for maintenance staff.
	Name	CHAR(10)	Name of the maintenance staff.
	Phone	VARCHAR(10)	Phone number of the maintenance staff.
	Shift_Timings	VARCHAR(50)	Shift timings of the maintenance staff.
	Contract_Length	VARCHAR(20)	Length of the maintenance staff's contract.
	Role	VARCHAR(20)	Role of the maintenance staff (e.g., technician).
	Email	Email Domain	Email address of the maintenance staff.

Tenant	Tenant_id	VARCHAR(10)	Unique identifier for tenants.
	Name	CHAR(20)	Name of the tenant.
	SSN	VARCHAR(9)	Social Security Number of the tenant.
	Age	Age Domain	Age of the tenant.
	Perm_address	VARCHAR(50)	Permanent address of the tenant.
	Apt_no	VARCHAR(10)	Apartment number of the tenant.
	Email	Email Domain	Email address of the tenant.
Tenant_Contact	Tenant_id	VARCHAR(10)	Foreign key referencing the tenant.
	Phone	Phone Number Domain	Phone number of the tenant.
Block	Block_id	INT	Unique identifier for blocks.
	Block_name	VARCHAR(10)	Name of the block.
	Address	VARCHAR(50)	Address of the block.
Apartment	Apt_No	VARCHAR(10)	Apartment number.
	Block_id	INT	Foreign key referencing the block.
	Bedrooms	INT	Number of bedrooms in the apartment.
	Type	VARCHAR(10)	Type of apartment (e.g., studio, one-bedroom).

	Area	INT	Area of the apartment in square feet.
	Floor	INT	Floor number of the apartment.
	Address	VARCHAR(50)	Address of the apartment.
	Owner_id	VARCHAR(10)	Foreign key referencing the owner.
Parking	Spot_no	INT	Parking spot number.
	Type	VARCHAR(10)	Type of parking spot (e.g., reserved, visitor).
	Block_id	INT	Foreign key referencing the block.
	Tenant_id	VARCHAR(10)	Foreign key referencing the tenant.
Manages	Emp_ID	Emp_ID Domain	Foreign key referencing the admin.
	Block_id	INT	Foreign key referencing the block.
Complaint	Complaint_ID	VARCHAR(20)	Unique identifier for complaints.
	Complaint_description	VARCHAR(100)	Description of the complaint.
	Complaint_date	Complaint_date Domain	Date when the complaint was filed.
	Emp_ID	Emp_ID Domain	Foreign key referencing the maintenance staff.
	Tenant_id	VARCHAR(10)	Foreign key referencing the tenant.

Rents	Tenant_id	VARCHAR(10)	Foreign key referencing the tenant.
	Owner_id	VARCHAR(10)	Foreign key referencing the owner.
	Rent_amount	Rent_amount Domain	Rent amount for the tenant.
Apartment_application	Email	Email Domain	Email address used for application.
	Phone	Phone Number Domain	Phone number used for application.
	Name	CHAR(20)	Name of the applicant.
	Apt_no	VARCHAR(10)	Apartment number of the apartment applied for.
	Owner_id	VARCHAR(10)	Foreign key referencing the owner.

Proposed constraints for attributes and implementation

Attribute	Proposed Constraint	Implementation
Email	Email format	ALTER TABLE <Table_Name> ADD CONSTRAINT check_email CHECK (Email ~ '^[a-zA-Z0-9.%]+@[a-zA-Z]+\.[a-zA-Z]+$');
Password	Password criteria	ALTER TABLE <Table_Name> ADD CONSTRAINT check_password CHECK (Password ~

		'^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[!@#$%^&*])(?=.{8,})[a-zA-Z0-9!@#$%^&*]+$');
Phone_no	Phone number format	ALTER TABLE <Table_Name> ADD CONSTRAINT check_phone CHECK (Phone_no ~ '^\+[0-9]{1,3}[0-9]{10}$');
Owner_id	Unique constraint	ALTER TABLE Owner ADD CONSTRAINT unique_owner_id UNIQUE (Owner_id);
Name	No constraints	No additional constraints needed
SSN	Unique constraint	ALTER TABLE <Table_Name> ADD CONSTRAINT unique_ssn UNIQUE (SSN);
Address	No constraints	No additional constraints needed
Emp_ID	Unique constraint	ALTER TABLE <Table_Name> ADD CONSTRAINT unique_emp_id UNIQUE (Emp_ID);
Shift_Timings	No constraints	No additional constraints needed
Authorization_Type	No constraints	No additional constraints needed
Tenant_id	Unique constraint	ALTER TABLE <Table_Name> ADD CONSTRAINT unique_tenant_id UNIQUE (Tenant_id);
Apt_no	No constraints	No additional constraints needed

Block_id	Unique constraint	ALTER TABLE <Table_Name> ADD CONSTRAINT unique_block_id UNIQUE (Block_id);
Spot_no	No constraints	No additional constraints needed
Complaint_ID	Unique constraint	ALTER TABLE <Table_Name> ADD CONSTRAINT unique_complaint_id UNIQUE (Complaint_ID);
Tenant_id	No constraints	No additional constraints needed
Rent_amount	No constraints	No additional constraints needed
Email	Unique constraint	ALTER TABLE <Table_Name> ADD CONSTRAINT unique_email UNIQUE (Email);
Phone	No constraints	No additional constraints needed
Apt_address	No constraints	No additional constraints needed

Planning and implementing referential integrity (cascade delete/update, and others)

```
-- Add foreign key constraints using ALTER TABLE statements
ALTER TABLE Owner ADD CONSTRAINT fk_owner_login FOREIGN KEY (Email) REFERENCES Login ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Payment ADD CONSTRAINT fk_payment_owner FOREIGN KEY (Owner_id) REFERENCES Owner ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Payment ADD CONSTRAINT fk_payment_tenant FOREIGN KEY (Tenant_id) REFERENCES Tenant ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Admin ADD CONSTRAINT fk_admin_login FOREIGN KEY (Email) REFERENCES Login ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Maintenance_Staff ADD CONSTRAINT fk_maintenance_staff_login FOREIGN KEY (Email) REFERENCES Login ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Tenant ADD CONSTRAINT fk_tenant_login FOREIGN KEY (Email) REFERENCES Login ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Tenant ADD CONSTRAINT fk_tenant_apartment FOREIGN KEY (Apt_no,block_id) REFERENCES Apartment;
ALTER TABLE Tenant_Contact ADD CONSTRAINT fk_tenant_contact FOREIGN KEY (Tenant_id) REFERENCES Tenant ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Apartment ADD CONSTRAINT fk_apartment_block FOREIGN KEY (Block_id) REFERENCES Block ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Apartment ADD CONSTRAINT fk_apartment_owner FOREIGN KEY (Owner_id) REFERENCES Owner ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Parking ADD CONSTRAINT fk_parking_block FOREIGN KEY (Block_id) REFERENCES Block ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Parking ADD CONSTRAINT fk_parking_tenant FOREIGN KEY (Tenant_id) REFERENCES Tenant ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Manages ADD CONSTRAINT fk_manages_admin FOREIGN KEY (Emp_ID) REFERENCES Admin ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Manages ADD CONSTRAINT fk_manages_block FOREIGN KEY (Block_id) REFERENCES Block ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Complaint ADD CONSTRAINT fk_complaint_maintenance_staff FOREIGN KEY (Emp_ID) REFERENCES Maintenance_Staff ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Complaint ADD CONSTRAINT fk_complaint_tenant FOREIGN KEY (Tenant_id) REFERENCES Tenant ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Rents ADD CONSTRAINT fk_rents_tenant FOREIGN KEY (Tenant_id) REFERENCES Tenant ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Rents ADD CONSTRAINT fk_rents_owner FOREIGN KEY (Owner_id) REFERENCES Owner ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Apartment_application ADD CONSTRAINT fk_apartment_application_login FOREIGN KEY (Email) REFERENCES Login ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Apartment_application ADD CONSTRAINT fk_apartment_application_owner FOREIGN KEY (Owner_id) REFERENCES Owner ON UPDATE CASCADE ON DELETE CASCADE;
```

- **ALTER TABLE Owner ADD CONSTRAINT fk_owner_login FOREIGN KEY (Email) REFERENCES Login ON UPDATE CASCADE ON DELETE CASCADE;**
    - ★ This statement adds a foreign key constraint named fk_owner_login to the Owner table.
    - ★ It ensures that the Email column in the Owner table references the email column in the Login table.
    - ★ The ON UPDATE CASCADE ON DELETE CASCADE specifies that if the referenced email in the Login table is updated or deleted, the corresponding rows in the Owner table will also be updated or deleted, cascading the changes.
- **ALTER TABLE Payment ADD CONSTRAINT fk_payment_owner FOREIGN KEY (Owner_id) REFERENCES Owner ON UPDATE CASCADE ON DELETE CASCADE;**
    - ★ Adds a foreign key constraint named fk_payment_owner to the Payment table.
    - ★ Ensures that the Owner_id column in the Payment table references the Owner_id column in the Owner table.
    - ★ Uses ON UPDATE CASCADE ON DELETE CASCADE to cascade updates and deletes.

- **ALTER TABLE Payment ADD CONSTRAINT fk_payment_tenant FOREIGN KEY (Tenant_id) REFERENCES Tenant ON UPDATE CASCADE ON DELETE CASCADE;**
    - ★ Adds a foreign key constraint named fk_payment_tenant to the Payment table.
    - ★ Ensures that the Tenant_id column in the Payment table references the Tenant_id column in the Tenant table.
    - ★ Uses ON UPDATE CASCADE ON DELETE CASCADE for cascading updates and deletes.

- **ALTER TABLE Admin ADD CONSTRAINT fk_admin_login FOREIGN KEY (Email) REFERENCES Login ON UPDATE CASCADE ON DELETE CASCADE;**
    - ★ Adds a foreign key constraint named fk_admin_login to the Admin table.
    - ★ Ensures that the Email column in the Admin table references the Email column in the Login table.
    - ★ Uses ON UPDATE CASCADE ON DELETE CASCADE for cascading updates and deletes.

- **ALTER TABLE Maintenance_Staff ADD CONSTRAINT fk_maintenance_staff_login FOREIGN KEY (Email) REFERENCES Login ON UPDATE CASCADE ON DELETE CASCADE;**
    - ★ Adds a foreign key constraint named fk_maintenance_staff_login to the Maintenance_Staff table.
    - ★ Ensures that the Email column in the Maintenance_Staff table references the Email column in the Login table.
    - ★ Uses ON UPDATE CASCADE ON DELETE CASCADE for cascading updates and deletes.

- **ALTER TABLE Tenant ADD CONSTRAINT fk_tenant_login FOREIGN KEY (Email) REFERENCES Login ON UPDATE CASCADE ON DELETE CASCADE;**
    - ★ Adds a foreign key constraint named fk_tenant_login to the Tenant table.
    - ★ Ensures that the Email column in the Tenant table references the Email column in the Login table.
    - ★ Uses ON UPDATE CASCADE ON DELETE CASCADE for cascading updates and deletes.

- **ALTER TABLE Tenant ADD CONSTRAINT fk_tenant_apartment FOREIGN KEY (Apt_no,block_id) REFERENCES Apartment;**
    - ★ Adds a foreign key constraint to the Tenant table.
    - ★ Ensures that the combination of Apt_no and block_id columns in the Tenant table references the corresponding columns in the Apartment table.

- **ALTER TABLE Tenant_Contact ADD CONSTRAINT fk_tenant_contact FOREIGN KEY (Tenant_id) REFERENCES Tenant ON UPDATE CASCADE ON DELETE CASCADE;**
    - ★ Adds a foreign key constraint named fk_tenant_contact to the Tenant_Contact table.
    - ★ Ensures that the Tenant_id column in the Tenant_Contact table references the Tenant_id column in the Tenant table.
    - ★ Uses ON UPDATE CASCADE ON DELETE CASCADE for cascading updates and deletes.

- **ALTER TABLE Apartment ADD CONSTRAINT fk_apartment_block FOREIGN KEY (Block_id) REFERENCES Block ON UPDATE CASCADE ON DELETE CASCADE;**
    - ★ Adds a foreign key constraint named fk_apartment_block to the Apartment table.

★ Ensures that the Block_id column in the Apartment table references the Block_id column in the Block table.
★ Uses ON UPDATE CASCADE ON DELETE CASCADE for cascading updates and deletes.

- **ALTER TABLE Apartment ADD CONSTRAINT fk_apartment_owner FOREIGN KEY (Owner_id) REFERENCES Owner ON UPDATE CASCADE ON DELETE CASCADE;**
  - ★ Adds a foreign key constraint named fk_apartment_owner to the Apartment table.
  - ★ Ensures that the Owner_id column in the Apartment table references the Owner_id column in the Owner table.
  - ★ Uses ON UPDATE CASCADE ON DELETE CASCADE for cascading updates and deletes.

- **ALTER TABLE Parking ADD CONSTRAINT fk_parking_block FOREIGN KEY (Block_id) REFERENCES Block ON UPDATE CASCADE ON DELETE CASCADE;**
  - ★ Adds a foreign key constraint named fk_parking_block to the Parking table.
  - ★ Ensures that the Block_id column in the Parking table references the Block_id column in the Block table.
  - ★ Uses ON UPDATE CASCADE ON DELETE CASCADE for cascading updates and deletes.

- **ALTER TABLE Parking ADD CONSTRAINT fk_parking_tenant FOREIGN KEY (Tenant_id) REFERENCES Tenant ON UPDATE CASCADE ON DELETE CASCADE;**
  - ★ Adds a foreign key constraint named fk_parking_tenant to the Parking table.
  - ★ Ensures that the Tenant_id column in the Parking table references the Tenant_id column in the Tenant table.

★ Uses ON UPDATE CASCADE ON DELETE CASCADE for cascading updates and deletes.

- **ALTER TABLE Manages ADD CONSTRAINT fk_manages_admin FOREIGN KEY (Emp_ID) REFERENCES Admin ON UPDATE CASCADE ON DELETE CASCADE;**
  - ★ This statement adds a foreign key constraint named fk_manages_admin to the Manages table.
  - ★ It ensures that the Emp_ID column in the Manages table references the Emp_ID column in the Admin table.
  - ★ ON UPDATE CASCADE ON DELETE CASCADE specifies that if the referenced Emp_ID in the Admin table is updated or deleted, the corresponding rows in the Manages table will also be updated or deleted, ensuring referential integrity.

- **ALTER TABLE Manages ADD CONSTRAINT fk_manages_block FOREIGN KEY (Block_id) REFERENCES Block ON UPDATE CASCADE ON DELETE CASCADE;**
  - ★ This statement adds a foreign key constraint named fk_manages_block to the Manages table.
  - ★ It ensures that the Block_id column in the Manages table references the Block_id column in the Block table.
  - ★ Similar to the previous statement, ON UPDATE CASCADE ON DELETE CASCADE specifies the cascading behavior for updates and deletes.

- **ALTER TABLE Complaint ADD CONSTRAINT fk_complaint_maintenance_staff FOREIGN KEY (Emp_ID) REFERENCES Maintenance_Staff ON UPDATE CASCADE ON DELETE CASCADE;**
  - ★ This statement adds a foreign key constraint named fk_complaint_maintenance_staff to the Complaint table.

★ It ensures that the Emp_ID column in the Complaint table references the Emp_ID column in the Maintenance_Staff table.

★ Once again, ON UPDATE CASCADE ON DELETE CASCADE is used for cascading updates and deletes.

- **ALTER TABLE Complaint ADD CONSTRAINT fk_complaint_tenant FOREIGN KEY (Tenant_id) REFERENCES Tenant ON UPDATE CASCADE ON DELETE CASCADE;**

★ This statement adds a foreign key constraint named fk_complaint_tenant to the Complaint table.

★ It ensures that the Tenant_id column in the Complaint table references the Tenant_id column in the Tenant table.

★ The cascade options ON UPDATE CASCADE ON DELETE CASCADE specify the desired behavior for updates and deletes.

- **ALTER TABLE Rents ADD CONSTRAINT fk_rents_tenant FOREIGN KEY (Tenant_id) REFERENCES Tenant ON UPDATE CASCADE ON DELETE CASCADE;**

★ This statement adds a foreign key constraint named fk_rents_tenant to the Rents table.

★ It ensures that the Tenant_id column in the Rents table references the Tenant_id column in the Tenant table.

★ Once again, ON UPDATE CASCADE ON DELETE CASCADE is used for cascading updates and deletes.

- **ALTER TABLE Rents ADD CONSTRAINT fk_rents_owner FOREIGN KEY (Owner_id) REFERENCES Owner ON UPDATE CASCADE ON DELETE CASCADE;**

★ This statement adds a foreign key constraint named fk_rents_owner to the Rents table.

★ It ensures that the Owner_id column in the Rents table references the Owner_id column in the Owner table.
★ The cascade options ON UPDATE CASCADE ON DELETE CASCADE specify the desired behavior for updates and deletes.

- **ALTER TABLE Apartment_application ADD CONSTRAINT fk_apartment_application_login FOREIGN KEY (Email) REFERENCES Login ON UPDATE CASCADE ON DELETE CASCADE;**
  - ★ This statement adds a foreign key constraint named fk_apartment_application_login to the Apartment_application table.
  - ★ It ensures that the Email column in the Apartment_application table references the Email column in the Login table.
  - ★ Once again, ON UPDATE CASCADE ON DELETE CASCADE is used for cascading updates and deletes.

- **ALTER TABLE Apartment_application ADD CONSTRAINT fk_apartment_application_owner FOREIGN KEY (Owner_id) REFERENCES Owner ON UPDATE CASCADE ON DELETE CASCADE;**
  - ★ This statement adds a foreign key constraint named fk_apartment_application_owner to the Apartment_application table.
  - ★ It ensures that the Owner_id column in the Apartment_application table references the Owner_id column in the Owner table.
  - ★ The cascade options ON UPDATE CASCADE ON DELETE CASCADE specify the desired behavior for updates and deletes.

User Roles for the Database and Tables:

User roles created for the database and table are as follows:

- ➔ **`app_admin_role`**
- ➔ **`app_owner_role`**
- ➔ **`app_tenant_role`)**

Each role is assigned specific permissions to access and interact with the database tables. Let's explain the roles and their permissions.

1. app_admin_role: This role is intended for administrative users who have full control over the system.

- ❖ **Permissions:**
  - **Granted `SELECT`, `INSERT`, `UPDATE`, and `DELETE` permissions on all tables in the `public` schema. This allows admins to perform CRUD operations on all data.**
  - **Granted `ALL PRIVILEGES` on all sequences in the `public` schema. Sequences are typically used for generating unique identifiers, and granting all privileges ensures admins can manipulate them as needed.**

2. app_owner_role: This role is intended for apartment owners who need to manage their properties.

- ❖ **Permissions:**
  - **Granted `SELECT` permission on the `apartment` table. Owners can view details of their own properties.**
  - **Granted `SELECT` and `UPDATE` permissions on the `complaint` table. This allows owners to respond to complaints related to their properties.**
  - **Granted `SELECT` permission on the `payment` table. Owners can view payment details for their properties.**
  - **Granted `SELECT` permission on the `tenant` table. Owners can view tenant information for their properties.**

3. **App_tenant_role: This role is intended for tenants who rent apartments within the complex.**

- **Permissions:**
  - **Granted `INSERT` and `SELECT` permissions on the `payment` table. Tenants can make payments and view their own payment history.**
  - **Granted `INSERT` and `SELECT` permissions on the `complaint` table. Tenants can file complaints and view their own complaints.**
  - **Granted `SELECT` permission on the `apartment` table. Tenants can view details of their own apartments.**

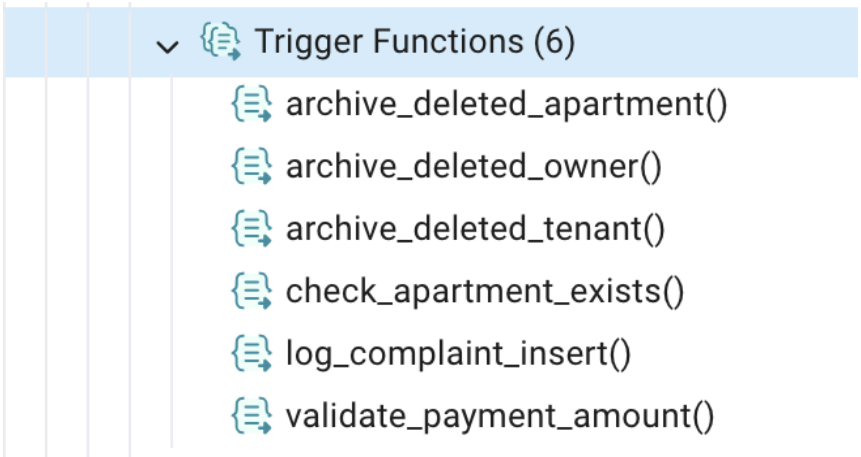
Each role is assigned permissions tailored to the specific needs and responsibilities of the users within the system. By carefully defining roles and permissions, access control is enforced, ensuring that users can only perform actions relevant to their roles while maintaining the security and integrity of the data.

```sql
CREATE ROLE app_admin_role LOGIN PASSWORD 'admin_password';
CREATE ROLE app_owner_role LOGIN PASSWORD 'owner_password';
CREATE ROLE app_tenant_role LOGIN PASSWORD 'tenant_password';

GRANT CONNECT ON DATABASE AptMgmt TO app_admin_role ;
GRANT CONNECT ON DATABASE AptMgmt TO app_owner_role ;
GRANT CONNECT ON DATABASE AptMgmt TO app_tenant_role ;


GRANT USAGE ON SCHEMA schema_name TO app_admin_role;
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO app_admin_role;
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO app_admin_role;



-- owner
-- Grant access to view their properties
GRANT SELECT ON TABLE apartments TO app_owner_role;
-- Grant access to manage (respond to) complaints related to their properties
GRANT SELECT, UPDATE ON TABLE complaints TO app_owner_role;
-- Grant access to view payments for their properties
GRANT SELECT ON TABLE payments TO app_owner_role;
-- Grant access to view tenants for their properties
GRANT SELECT ON tenants TO app_owner_role;


--tenant role
GRANT SELECT, INSERT, Update, delete ON complaints TO app_tenant_role;
-- Grant access to view their apartment details
GRANT SELECT ON TABLE apartment TO app_tenant_role;
-- Grant access to make payments and view their own payments
GRANT INSERT, SELECT ON TABLE payment TO app_tenant_role;
```

Data Output | Messages | Notifications

Total rows: 0 of 0 Ln 27,

Triggers:



1. **Complaint Audit Trigger**

Name:

after_complaint_insert

Operation: AFTER INSERT

Table: complaint

Purpose: This trigger captures the details of each new complaint when it is inserted into the complaint table. It logs the complaint_id, the date it was created, and the tenant_id associated with the complaint to a separate complaint_audit table. The log can be used for audit purposes to maintain a record of all complaints filed.

```
-- since admin or tenant might delete a complaint either to withdraw or once complaint is solved might delete it
CREATE OR REPLACE FUNCTION log_complaint_insert()
RETURNS TRIGGER AS $$
BEGIN
  INSERT INTO complaint_audit (complaint_id, created_at, tenant_id)
  VALUES (NEW.complaint_id, NEW.date, NEW.tenant_id);
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER after_complaint_insert
AFTER INSERT ON complaint
FOR EACH ROW EXECUTE FUNCTION log_complaint_insert();
```

2. **Apartment Existence Check**

Trigger Name:

before_tenant_insert

Operation: BEFORE INSERT

Table: tenant

Purpose: This trigger validates the existence of an apartment number in the apartment table before a new tenant record is inserted into the tenant table. If the apartment number does not exist, the trigger raises an exception to prevent the creation of a tenant record with an invalid apartment reference, thus maintaining referential integrity.

```
-- Check if the apartment number exists in the apartment table
CREATE OR REPLACE FUNCTION check_apartment_exists()
RETURNS TRIGGER AS $$
BEGIN
    -- Check if the apartment number exists in the apartment table
    IF NOT EXISTS (SELECT 1 FROM apartment WHERE apt_no = NEW.apt_no) THEN
        RAISE EXCEPTION 'Apartment number does not exist.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_tenant_insert
BEFORE INSERT ON tenant
FOR EACH ROW EXECUTE FUNCTION check_apartment_exists();
```

3. **Payment Validation Trigger**

Name: before_payment_insert

Operation: BEFORE INSERT

Table: payment

Purpose: This trigger ensures that the amount for a new payment is positive before it is inserted into the payment table. If the amount is not positive, the trigger raises an exception and prevents the insertion. This helps maintain data integrity by enforcing business rules that payment amounts must be greater than zero.

```
-- Trigger to Validate Payment Amount
CREATE OR REPLACE FUNCTION validate_payment_amount()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.amount <= 0 THEN
    RAISE EXCEPTION 'Payment amount must be positive.';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_payment_insert
BEFORE INSERT ON payment
FOR EACH ROW EXECUTE FUNCTION validate_payment_amount();
```

4. **Owner Archiving Trigger**

Name: owner_before delete

Operation: BEFORE DELETE

Table: owner

Purpose: Similar to the tenant archiving trigger, this trigger archives the data of owners when they are deleted from the owner table. The archived information includes personal details and the date of archiving, and it is stored in the owner_archive table for future reference or audit trails.

```
-- When a owner is removed from the system, archive their information for record-keeping purposes.
CREATE OR REPLACE FUNCTION archive_deleted_owner()
RETURNS TRIGGER AS $$
BEGIN
  INSERT INTO owner_archive (owner_id, ssn, phone_no, address, name, email, archived_date)
  VALUES (OLD.owner_id, OLD.ssn, OLD.phone_no, OLD.address, OLD.name, OLD.email, CURRENT_DATE);
  RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER owner_before_delete
BEFORE DELETE ON owner
FOR EACH ROW EXECUTE FUNCTION archive_deleted_owner();
```

5. **Apartment Archiving Trigger**

Name: apartment_before_delete

Operation: BEFORE DELETE

Table: APARTMENT

Purpose: This trigger is invoked before the deletion of an apartment record. It archives the details of the apartment, including its number, block ID, and other attributes, into an apartment_archive table with the addition of the archival date. It serves to keep a historical log of apartment data that has been removed from active listings.

```
CREATE OR REPLACE FUNCTION archive_deleted_apartment()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO apartment_archive (
        Apt_No,
        Block_id,
        Bedrooms,
        Type,
        Area,
        Floor,
        Address,
        Owner_id,
        Archived_date
    ) VALUES (
        OLD.Apt_No,
        OLD.Block_id,
        OLD.Bedrooms,
        OLD.Type,
        OLD.Area,
        OLD.Floor,
        OLD.Address,
        OLD.Owner_id,
        CURRENT_DATE
    );
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER apartment_before_delete
BEFORE DELETE ON Apartment
FOR EACH ROW EXECUTE FUNCTION archive_deleted_apartment();
```

Encryption/Decryption Details

Concepts of bcrypt and salt used for securely hashing passwords in our database.

1. **Hashing Passwords:**
   - **When storing passwords in a database, it's essential to ensure that they are securely hashed to prevent unauthorized access to sensitive user data.**
   - **Hashing is a process of converting plain text passwords into a fixed-size string of characters, called a hash value, using a cryptographic hash function.**

A good cryptographic hash function should have the following properties:

- **Deterministic: The same input always produces the same output.**
- **Irreversible: It should be computationally infeasible to reverse the hash to obtain the original password.**

- Unique: Different inputs should produce different hash values.
- Collision-resistant: It should be computationally infeasible to find two different inputs that produce the same hash value.

2. Bcrypt Algorithm:
   - Bcrypt (Blowfish-crypt) is a popular password-hashing function designed to be computationally expensive, making it resistant to brute-force attacks.
   - It uses the Blowfish cipher internally and incorporates a salt value and a cost parameter to enhance security.
   - Bcrypt introduces a work factor or cost parameter (typically denoted as `cost` or `strength`), which determines the number of iterations used in the hashing process. The higher the cost, the more computationally expensive the hashing becomes.
   - The cost parameter allows developers to adjust the computational intensity of the algorithm, making it adaptive to hardware advancements and increasing security over time.

3. Salt:
   - In addition to the cost parameter, bcrypt also incorporates a salt value into the hashing process.
   - A salt is a randomly generated value that is combined with the password before hashing. It ensures that even if two users have the same password, their hash values will be different due to the unique salt.
   - Salting prevents rainbow table attacks, where attackers pre-compute hashes for commonly used passwords and compare them to stolen password hashes to find matches.
   - By using a unique salt for each password, even if two users have the same password, their hash values will be different, adding an extra layer of security.

In summary, bcrypt with salt provides a robust mechanism for securely hashing passwords by incorporating a cost parameter to adjust computational intensity and a unique salt value for each password to prevent attacks such as brute-forcing and rainbow table attacks. This combination ensures that passwords are securely stored and protected in a database.

User logging in with the username and password and bcrypt hashing with a plain password authentication request

Example:

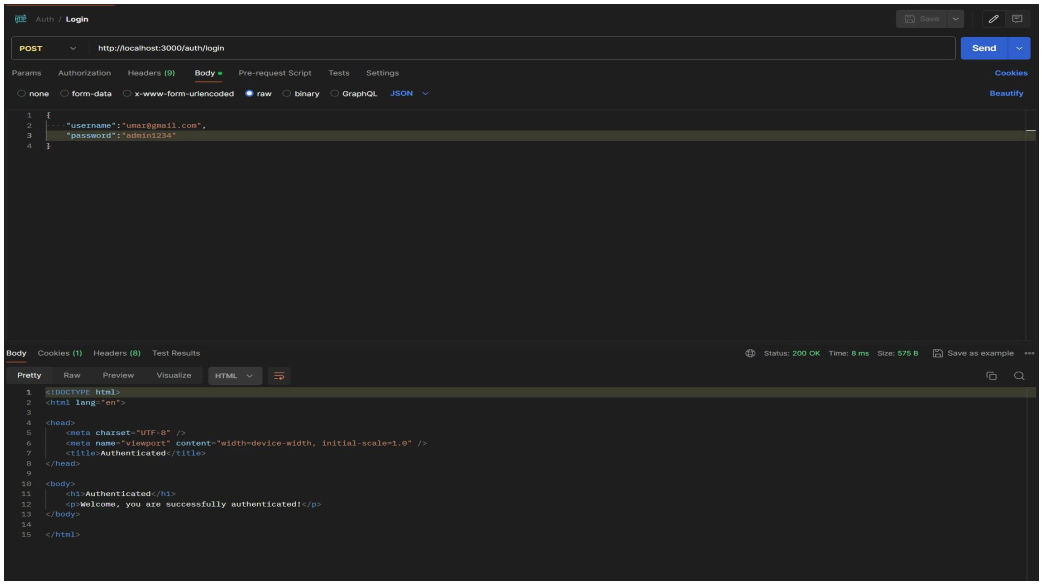
Username :umar@gmail.com

Password: admin1234

After Encryption:

Data Output | Messages | Notifications

	email [PK] character varying (50)	password character varying (72)	role character varying (15)
1	umar@gmail.com	$2b$10$ae.JxngKDt30zs2zGp8WiOUPOO4RAOSl5CZAot8PHSbwCrzVfn...	Admin



Tuple data for each table of the database

1. Admin Table:

Query | Query History

```
1  select * from admin;
2
```

Data Output | Messages | Notifications

	emp_id [PK] character varying (38)	name character	phone character varying (15)	shift_timings character varying (50)	authorization_type character varying (15)	email character varying (50)
1	A52c756f6-df17-4df0-892c-b1efcf141d38	umar	+1234567890123	9:00 AM - 5:00 PM	Admin	umar@gmail.com
2	A6efb4f57-cbe9-4266-8a8a-594b7f251f3d	Kaylah	+802753137791	9:00 AM - 5:00 PM	Admin	Reed65@hotmail.com
3	Aaaedbb47-6b2b-4537-824e-ed4ef638ba1b	Aniyah	+502933154536	9:00 AM - 5:00 PM	Admin	Kadin97@gmail.com
4	A79603b2d-f217-4954-a814-5bc42a2460fc	Keara	+689756122145	9:00 AM - 5:00 PM	Admin	Burnice.Weimann@hotmail.com
5	Ad6ced32a-e88f-4306-900b-d64cb7c55509	Jayne	+769333122989	9:00 AM - 5:00 PM	Admin	Nyah.Rempel@hotmail.com
6	Afee35ea3-dbb3-4895-9c0f-dbf16fda4938	Reuben	+146544112885	9:00 AM - 5:00 PM	Admin	Aric37@yahoo.com
7	A48b6c8b0-2b93-43f7-9913-96810a1c03e7	Angela	+948166093293	9:00 AM - 5:00 PM	Admin	Karine59@yahoo.com
8	A0c709594-d480-4547-91fa-b75862a6c81f	Deanna	+866089062780	9:00 AM - 5:00 PM	Admin	Rex.Schneider75@gmail.com
9	A6e870bca-526d-4baf-b935-6888d6067ab1	Bernadine	+997848316184	9:00 AM - 5:00 PM	Admin	Liam19@gmail.com

2. Login table:

Query | Query History

```
1  select * from login;
2
```

Data Output | Messages | Notifications

	email [PK] character varying (50)	password character varying (72)	role character varying (15)
1	umar@gmail.com	$2b$10$ae.JxngKDt30zs2zGp8WiOUPOO4RAOSl5CZAot8PHSbwCrzVfn...	Admin
2	employee@gmail.com	$2b$10$.BCDMmC07T347VhMfE/hJukFWQiBpYJDRbep.i1c5bpWr7fepf...	Employee
3	Reed65@hotmail.com	$2b$10$.O.LeHF4qN/gE6PTz2x6Mub6yavuDEUCg/a9iFECJWuxnOaWiw...	Admin
4	Kadin97@gmail.com	$2b$10$LsEjSCBv9NYea.OEFh1KP.AXOQEkMoDMuP86i0DWIXjrIvizssbLy	Admin
5	Burnice.Weimann@hotmail.c...	$2b$10$gTZ50z2KM2GcrzVOoUigfekqgfXNOpzT4dpWF1exCHxQQzRV5...	Admin
6	Nyah.Rempel@hotmail.com	$2b$10$8Byc1DV5EU76ZgXR5OpJ8u3gYYnLZCHRiPKlsYTvORczDonG.r...	Admin
7	Aric37@yahoo.com	$2b$10$RDHoARymGRrgOaqAblX05eJJKPQ4Btu440tQZP0xjfs9GP94gj...	Admin
8	Karine59@yahoo.com	$2b$10$RDsavXCMCQHKkcUmAxPTQOcKFr9YmRmpUwxOHPr4Sontl3y...	Admin
9	Rex.Schneider75@gmail.com	$2b$10$XbGQLPsvOPDAk/OHDT/vcOKp8jNnEjf.oUbytjwCStVk2eLQtaw...	Admin
10	Liam19@gmail.com	$2b$10$tVDVderzbYjrEx1rYvz5Z..fMzr3iHLlES/FjMHhWbtJ0EuLoiCmu	Admin
11	Francisca.Frami@yahoo.com	$2b$10$OGDnnnh8WqB.D9lVgjy3autX23bnlVq.JRB2va/kiiwopxH.NTR0q	Owner
12	Stella.Windler51@yahoo.com	$2b$10$/8KDBCaGG5OQQVDson227OQPZIgvmhgrIAcakmCBZTUJWvp...	Owner
13	Leon70@gmail.com	$2b$10$58bgmadXJphAUSSLtLMjuiLLb.S9jUIep1UXfR1aWfxYTBomnf...	Owner
14	Murl.Stehr52@gmail.com	$2b$10$qw6.zGGbN23cVwPgdi1C7Oin1/.hrcV8RbJoaswAuw3l3oJ0XK...	Owner
15	Lou.Lakin@gmail.com	$2b$10$O.si9ej34O2BvskQvrgtneylXHlnmVejFeLSlrL1raR1X4q.RU9sW	Owner
16	Cleta77@gmail.com	$2b$10$Jt7qnkGx38HYQAc9WnOl3.LahQqDAY5sJrilspmHQJ5zU0ReA8...	Owner
17	Alfred.Herzog@gmail.com	$2b$10$27xXwL.f3lyT9YJuertylO0heAdY48SjM0avK8ZhpuOaoDrlV6PHi	Owner
18	Jefferey.Brekke@hotmail.com	$2b$10$KhEENF2PaQReKtNIrWnRTOzbo2yOX6L5P1g.jB.f5sd6mve1Dj2...	Owner
19	Imelda72@hotmail.com	$2b$10$RkVGYQKL585TpkbbqDc0RumU1eQ8vLWaYBeGP6m5Hlo6Ezw...	Owner
20	London60@hotmail.com	$2b$10$GHYuUP5JMRL9yUk7o4.cCOjUUrtkE6kyVGRngClGu9qiiV3lv5v...	Owner

3. Owner Table

Open File
Alt O

Query History

```
ct * from owner;
```

Data Output | Messages | Notifications

	owner_id [PK] character varying (38)	name character varying (20)	ssn character varying (9)	phone_no character varying (15)	address character varying (30)	email character varying (50)
1	0f230bd53-f96a-49f2-ad95-dc77d1d7854e	Barton	123546987	+235744983942	07842 Braxton Hills	Francisca.Frami@yahoo.com
2	0b6c977a5-5845-4d7f-b2b5-3df5eba282ce	Janet	125687887	+647757836935	441 Mandy Drive	Stella.Windler51@yahoo.com
3	040740fe1-1371-41f6-b006-de3e84c9c07b	Claude	125617887	+873569698189	4718 Lonnie Ridge	Leon70@gmail.com
4	0d1410bac-2e35-4e26-aa96-fc0fe8ac5334	Aliyah	125617887	+626972802066	43298 Ethyl Plain	Murl.Stehr52@gmail.com
5	076ac053c-0e2f-4e5d-bf43-5d1442e89162	Elsie	125677887	+133796310613	58866 Rebekah Hollow	Lou.Lakin@gmail.com
6	0e567f8a6-dce2-4b2e-845b-e49bbdde8277	Irma	121577887	+818224400914	463 Elta Trace	Cleta77@gmail.com
7	04e9f1b75-e141-402d-9032-8dd91e4da3b0	Elias	224588887	+843402151053	1236 Paris Plains	Alfred.Herzog@gmail.com
8	0ed4d58de-e8bc-44b3-be04-9de6d9b39e59	Rebekah	224668887	+769563319228	92427 Cummings Heights	Jefferey.Brekke@hotmail.com
9	05e10aee0-5e5c-4918-b2ec-bd2d17d13fe8	Chesley	998668887	+112508674193	732 Sheldon Corners	Imelda72@hotmail.com
10	08b95ad9b-7082-410d-a342-adf2a7ba900c	Jailyn	998568887	+716674751612	7773 Jacobi Plaza	London60@hotmail.com

4. Block Table

Query | Query History

```
1 select * from owner;
2 select * from block;
```

Data Output | Messages | Notifications

	block_id [PK] integer	block_name character varying (10)	address character varying (50)
1	33028789	A	1170 Kessler Vista
2	42509553	B	447 Timmy Walk
3	52505100	B	3056 Wilhelmine Fields
4	93702634	D	3116 Idell Crossing
5	49786040	C	59675 Gutmann Trace
6	51206222	J	1552 Serenity Lock
7	41251798	I	0846 Korbin Burg
8	34543390	E	08546 Orin Corner
9	93731880	A	55321 Jacobs Ranch
10	39169924	C	301 Charlotte Island

5. Apartment Table

Query | Query History

```
1 select * from apartment;
2
```

Data Output | Messages | Notifications

	apt_no [PK] character varying (10)	block_id [PK] integer	bedrooms integer	type character varying (10)	area integer	floor integer	address character varying (50)	owner_id character varying (38)
1	412	33028789	4	Penthouse	2250	4	1170 Kessler Vista	Of230bd53-f96a-49f2-ad95-dc77d1d7854e
2	101	39169924	2	Apartment	1200	3	301 Charlotte Island	O8b95ad9b-7082-410d-a342-adf2a7ba90...
3	102	39169924	2	Apartment	1200	3	301 Charlotte Island	Ob6c977a5-5845-4d7f-b2b5-3df5eba282ce
4	103	39169924	2	Apartment	1200	3	301 Charlotte Island	O40740fe1-1371-41f6-b006-de3e84c9c07b
5	201	39169924	2	Apartment	1200	2	301 Charlotte Island	O76ac053c-0e2f-4e5d-bf43-5d1442e89162
6	111	34543390	3	Apartment	1300	1	08546 Orin Corner	O8b95ad9b-7082-410d-a342-adf2a7ba90...
7	112	34543390	3	Apartment	1300	1	08546 Orin Corner	O76ac053c-0e2f-4e5d-bf43-5d1442e89162
8	101	41251798	1	Apartment	800	1	0846 Korbin Burg	O40740fe1-1371-41f6-b006-de3e84c9c07b
9	102	41251798	1	Apartment	800	1	0846 Korbin Burg	Ob6c977a5-5845-4d7f-b2b5-3df5eba282ce
10	408	33028789	4	Penthouse	2000	4	1170 Kessler Vista	Oe567f8a6-dce2-4b2e-845b-e49bbdde82...
11	404	33028789	1	Penthouse	1500	4	1170 Kessler Vista	Ob6c977a5-5845-4d7f-b2b5-3df5eba282ce

6. Tenant Table

```
select * from tenant;
```

	tenant_id [PK] character varying (38)	name character	ssn character varying (9)	age integer	perm_address character varying (50)	apt_no character varying (10)	email character varying (50)	phone_no character varying (15)	block_id integer
1	T897dfba1-37af-448d-ba1e-0e0763e2dea3	Rebekah	292366311	[null]	63239 Hirthe Villages	201	Lilly40@hotmail.com	+915033244800	39169924
2	T719d6e21-e213-4883-8bec-a7bd07d2a8f9	Eddie	292649311	[null]	684 Otho Falls	111	Caesar.Howe87@yahoo.com	+83154540238	34543390
3	Tdafe6666-12ab-4425-8c85-5570f850c4f5	Efren	192669311	[null]	58199 Rosenbaum Ca...	112	Rachael.Jacobson67@gmail.com	+622547293796	34543390
4	Ta076e342-48cd-497f-bd84-95868382cede	Joy	192669311	[null]	38939 Mayert Neck	102	Lois.Klocko@yahoo.com	+977335304525	41251798
5	Ta48d457d-cf25-4c70-9966-7e6df868e51a	Buck	192669311	[null]	211 Raynor Crossroad	101	Dudley.Lubowitz@yahoo.com	+198453726326	41251798
6	Te08f6e81-c550-46a0-b523-9779fd851ad6	Haley	879654321	[null]	9943 Cecile Crossroad	412	Cynthia36@yahoo.com	+16739084567	33028789
7	Te6b3a611-d92d-4d08-99d9-27900ee1d25b	Lavada	874254321	[null]	12669 Catalina Light	408	Kathryn.Rolfson@yahoo.com	+42890375689	33028789
8	Tc55d5b49-15f8-4074-9b14-c27a7aa87322	Janiya	924254321	[null]	730 Kristopher Views	101	Kurt92@hotmail.com	+71984562130	39169924
9	Tafcb0515-94e5-4355-9596-1773985dbb59	Giovani	924222311	[null]	23973 Trace Via	102	Hardy1@yahoo.com	+59321768450	39169924
10	Tfccaf0af-14a4-42f0-94b0-b27bddb9a7ea	Chandler	297652311	[null]	1920 Wyman Lodge	103	Amira92@hotmail.com	+82150693724	39169924
11	Tbdf13862-11c8-45d8-8306-a7661731a008	Brooklyn	292366311	[null]	86687 Reginald Burg	201	Dolly41@yahoo.com	+95431027856	39169924

7. Apartment_application Table

```
select * from apartment_application;
```

	email [PK] character varying (50)	phone character varying (15)	name character	owner_id character varying (38)	apt_no character varying (10)	addresss character varying (50)	block_id integer
1	Thad.Mante@hotmail.com	+83644865990	Tillman	Ob6c977a5-5845-4d7f-b2b5-3df5eba282ce	404	1170 Kessler Vista	33028789
2	Rylan.Pacocha@gmail.com	+868378754563	Ally	Ob6c977a5-5845-4d7f-b2b5-3df5eba282ce	404	1170 Kessler Vista	33028789
3	Holly.Schroeder@hotmail.com	+216464805769	Jimmie	Ob6c977a5-5845-4d7f-b2b5-3df5eba282ce	404	1170 Kessler Vista	33028789
4	Anya_Jaskolski@gmail.com	+253347935843	Gerald	Ob6c977a5-5845-4d7f-b2b5-3df5eba282ce	102	301 Charlotte Island	39169924
5	Ara.Lang89@yahoo.com	+36668005977	Axel	Ob6c977a5-5845-4d7f-b2b5-3df5eba282ce	102	301 Charlotte Island	39169924
6	Tania.Mraz@yahoo.com	+78889642031	Daren	O40740fe1-1371-41f6-b006-de3e84c9c07b	103	301 Charlotte Island	39169924
7	Herminia97@gmail.com	+778957595109	Allan	O40740fe1-1371-41f6-b006-de3e84c9c07b	103	301 Charlotte Island	39169924
8	Jude21@gmail.com	+469589335008	Carmel	O40740fe1-1371-41f6-b006-de3e84c9c07b	103	301 Charlotte Island	39169924
9	Patrick.Rolfson@gmail.com	+287884146791	Makenzie	O40740fe1-1371-41f6-b006-de3e84c9c07b	103	301 Charlotte Island	39169924
10	Adriana_OHara17@yahoo.com	+518182125181	Mallie	O40740fe1-1371-41f6-b006-de3e84c9c07b	101	0846 Korbin Burg	41251798
11	Josefa67@hotmail.com	+483446602218	Clara	O40740fe1-1371-41f6-b006-de3e84c9c07b	101	0846 Korbin Burg	41251798

8. Complaint Table

Query | Query History

```
1 select * from complaint;
```

Data Output | Messages | Notifications

	complaint_id [PK] character varying (38)	complaint_description character varying (100)	complaint_date date	emp_id character varying (38)	tenant_id character varying (38)
1	CP07bf3e8f-6552-423d-9550-b686347f1653	Ceiling Dripping water	2024-04-14	[null]	Tca3aac71-9e61-4b35-a12a-de36f901c3eb
2	CP66cb8865-b91d-44ae-acb3-09ca4bb77e78	Toilet Flush issue	2024-04-14	[null]	Tca3aac71-9e61-4b35-a12a-de36f901c3eb
3	CP36e64036-4a87-4e2f-8ae1-0026ecfa20e6	door squeky	2024-04-14	[null]	Tca3aac71-9e61-4b35-a12a-de36f901c3eb
4	CPfd463548-1865-47bf-a324-6d8baa369a43	Kitchen Exhaust Not Working	2024-04-14	[null]	Tca3aac71-9e61-4b35-a12a-de36f901c3eb
5	CP8cb85e93-1e5f-4b29-b9fa-05f367aa900f	Window is cracked	2024-04-14	[null]	Tca3aac71-9e61-4b35-a12a-de36f901c3eb
6	CP8b28ce5c-e968-4ad6-8ae2-03ccae5ff894	water is not clear	2024-04-14	[null]	Tca3aac71-9e61-4b35-a12a-de36f901c3eb
7	CPb448fb88-a612-4dc4-b81c-42b0c9f4209f	other complaint	2024-04-14	[null]	Tca3aac71-9e61-4b35-a12a-de36f901c3eb
8	CP41b7cdc7-6f11-4789-a33f-7c002023f1a9	other complaint	2024-04-14	[null]	Tca3aac71-9e61-4b35-a12a-de36f901c3eb
9	CP7694a68b-022b-48f3-b027-4529fa5b13e0	Regular Maintainence	2024-04-14	[null]	Tca3aac71-9e61-4b35-a12a-de36f901c3eb
10	CPa499cfcf-f402-4f1d-983d-a82e97d5925a	Regular Maintainence	2024-04-14	[null]	Tca3aac71-9e61-4b35-a12a-de36f901c3eb
11	CP4de78609-255e-48df-ba71-703ed0422218	Regular Maintainence	2024-04-14	[null]	Tca3aac71-9e61-4b35-a12a-de36f901c3eb

9. Tenant_Contact Table

Query | Query History

```
1
2  select * from tenant_contact;
3
4
```

Data Output | Messages | Notifications

	tenant_id [PK] character varying (38)	phone [PK] character varying (15)
1	Te08f6e81-c550-46a0-b523-9779fd851ad6	+64543996676
2	Te6b3a611-d92d-4d08-99d9-27900ee1d2...	+496079992634
3	Tc55d5b49-15f8-4074-9b14-c27a7aa873...	+576037012984
4	Tafcb0515-94e5-4355-9596-1773985dbb...	+66054258725
5	Tfccaf0af-14a4-42f0-94b0-b27bddb9a7ea	+105112177954
6	Tbdf13862-11c8-45d8-8306-a7661731a0...	+332619071034
7	T897dfba1-37af-448d-ba1e-0e0763e2dea3	+915033244800
8	T719d6e21-e213-4883-8bec-a7bd07d2a8...	+83154540238
9	Tdafe6666-12ab-4425-8c85-5570f850c4f5	+622547293796
10	Ta076e342-48cd-497f-bd84-95868382cede	+977335304525
11	Ta48d457d-cf25-4c70-9966-7e6df868e51a	+198453726326

10. Rents Table

Query | Query History

```
1  select * from rents;
```

Data Output | Messages | Notifications

	tenant_id [PK] character varying (38)	owner_id [PK] character varying (38)	rent_amount numeric (5,2)
1	T897dfba1-37af-448d-ba1e-0e0763e2dea3	O76ac053c-0e2f-4e5d-bf43-5d1442e89162	120.00
2	T719d6e21-e213-4883-8bec-a7bd07d2a8f9	O8b95ad9b-7082-410d-a342-adf2a7ba90...	120.00
3	Tdafe6666-12ab-4425-8c85-5570f850c4f5	O76ac053c-0e2f-4e5d-bf43-5d1442e89162	280.00
4	Ta076e342-48cd-497f-bd84-95868382cede	Ob6c977a5-5845-4d7f-b2b5-3df5eba282ce	350.00
5	Te08f6e81-c550-46a0-b523-9779fd851ad6	Of230bd53-f96a-49f2-ad95-dc77d1d7854e	190.00
6	Te6b3a611-d92d-4d08-99d9-27900ee1d2...	Oe567f8a6-dce2-4b2e-845b-e49bbdde82...	190.00
7	Tc55d5b49-15f8-4074-9b14-c27a7aa87322	O8b95ad9b-7082-410d-a342-adf2a7ba90...	225.00
8	Tafcb0515-94e5-4355-9596-1773985dbb59	Ob6c977a5-5845-4d7f-b2b5-3df5eba282ce	500.00
9	Tbdf13862-11c8-45d8-8306-a7661731a0...	O76ac053c-0e2f-4e5d-bf43-5d1442e89162	500.00
10	Tca3aac71-9e61-4b35-a12a-de36f901c3eb	O40740fe1-1371-41f6-b006-de3e84c9c07b	450.00

11. Maintenance_staff Table

Query | Query History

```
1  select * from maintenance_staff;
```

Data Output | Messages | Notifications

	emp_id [PK] character varying (38)	name character	phone character varying (15)	shift_timings character varying (50)	contract_length character varying (20)	role character varying (15)	email character varying (50)
1	E8d27048a-3493-4964-a0f5-31759ba54587	majdoor	+11234567891	9:00 AM - 5:00 PM	20D	Employee	employee@gmail.com
2	Ecdc336cd-87e2-4bc6-8004-d6833490dc11	Jensen	+586295475243	9:00 AM - 5:00 PM	20D	Employee	Verdie28@gmail.com
3	Ea58cec09-788c-4335-b79e-2d0d523d62b8	Gianni	+916234108267	9:00 AM - 5:00 PM	15D	Employee	Ruben_OKeefe@hotmail.com
4	E2548435c-be3d-4a81-a851-92327112623c	Baron	+983844158382	9:00 AM - 5:00 PM	15D	Employee	Rowland_Olson9@hotmail.com
5	Ee0178007-144d-47b1-8f1c-46d410489191	Mellie	+568062589473	9:00 AM - 5:00 PM	35D	Employee	Trace27@hotmail.com
6	E91e31e26-ff8e-4a9a-9b17-81063ef16ae6	Reta	+925753977207	9:00 AM - 5:00 PM	24HR	Employee	Celestino_Stracke78@hotmail.com
7	E892c5e74-b87e-40f0-b31e-fcb111c9efba	Josie	+887136289942	9:00 AM - 5:00 PM	50D	Employee	Judy.Schoen@hotmail.com
8	Ed187b20a-03b7-462b-9850-6e37bfdec262	Ernie	+797489143458	9:00 AM - 5:00 PM	2Y	Employee	Deja.Lehner@yahoo.com
9	E322bde1c-3056-42b5-bea7-d68ee2139dfe	Cristobal	+119564275407	9:00 AM - 5:00 PM	5Y	Employee	Lamont.Wolff9@gmail.com
10	Ed09e6b87-9dc3-4ce9-bca4-925db18ee09e	Maryam	+214117367518	9:00 AM - 5:00 PM	65D	Employee	Valentine.Reynolds32@yahoo.com

12. Manages Table

Query | Query History

```
1 SELECT * FROM public.manages
2 ORDER BY emp_id ASC, block_id ASC
```

Data Output | Messages | Notifications

	emp_id [PK] character varying (38)	block_id [PK] integer
1	E2548435c-be3d-4a81-a851-9232711262...	33028789
2	E322bde1c-3056-42b5-bea7-d68ee2139dfe	33028789
3	E322bde1c-3056-42b5-bea7-d68ee2139dfe	52505100
4	E892c5e74-b87e-40f0-b31e-fcb111c9efba	33028789
5	E892c5e74-b87e-40f0-b31e-fcb111c9efba	42509553
6	E892c5e74-b87e-40f0-b31e-fcb111c9efba	52505100
7	E8d27048a-3493-4964-a0f5-31759ba54587	42509553
8	E91e31e26-ff8e-4a9a-9b17-81063ef16ae6	93702634
9	Ecdc336cd-87e2-4bc6-8004-d6833490dc11	42509553
10	Ed187b20a-03b7-462b-9850-6e37bfdec262	42509553

Front End Implementation

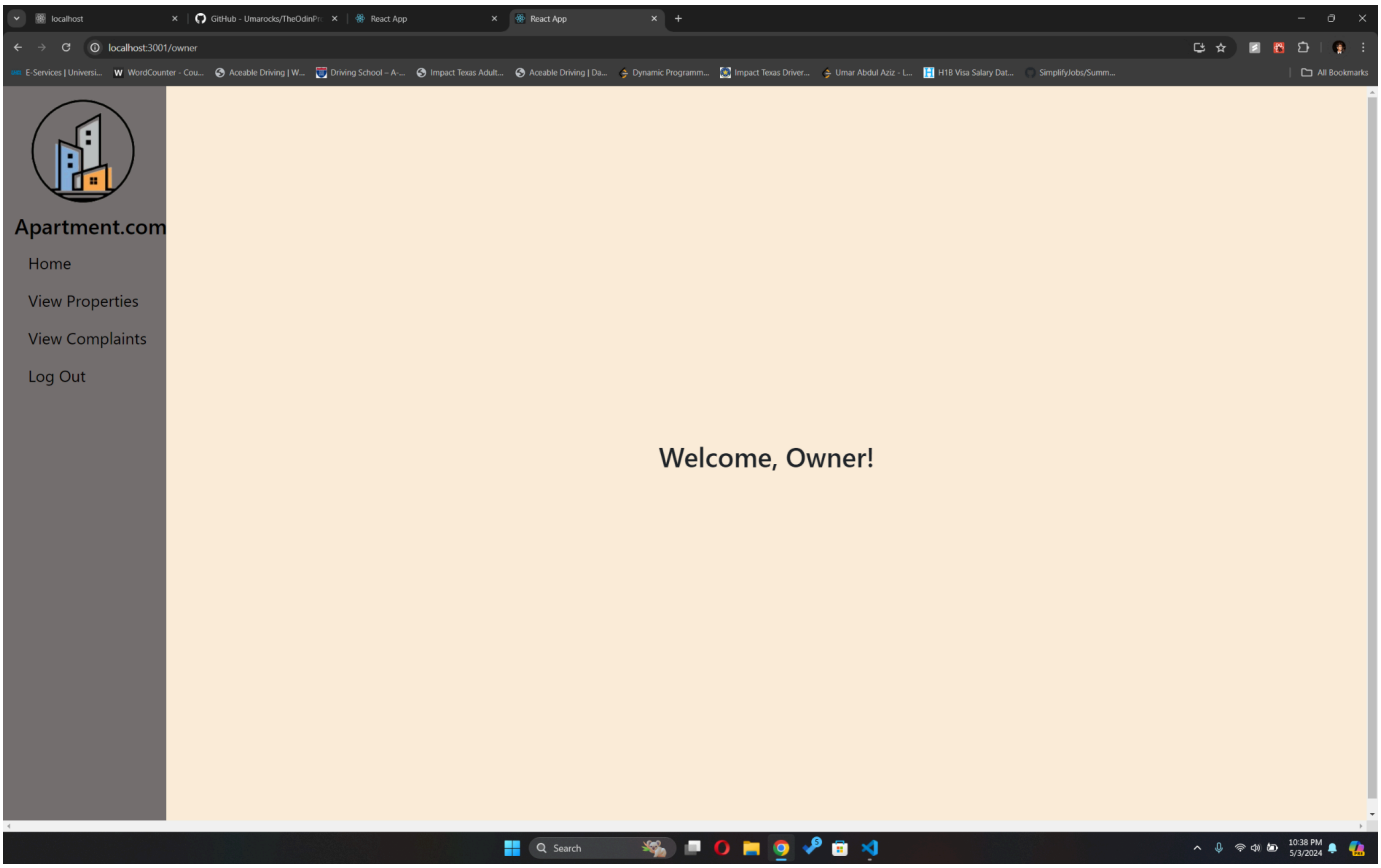
Owner Home Page:

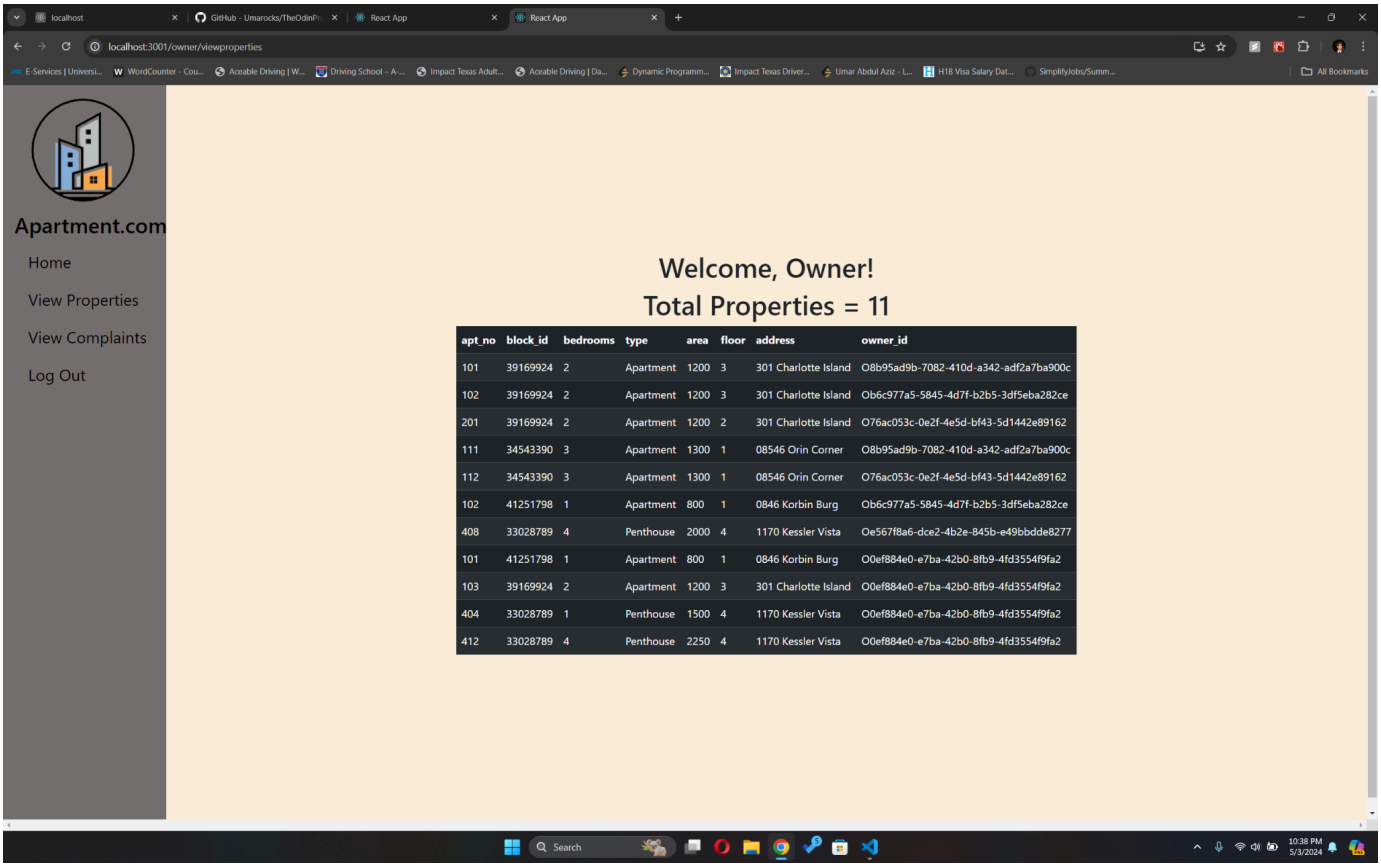
The screenshot displays the home page for an "Owner" user on a property management website, labeled "Apartment.com."

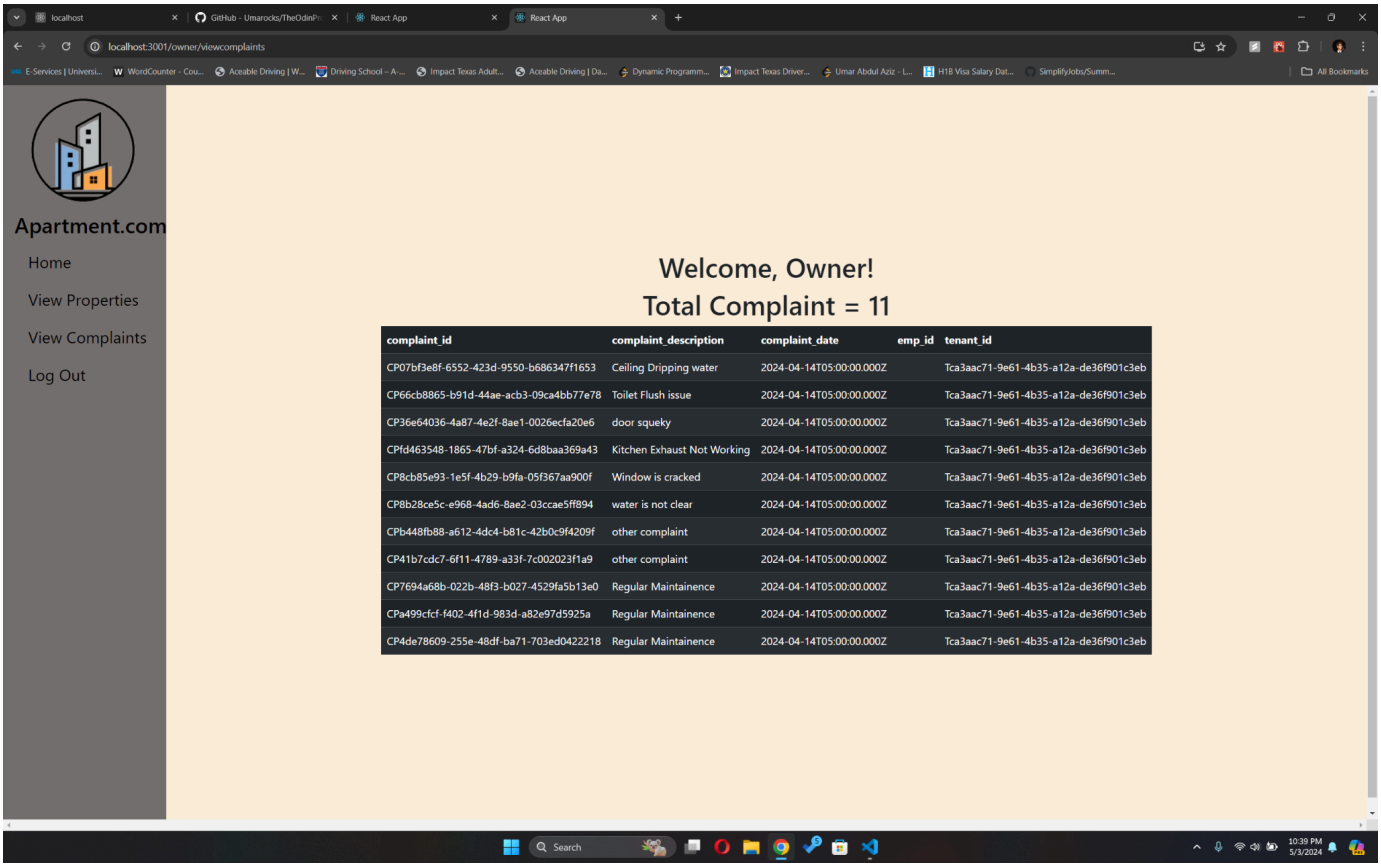
In the main area of the page, there is a prominent welcome message saying "Welcome, Owner!" This indicates that the page is personalized for the property owner and confirms that the user is logged in as an owner.

There is a navigation menu with the following options:

- Home
- View Properties
- View Complaints
- Log Out

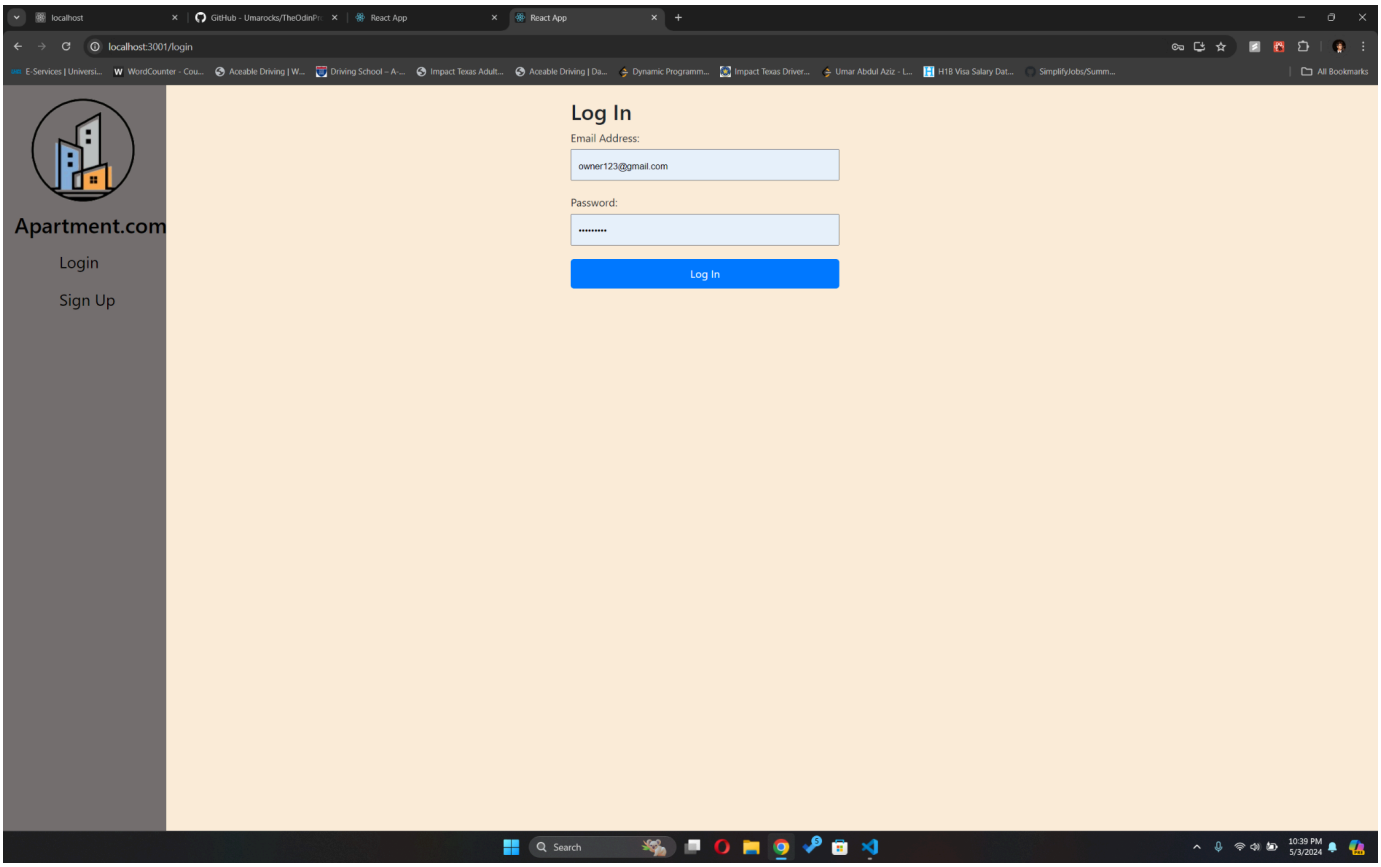



localhost
React App
localhost:3001/owner/viewproperties
Apartment.com
Home
View Properties
View Complaints
Log Out
Welcome, Owner!
Total Properties = 11
apt_no block_id bedrooms type area floor address owner_id
101 39169924 2 Apartment 1200 3 301 Charlotte Island O8b95ad9b-7082-410d-a342-adf2a7ba900c
102 39169924 2 Apartment 1200 3 301 Charlotte Island Ob6c977a5-5845-4d7f-b2b5-3df5eba282ce
201 39169924 2 Apartment 1200 2 301 Charlotte Island O76ac053c-0e2f-4e5d-bf43-5d1442e89162
111 34543390 3 Apartment 1300 1 08546 Orin Corner O8b95ad9b-7082-410d-a342-adf2a7ba900c
112 34543390 3 Apartment 1300 1 08546 Orin Corner O76ac053c-0e2f-4e5d-bf43-5d1442e89162
102 41251798 1 Apartment 800 1 0846 Korbin Burg Ob6c977a5-5845-4d7f-b2b5-3df5eba282ce
408 33028789 4 Penthouse 2000 4 1170 Kessler Vista Oe567f8a6-dce2-4b2e-845b-e49bbdde8277
101 41251798 1 Apartment 800 1 0846 Korbin Burg O0ef884e0-e7ba-42b0-8fb9-4fd3554f9fa2
103 39169924 2 Apartment 1200 3 301 Charlotte Island O0ef884e0-e7ba-42b0-8fb9-4fd3554f9fa2
404 33028789 1 Penthouse 1500 4 1170 Kessler Vista O0ef884e0-e7ba-42b0-8fb9-4fd3554f9fa2
412 33028789 4 Penthouse 2250 4 1170 Kessler Vista O0ef884e0-e7ba-42b0-8fb9-4fd3554f9fa2
Search
10:38 PM
5/3/2024


localhost
React App
localhost:3001/owner/viewcomplaints
Apartment.com
Home
View Properties
View Complaints
Log Out
Welcome, Owner!
Total Complaint = 11
complaint_id | complaint_description | complaint_date | emp_id | tenant_id
CP07bf3e8f-6552-423d-9550-b686347f1653 | Ceiling Dripping water | 2024-04-14T05:00:00.000Z | | Tca3aac71-9e61-4b35-a12a-de36f901c3eb
CP66cb8865-b91d-44ae-acb3-09ca4bb77e78 | Toilet Flush issue | 2024-04-14T05:00:00.000Z | | Tca3aac71-9e61-4b35-a12a-de36f901c3eb
CP36e64036-4a87-4e2f-8ae1-0026ecfa20e6 | door squeky | 2024-04-14T05:00:00.000Z | | Tca3aac71-9e61-4b35-a12a-de36f901c3eb
CPfd463548-1865-47bf-a324-6d8baa369a43 | Kitchen Exhaust Not Working | 2024-04-14T05:00:00.000Z | | Tca3aac71-9e61-4b35-a12a-de36f901c3eb
CP8cb85e93-1e5f-4b29-b9fa-05f367aa900f | Window is cracked | 2024-04-14T05:00:00.000Z | | Tca3aac71-9e61-4b35-a12a-de36f901c3eb
CP8b28ce5c-e968-4ad6-8ae2-03ccae5ff894 | water is not clear | 2024-04-14T05:00:00.000Z | | Tca3aac71-9e61-4b35-a12a-de36f901c3eb
CPb448fb88-a612-4dc4-b81c-42b0c9f4209f | other complaint | 2024-04-14T05:00:00.000Z | | Tca3aac71-9e61-4b35-a12a-de36f901c3eb
CP41b7cdc7-6f11-4789-a33f-7c002023f1a9 | other complaint | 2024-04-14T05:00:00.000Z | | Tca3aac71-9e61-4b35-a12a-de36f901c3eb
CP7694a68b-022b-48f3-b027-4529fa5b13e0 | Regular Maintainence | 2024-04-14T05:00:00.000Z | | Tca3aac71-9e61-4b35-a12a-de36f901c3eb
CPa499cfcf-f402-4f1d-983d-a82e97d5925a | Regular Maintainence | 2024-04-14T05:00:00.000Z | | Tca3aac71-9e61-4b35-a12a-de36f901c3eb
CP4de78609-255e-48df-ba71-703ed0422218 | Regular Maintainence | 2024-04-14T05:00:00.000Z | | Tca3aac71-9e61-4b35-a12a-de36f901c3eb
Search
10:39 PM
5/3/2024

Login Page

The screenshot shows the login page for the "Apartment.com" website, which is designed for users to access their accounts by entering their credentials.

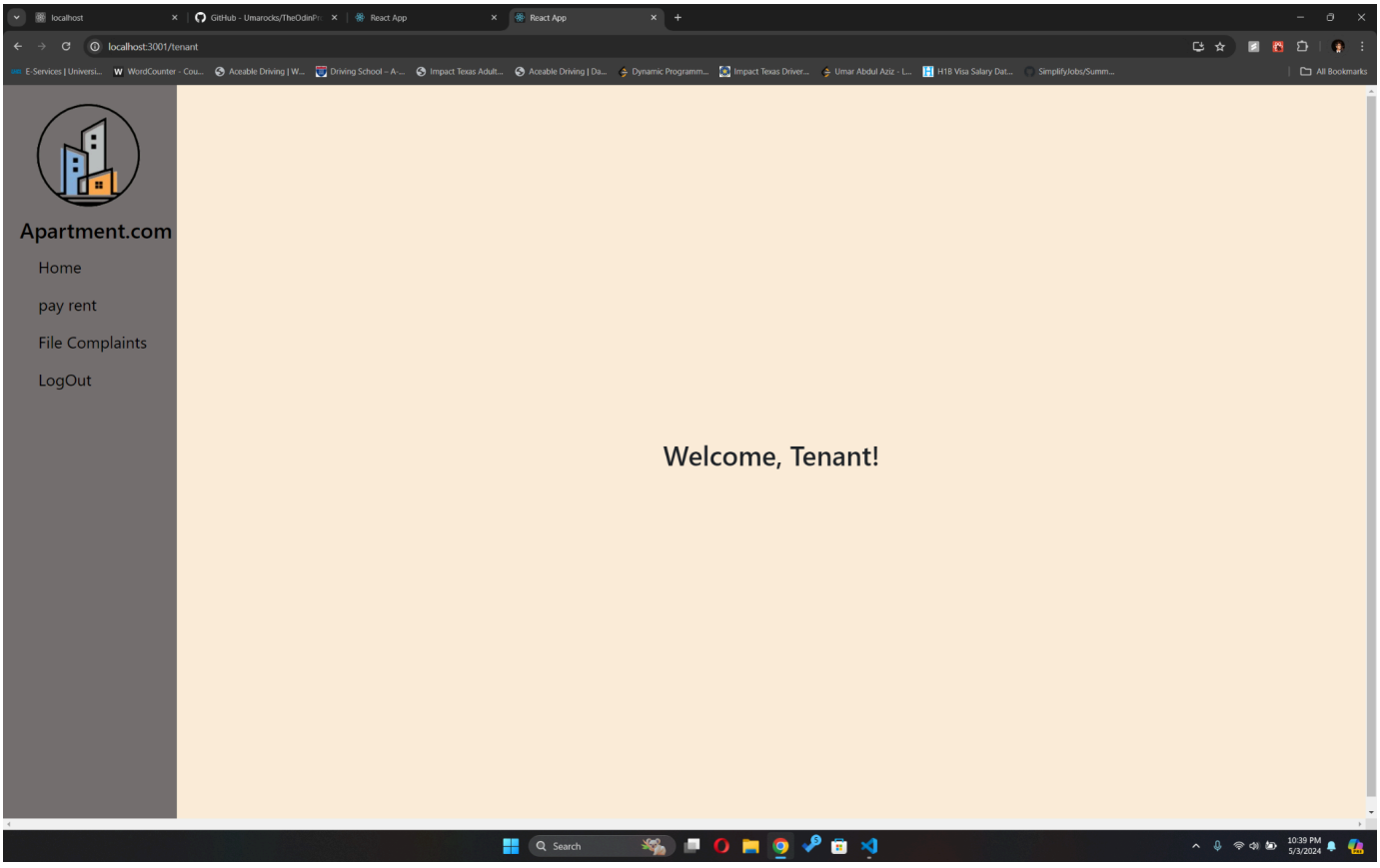


Tenant Home Page

In the main area of the page, there is a prominent welcome message saying "Welcome, Tenant!" This indicates that the page is personalized for the property owner and confirms that the user is logged in as a tenant.

There is a navigation menu with the following options:

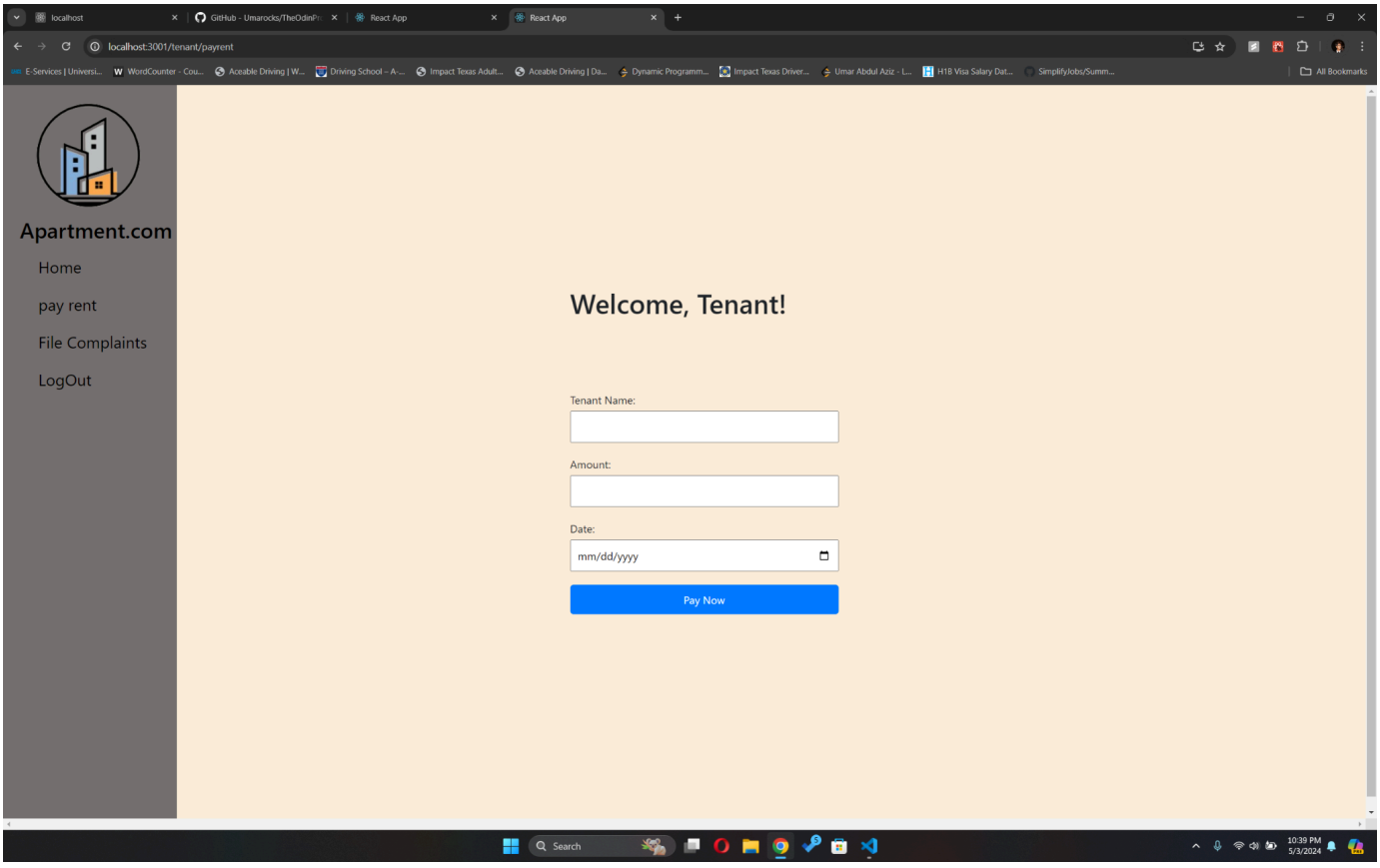
- **Home**
- **Pay Rent**
- **File Complaints**
- **Log Out**



The screenshot shows a "Pay Rent" page on the "Apartment.com" website, tailored for a tenant user.

There is a form to submit rent payments, which includes:

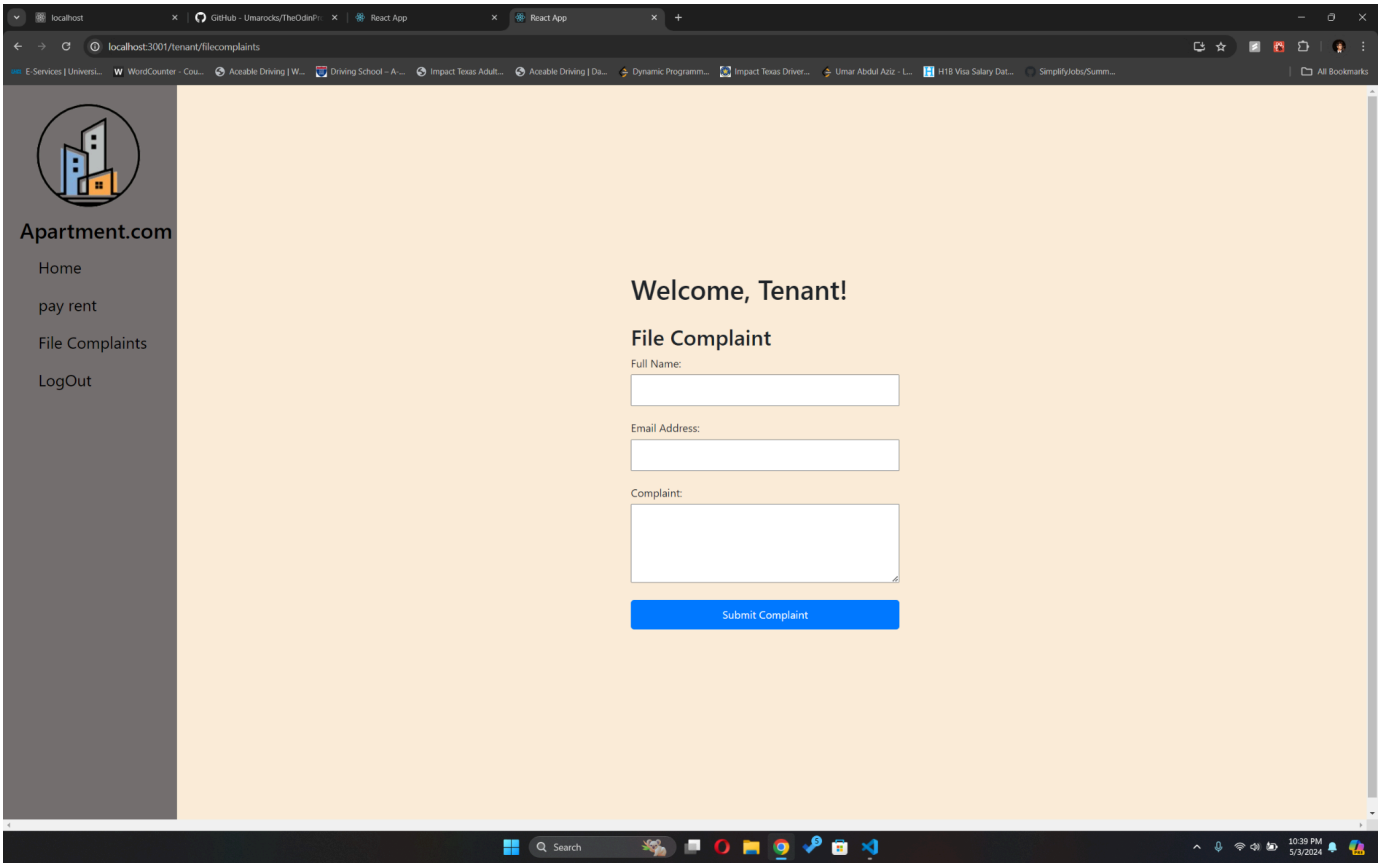
- **Tenant Name: A field for entering the tenant's name.**
- **Amount: A field for entering the rent amount.**
- **Date: A field with a calendar dropdown for selecting the date of the payment.**
- **Pay Now Button: A prominent blue button for submitting the payment details.**



The screenshot shows a "File Complaint" page on the "Apartment.com" website, tailored for a tenant user.

There is a form to submit rent payments, which includes:

- **Tenant Full Name: A field for entering the tenant's name.**
- **Email Address: A field for entering the email address of the Tenant.**
- **Complaint: A field to enter the details of the complaint.**
- **Submit Complaint Button: A button for submitting the complaint.**



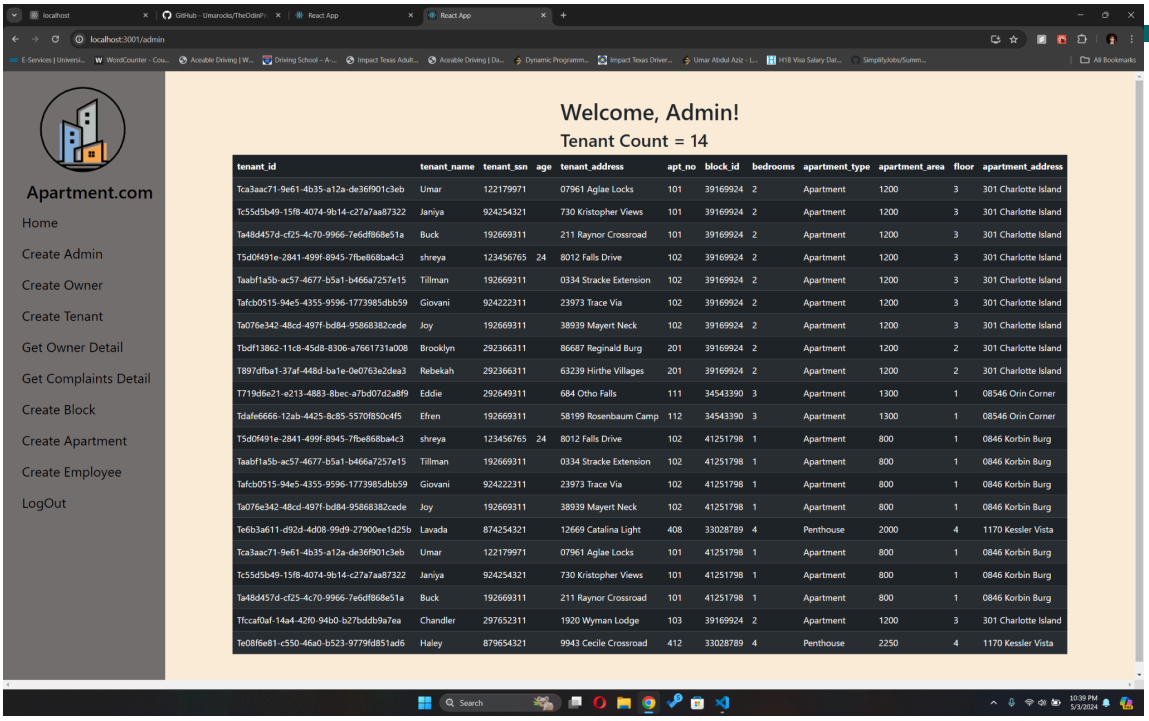
Admin Home Page

The screenshot displays the home page for an administrator on "Apartment.com." This page appears designed to provide the admin with a comprehensive overview and management capabilities for tenants and properties.

The left sidebar contains multiple navigation options, indicating the extensive functionalities available to the admin:

- **Home**
- **Create Admin**
- **Create Owner**
- **Create Tenant**
- **Get Owner Detail**
- **Get Complaints Detail**
- **Create Block**
- **Create Apartment**
- **Create Employee**
- **Log Out**

The main content area starts with a greeting, "Welcome, Admin!" followed by a notice that the current tenant count is 14, suggesting a summary or dashboard functionality.

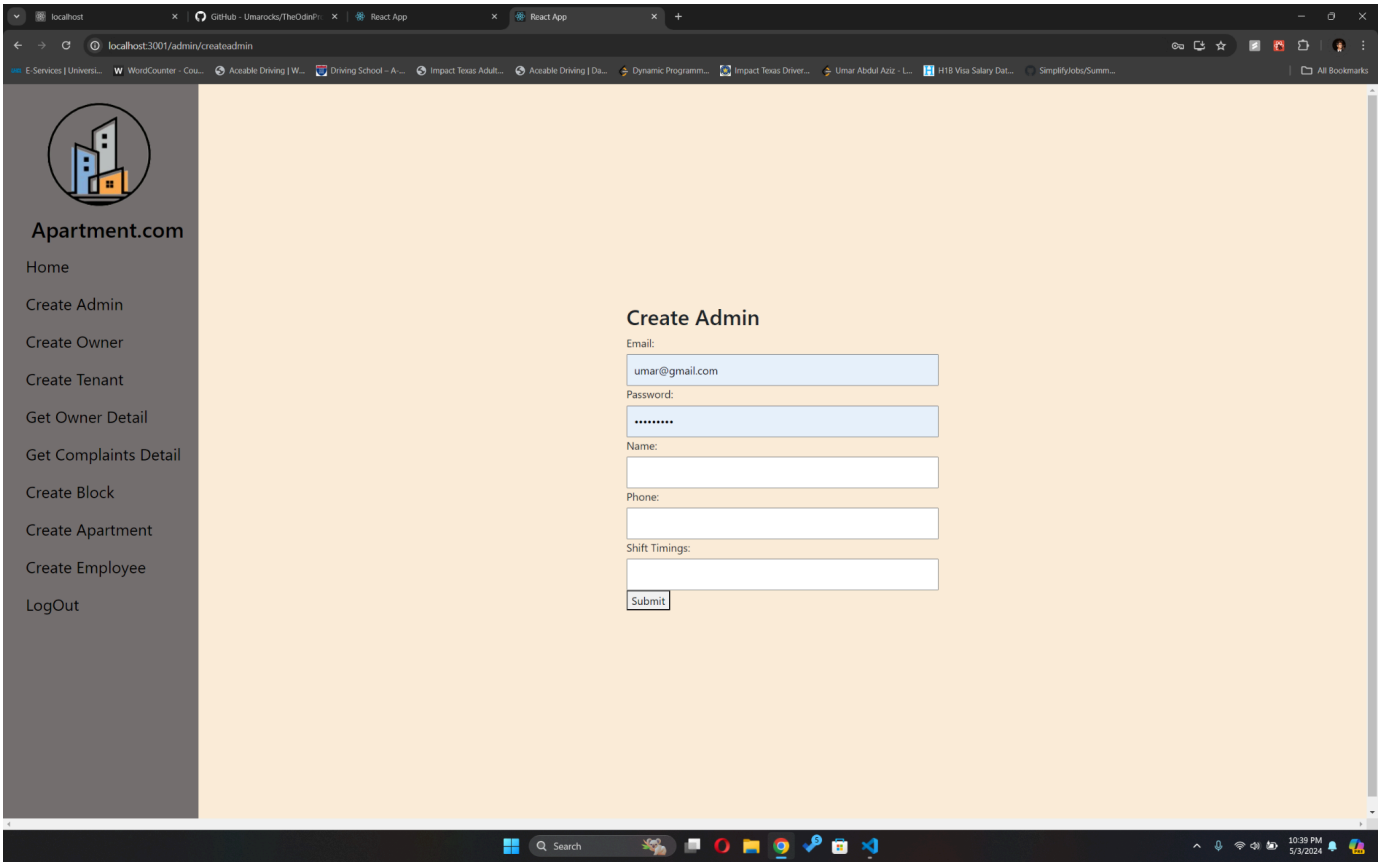


tenant_id	tenant_name	tenant_ssn	age	tenant_address	apt_no	block_id	bedrooms	apartment_type	apartment_area	floor	apartment_address
Tca3aac71-9e61-4b35-a12a-de36f901c3eb	Umar	122179971		07961 Aglae Locks	101	39169924	2	Apartment	1200	3	301 Charlotte Island
Tc55d5b49-15f8-4074-9b14-c27a7aa87322	Janiya	924254321		730 Kristopher Views	101	39169924	2	Apartment	1200	3	301 Charlotte Island
Ta48d457d-cf25-4c70-9966-7e6df868e51a	Buck	192669311		211 Raynor Crossroad	101	39169924	2	Apartment	1200	3	301 Charlotte Island
T5d0f491e-2841-499f-8945-7fbe868ba4c3	shreya	123456765	24	8012 Falls Drive	102	39169924	2	Apartment	1200	3	301 Charlotte Island
Taabf1a5b-ac57-4677-b5a1-b466a7257e15	Tillman	192669311		0334 Stracke Extension	102	39169924	2	Apartment	1200	3	301 Charlotte Island
Tafcb0515-94e5-4355-9596-1773985dbb59	Giovani	924222311		23973 Trace Via	102	39169924	2	Apartment	1200	3	301 Charlotte Island
Ta076e342-48cd-497f-bd84-95868382cede	Joy	192669311		38939 Mayert Neck	102	39169924	2	Apartment	1200	3	301 Charlotte Island
Tbdf13862-11c8-45d8-8306-a7661731a008	Brooklyn	292366311		86687 Reginald Burg	201	39169924	2	Apartment	1200	2	301 Charlotte Island
T897dfba1-37af-448d-ba1e-0e0763e2dea3	Rebekah	292366311		63239 Hirthe Villages	201	39169924	2	Apartment	1200	2	301 Charlotte Island
T719d6e21-e213-4883-8bec-a7bd07d2a8f9	Eddie	292649311		684 Otho Falls	111	34543390	3	Apartment	1300	1	08546 Orin Corner
Tdafe6666-12ab-4425-8c85-5570f850c4f5	Efren	192669311		58199 Rosenbaum Camp	112	34543390	3	Apartment	1300	1	08546 Orin Corner
T5d0f491e-2841-499f-8945-7fbe868ba4c3	shreya	123456765	24	8012 Falls Drive	102	41251798	1	Apartment	800	1	0846 Korbin Burg
Taabf1a5b-ac57-4677-b5a1-b466a7257e15	Tillman	192669311		0334 Stracke Extension	102	41251798	1	Apartment	800	1	0846 Korbin Burg
Tafcb0515-94e5-4355-9596-1773985dbb59	Giovani	924222311		23973 Trace Via	102	41251798	1	Apartment	800	1	0846 Korbin Burg
Ta076e342-48cd-497f-bd84-95868382cede	Joy	192669311		38939 Mayert Neck	102	41251798	1	Apartment	800	1	0846 Korbin Burg
Te6b3a611-d92d-4d08-99d9-27900ee1d25b	Lavada	874254321		12669 Catalina Light	408	33028789	4	Penthouse	2000	4	1170 Kessler Vista
Tca3aac71-9e61-4b35-a12a-de36f901c3eb	Umar	122179971		07961 Aglae Locks	101	41251798	1	Apartment	800	1	0846 Korbin Burg
Tc55d5b49-15f8-4074-9b14-c27a7aa87322	Janiya	924254321		730 Kristopher Views	101	41251798	1	Apartment	800	1	0846 Korbin Burg
Ta48d457d-cf25-4c70-9966-7e6df868e51a	Buck	192669311		211 Raynor Crossroad	101	41251798	1	Apartment	800	1	0846 Korbin Burg
Tfccaf0af-14a4-42f0-94b0-b27bddb9a7ea	Chandler	297652311		1920 Wyman Lodge	103	39169924	2	Apartment	1200	3	301 Charlotte Island
Te08f6e81-c550-46a0-b523-9779fd851ad6	Haley	879654321		9943 Cecile Crossroad	412	33028789	4	Penthouse	2250	4	1170 Kessler Vista

The screenshot displays the "Create Admin" page on the "Apartment.com" website, accessible via the administrator's dashboard. This page is designed for the creation of new admin accounts within the system.

The central area features a form with the following fields:

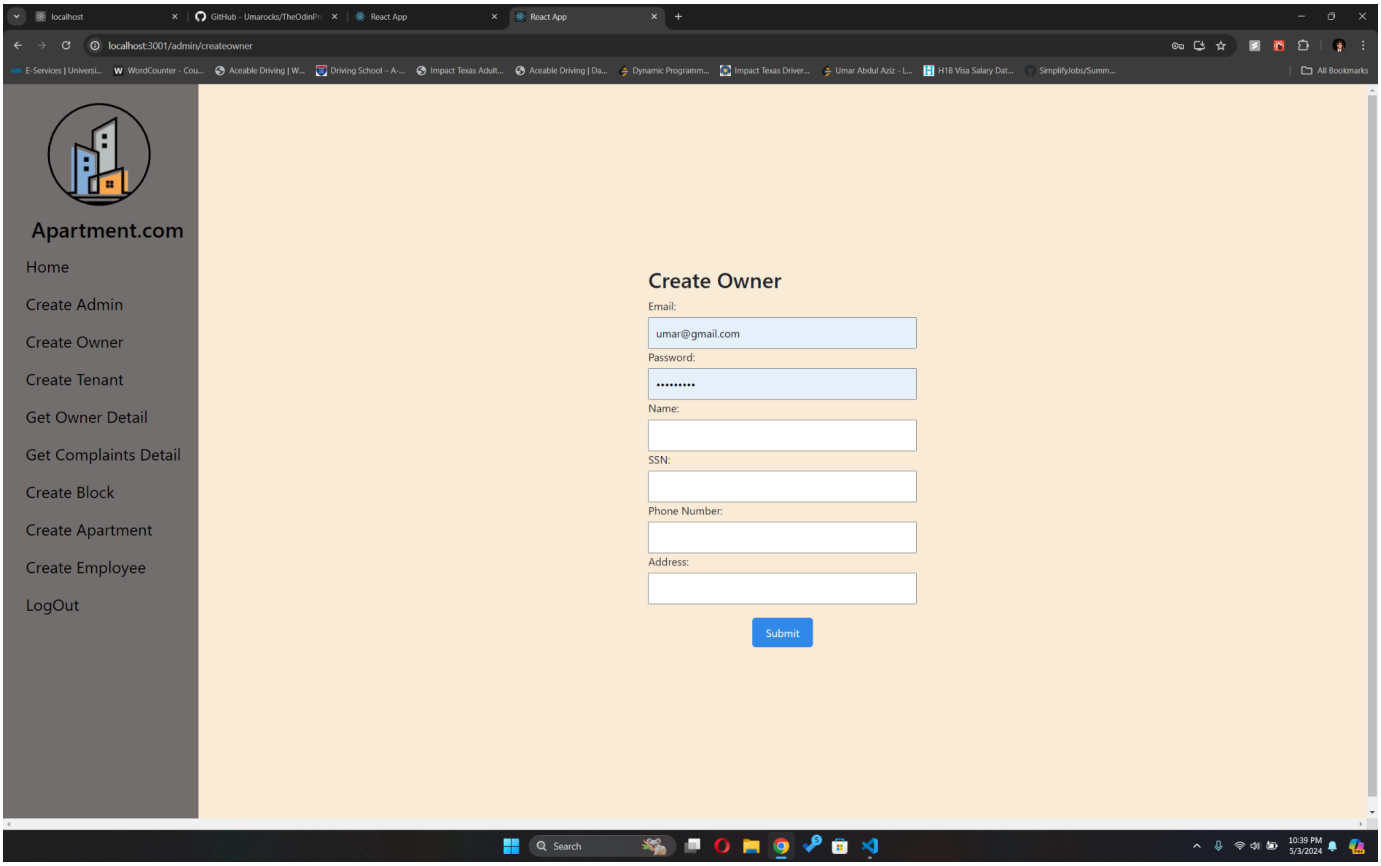
- **Email: To enter the email address of the new admin. This is typically used as the username for login purposes.**
- **Password: To set a password for the new admin account.**
- **Name: To enter the full name of the new admin.**
- **Phone: To provide a contact phone number for the admin.**
- **Shift Timings: To specify the working hours or shift timings for the admin.**
- **Submit Button: A blue button labeled "Submit" is used to save and create the new admin account.**



The screenshot displays the "Create Owner" page on the "Apartment.com" website, accessible via the administrator's dashboard. This page is designed for the addition of new owner accounts within the system.

The central area features a form with the following fields:

- **Email: To enter the email address of the new owner. This is typically used as the username for login purposes.**
- **Password: To set a password for the new owner account.**
- **Name: To enter the full name of the new owner.**
- **SSN : to enter the SSN of the new owner.**
- **Phone: To provide a contact phone number for the owner.**
- **Address: To provide address of the owner.**
- **Submit Button: A blue button labeled "Submit" is used to save and create the new owner account.**



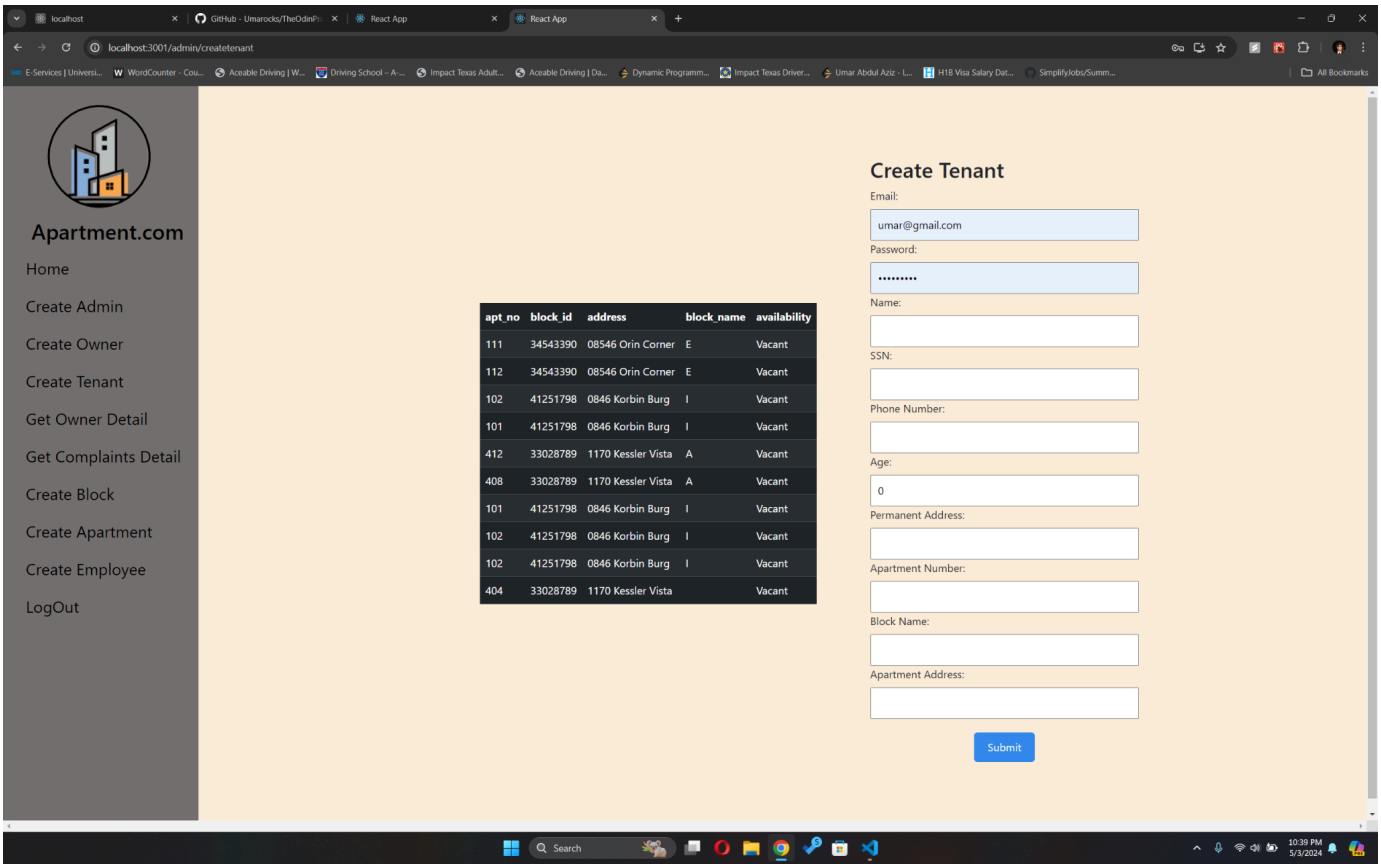
The screenshot displays the "Create Tenant" page on the "Apartment.com" website, accessible via the administrator's dashboard. This page is designed for the addition of new tenant accounts within the system.

The central area features a form with the following fields:

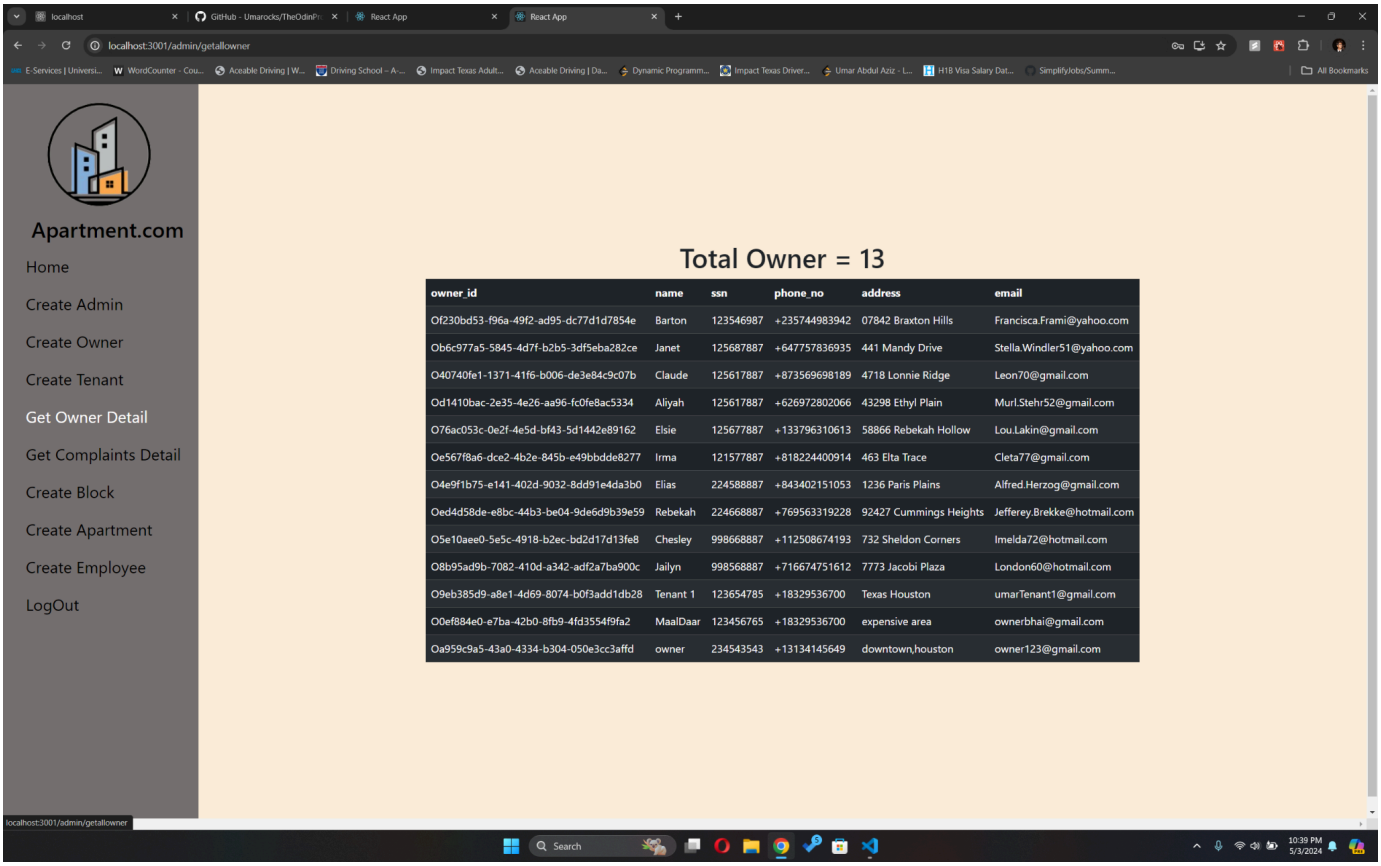
- **Email: To enter the email address of the new tenant. This is typically used as the username for login purposes.**
- **Password: To set a password for the new tenant account.**
- **Name: To enter the full name of the new tenant.**
- **SSN : to enter the SSN of the new tenant.**
- **Phone: To provide a contact phone number for the tenant.**
- **Age: To provide the age of the tenant.**
- **Permanent Address: To provide address of the owner.**
- **Apartment Number: To enter the leased apartment number.**
- **Block Name: To enter the block name which the apartment is part of.**

- **Submit Button: A blue button labeled "Submit" is used to save and create the new tenant account.**

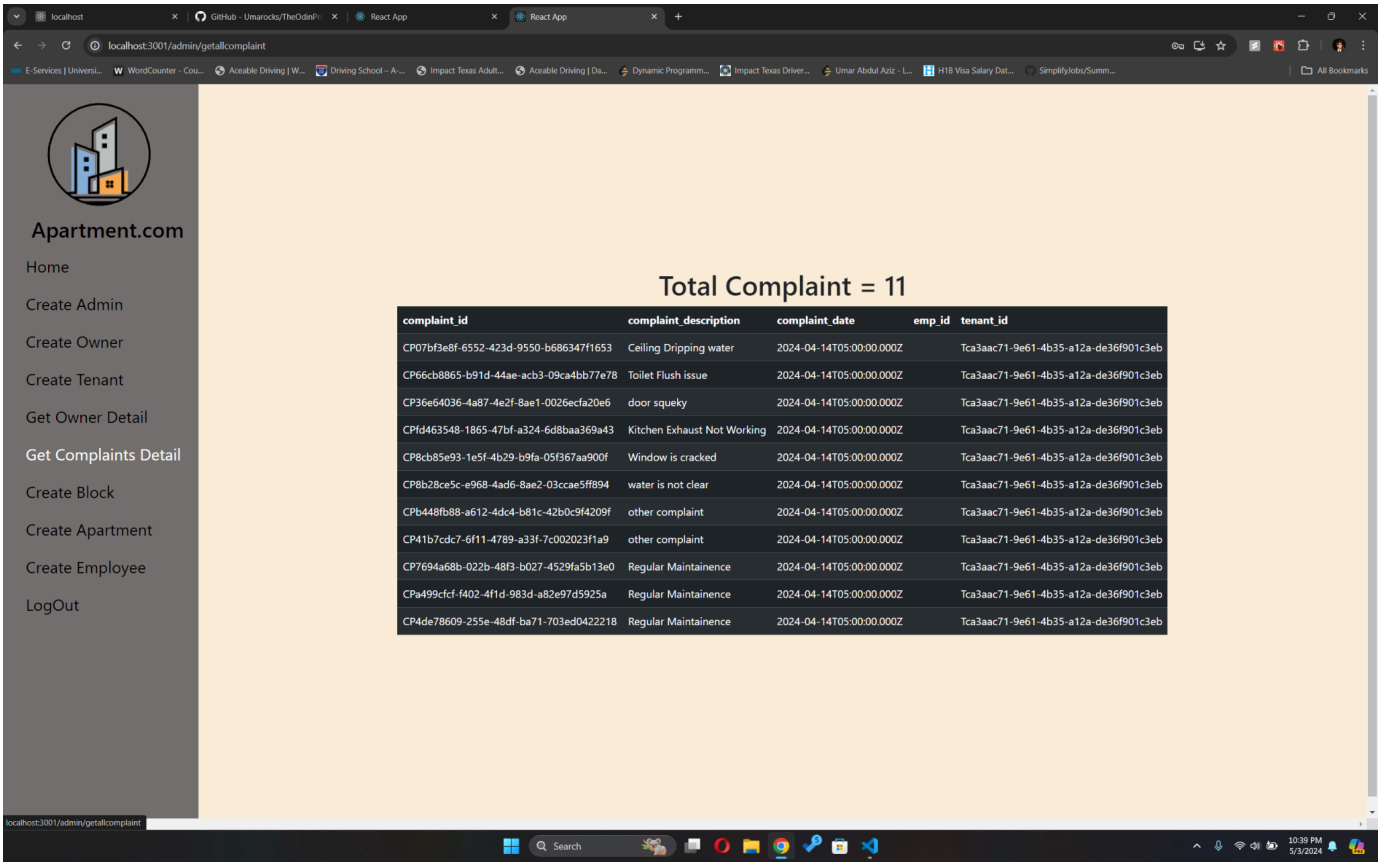
On the left side below the menu, there is a table showing the available apartments which includes columns for apartment number, block ID, address, block name, and availability status.



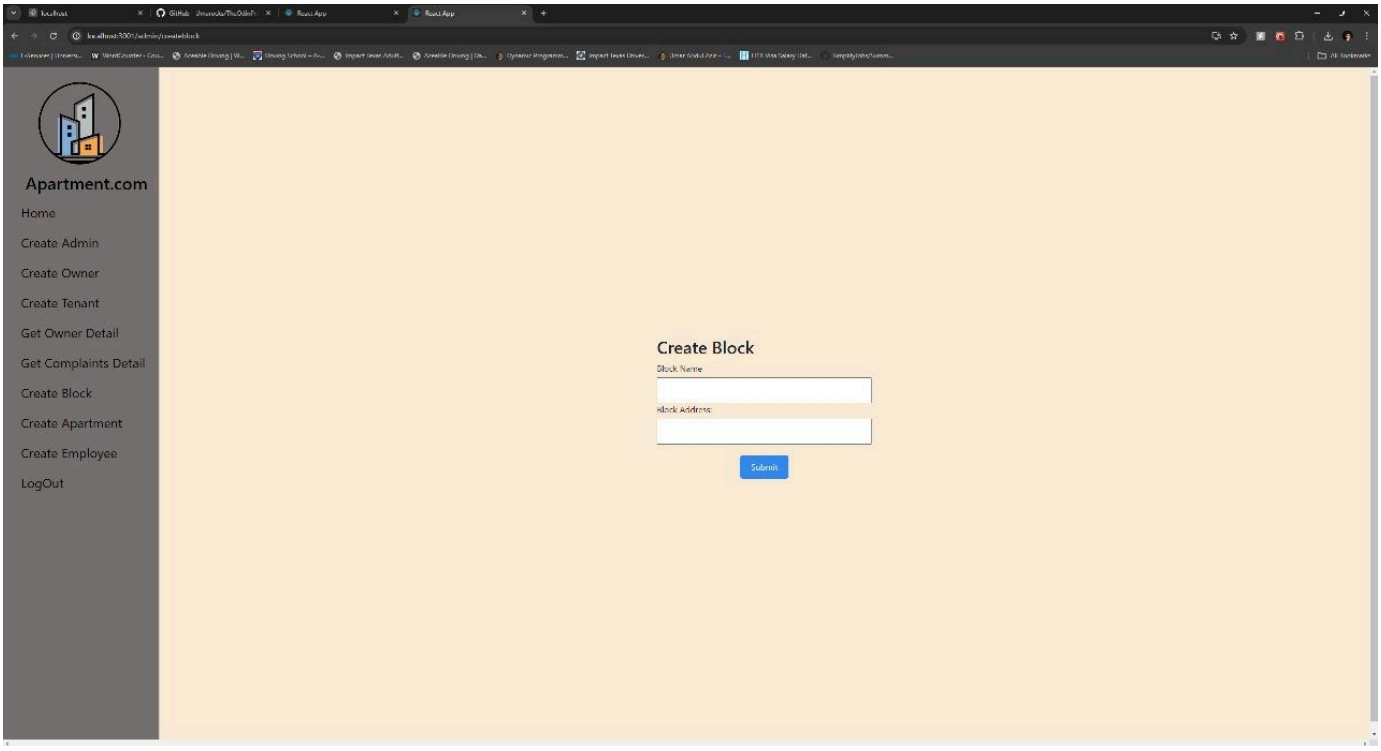
The screenshot displays the "Total Owner" page on the "Apartment.com" website, accessible via the administrator's dashboard. This page is designed for displaying the details of total number of Owners.



The screenshot displays the "Total Complainy" page on the "Apartment.com" website, accessible via the administrator's dashboard. This page is designed for displaying the details of the complaints lodged by the tenants.



The screenshot displays the "Create Block" page on the "Apartment.com" website, accessible via the administrator's dashboard. This page is designed for the addition of the new block which asks for the Block Name and Block Address.

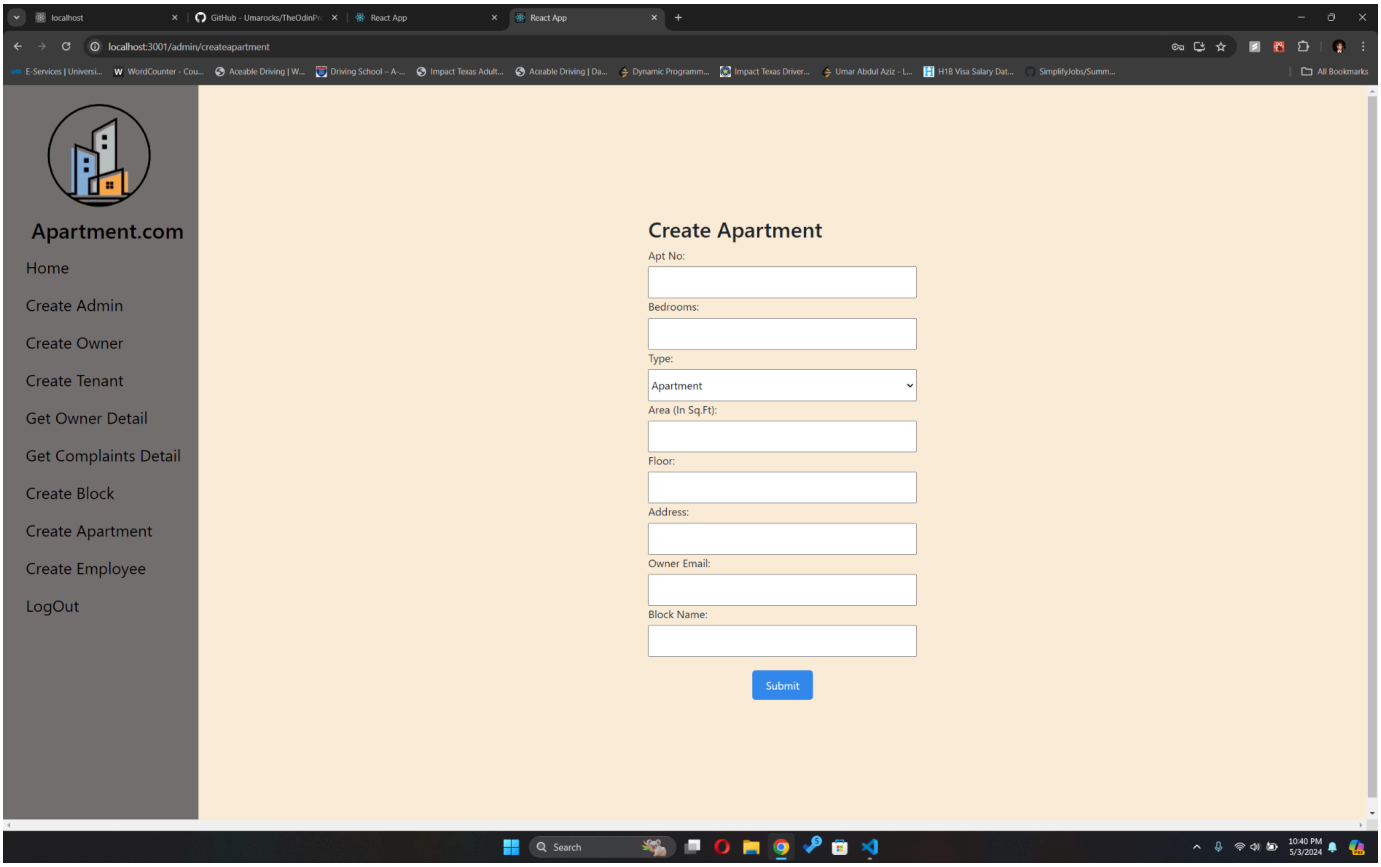


The screenshot displays the "Create Apartment" page on the "Apartment.com" website, accessible via the administrator's dashboard. This page is designed for

where an administrator can create new apartment listings in the system.

The central area features a form with the following fields:

- **Apt No: To enter the apartment number.**
- **Bedrooms: To specify the number of bedrooms.**
- **Type: Dropdown to select the type of the apartment (e.g., Apartment, Studio).**
- **Area (In Sq.Ft): To specify the size of the apartment.**
- **Floor: To input which floor the apartment is on.**
- **Address: To provide the full address of the apartment.**
- **Owner Email: To associate an apartment with a specific owner's email.**
- **Block Name: To associate the apartment with a specific block within the property.**
- **Submit Button: To save the apartment information in the system.**

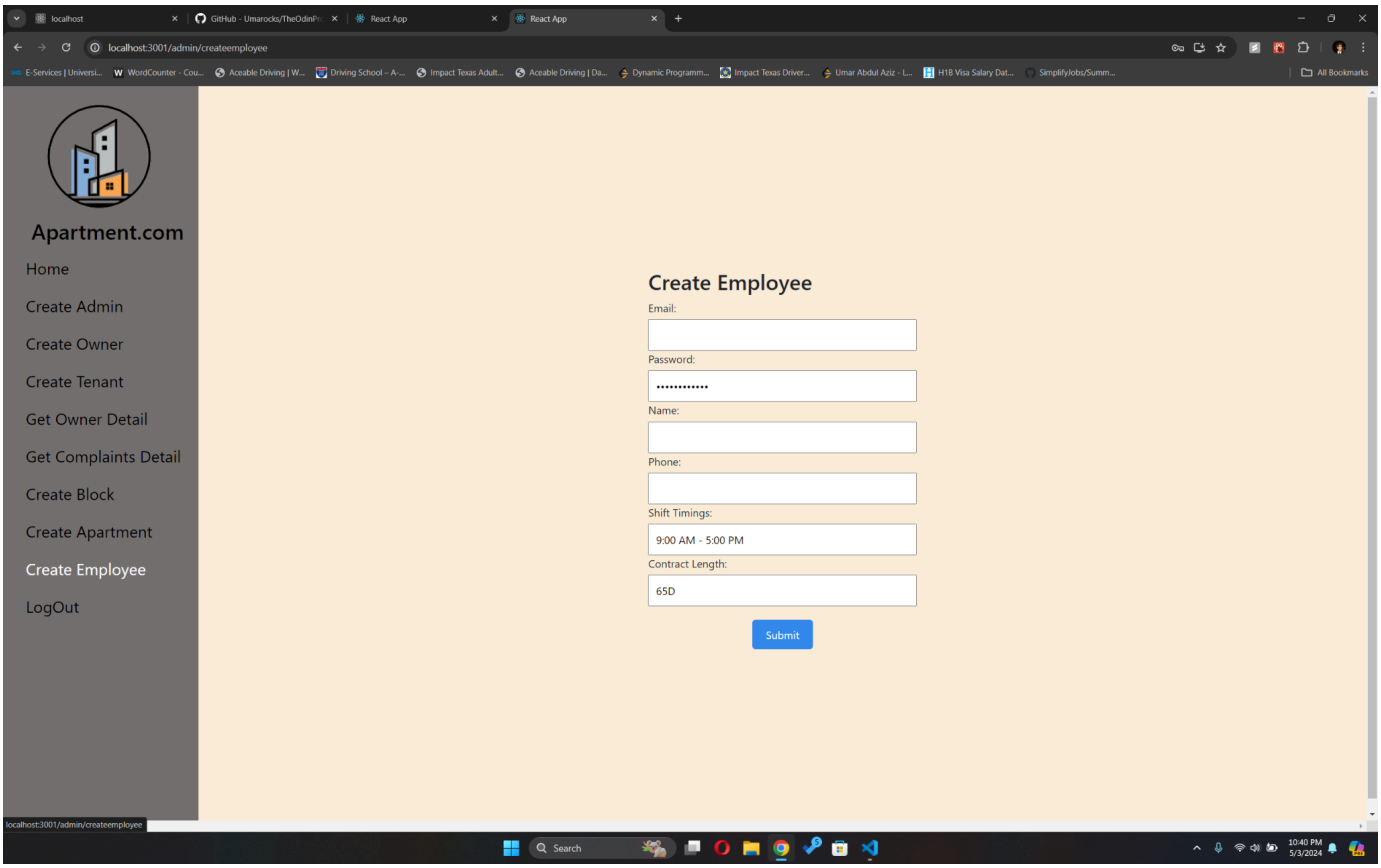


The screenshot displays the "Create Employee" page on the "Apartment.com" website, accessible via the administrator's dashboard. This page is designed for

where an administrator can add a new employee in the system.

The central area features a form with the following fields:

- **Email: For the employee's email address, which may also serve as their login username.**
- **Password: For setting up the employee's account password.**
- **Name: To enter the employee's full name.**
- **Phone: For the employee's contact number.**
- **Shift Timings: To specify the working hours of the employee, such as "9:00 AM - 5:00 PM".**
- **Contract Length: To specify the duration of the employment contract in days.**
- **Submit Button: To finalize and save the employee's data.**



Conclusion

In summary, our created Apartment Leasing Management System (ALMS) provides a comprehensive solution for efficiently handling the complexities of apartment leasing. This system improves the speed of financial transactions and complaint resolutions in addition to streamlining the maintenance of tenant data and apartment inventories. With capabilities including powerful relational schema, user-friendly interface, and comprehensive entity and connection designations, ALMS can meet the ever-changing demands of contemporary apartment buildings.

During the whole development process, our top priorities were user experience, data integrity, and security. Our database is kept consistent and dependable using cascading updates and removals, and sensitive data is protected from unwanted access using a role-based access control system.

Our team's goal is to continuously improve the system by considering customer input and changing needs. With its innovative blend of cutting-edge technology and useful features to enable smooth apartment management operations, we think ALMS will establish a new benchmark for property management systems.

Future developers and system administrators will be able to efficiently maintain and improve ALMS by utilizing our comprehensive documentation, which will support the system's adaptability and continued importance in the constantly evolving property management industry.

****