SYSTEM DOCUMENTATION

1. Project Information

Project Name: Luxur Management System
Student Name: Samuel Prada
Course: Web Development / Systems Engineering
Semester: 2025-2
Date: November 20, 2025

---

2. Short Project Description

Luxur Management System is a comprehensive web application designed to centralize and optimize the management of clients, properties, and contracts. Built with **Angular + Django**, the system provides:

- CRUD operations
- Secure authentication
- Role-based access
- Real-time validations
- Responsive SPA design

The goal is to streamline administrative workflows for real estate agencies.

---

3. System Architecture Overview

3.1 Architecture Description

The Luxur Management System follows a **client-server** architecture.

Frontend (Angular 17)

- SPA (Single Page Application)
- PrimeNG
- RxJS
- Angular Router
- HttpClient

Backend (Django + DRF)

- RESTful API
- Token authentication
- Business logic & data persistence

Database

- MySQL 8 / PostgreSQL / SQLite

Authentication

- Token-based (JWT)

Communication

- JSON over HTTP/HTTPS

Architecture Pattern: MVC

Layer	Technology
Model	Django ORM
View	DRF views / viewsets
Controller	Angular components/services

3.2 Technologies Used

Frontend

- Angular 17+
- PrimeNG
- TypeScript 5
- HTML5 / CSS3
- RxJS

Backend

- Python 3.10+
- Django 4.x
- Django REST Framework
- Django CORS Headers

Tools

- Git
- npm
- pip
- Postman
- VSCode

---

4. Database Description

4.1 Design Principles

- **Entity Integrity:** Primary keys
- **Referential Integrity:** Foreign keys
- **Normalization:** Up to 3NF
- **Indexes:** For performance
- **Audit Trail:** Created_at / updated_at

4.2 ERD (Entity Relationship Diagram)

3.3 Logical Model

Entities and Attributes **User** - Stores system user information and authentication data - Controls access and permissions - Tracks user activity

Client - Represents customers/clients in the system - Contains contact and identification information - Linked to contracts

Property - Represents real estate properties - Contains property details and characteristics - Linked to contracts

Contract - Represents agreements between clients and properties - Contains terms, dates, and financial information - Links clients to properties

Business Rules

- A client must exist before creating a contract
- A property must exist before creating a contract
- A contract must reference both a valid client and property
- Deletion of client/property is restricted if active contracts exist
- Users must be authenticated to perform any operation
- Role-based permissions control CRUD operations

3.4 Physical Model (Tables)

Table: auth_user

Column	Type	PK/FK	Null	Default	Description
id	INTEGER	PK	NO	AUTO_INCREMENT	Unique user identifier
username	VARCHAR(150)		NO		Unique username for login
email	VARCHAR(254)		NO		User email address
password	VARCHAR(128)		NO		Hashed password

Column	Type	PK/FK	Null	Default	Description
first_name	VARCHAR(150)		YES	NULL	User's first name
last_name	VARCHAR(150)		YES	NULL	User's last name
is_active	BOOLEAN		NO	TRUE	Account active status
is_staff	BOOLEAN		NO	FALSE	Staff status
is_superuser	BOOLEAN		NO	FALSE	Superuser status
date_joined	DATETIME		NO	NOW()	Account creation date
last_login	DATETIME		YES	NULL	Last login timestamp

Indexes: - PRIMARY KEY (id) - UNIQUE INDEX (username) - UNIQUE INDEX (email)

Table: clients

Column	Type	PK/FK	Null	Default	Description
id	INTEGER	PK	NO	AUTO_INCREMENT	Unique client identifier
name	VARCHAR(200)		NO		Full name of the client
email	VARCHAR(254)		NO		Client email address
phone	VARCHAR(20)		NO		Contact phone number
address	TEXT		YES	NULL	Physical address
document	VARCHAR(50)		NO		ID document number
created_at	DATETIME		NO	NOW()	Record creation timestamp
updated_at	DATETIME		NO	NOW()	Last update timestamp
created_by_id	INTEGER	FK	YES	NULL	User who created record

Indexes: - PRIMARY KEY (id) - UNIQUE INDEX (email) - UNIQUE INDEX (document) - FOREIGN KEY (created_by_id) REFERENCES auth_user(id) - INDEX (created_at)

Constraints: - `email` must be valid format
- `phone` must be numeric
- `document` must be unique

Table: properties

Column	Type	PK/FK	Null	Default	Description
id	INTEGER	PK	NO	AUTO_INCREMENT	Unique property identifier
address	VARCHAR(255)		NO		Property address
type	VARCHAR(50)		NO		Property type
value	DECIMAL(12,2)		NO		Property value/price
area	DECIMAL(10,2)		NO		Area in square meters
rooms	INTEGER		YES	NULL	Number of rooms
status	VARCHAR(20)		NO	'available'	Property status

Column	Type	PK/FK	Null	Default	Description
description	TEXT		YES	NULL	Additional description
created_at	DATETIME		NO	NOW()	Record creation timestamp
updated_at	DATETIME		NO	NOW()	Last update timestamp
created_by_id	INTEGER	FK	YES	NULL	User who created record

Indexes: - PRIMARY KEY (id) - FOREIGN KEY (created_by_id) REFERENCES auth_user(id) - INDEX (status) - INDEX (type) - INDEX (created_at)

Constraints: - `value` must be > 0
- `area` must be > 0
- `rooms` must be >= 0
- `status` ENUM ('available', 'occupied', 'maintenance')

Table: contracts

Column	Type	PK/FK	Null	Default	Description
id	INTEGER	PK	NO	AUTO_INCREMENT	Unique contract identifier
client_id	INTEGER	FK	NO		Reference to client
property_id	INTEGER	FK	NO		Reference to property
start_date	DATE		NO		Contract start date
end_date	DATE		NO		Contract end date
value	DECIMAL(12,2)		NO		Contract value/rent
terms	TEXT		YES	NULL	Contract terms and conditions
status	VARCHAR(20)		NO	'active'	Contract status
created_at	DATETIME		NO	NOW()	Record creation timestamp
updated_at	DATETIME		NO	NOW()	Last update timestamp
created_by_id	INTEGER	FK	YES	NULL	User who created record

Indexes: - PRIMARY KEY (id) - FOREIGN KEY (client_id) REFERENCES clients(id) ON DELETE RESTRICT - FOREIGN KEY (property_id) REFERENCES properties(id) ON DELETE RESTRICT - FOREIGN KEY (created_by_id) REFERENCES auth_user(id) - INDEX (status) - INDEX (start_date, end_date) - UNIQUE INDEX (property_id, start_date, end_date)

Constraints: - `end_date` must be > `start_date`
- `value` must be > 0
- `client_id` must exist in clients table
- `property_id` must exist in properties table
- `status` ENUM ('active', 'expired', 'cancelled')

4. Use Cases – CRUD

4.1 Use Case: Create Client

Actor:
Authenticated User (Administrator or User with permissions)

Description:
The system allows authorized users to register new clients by providing their personal and contact information.

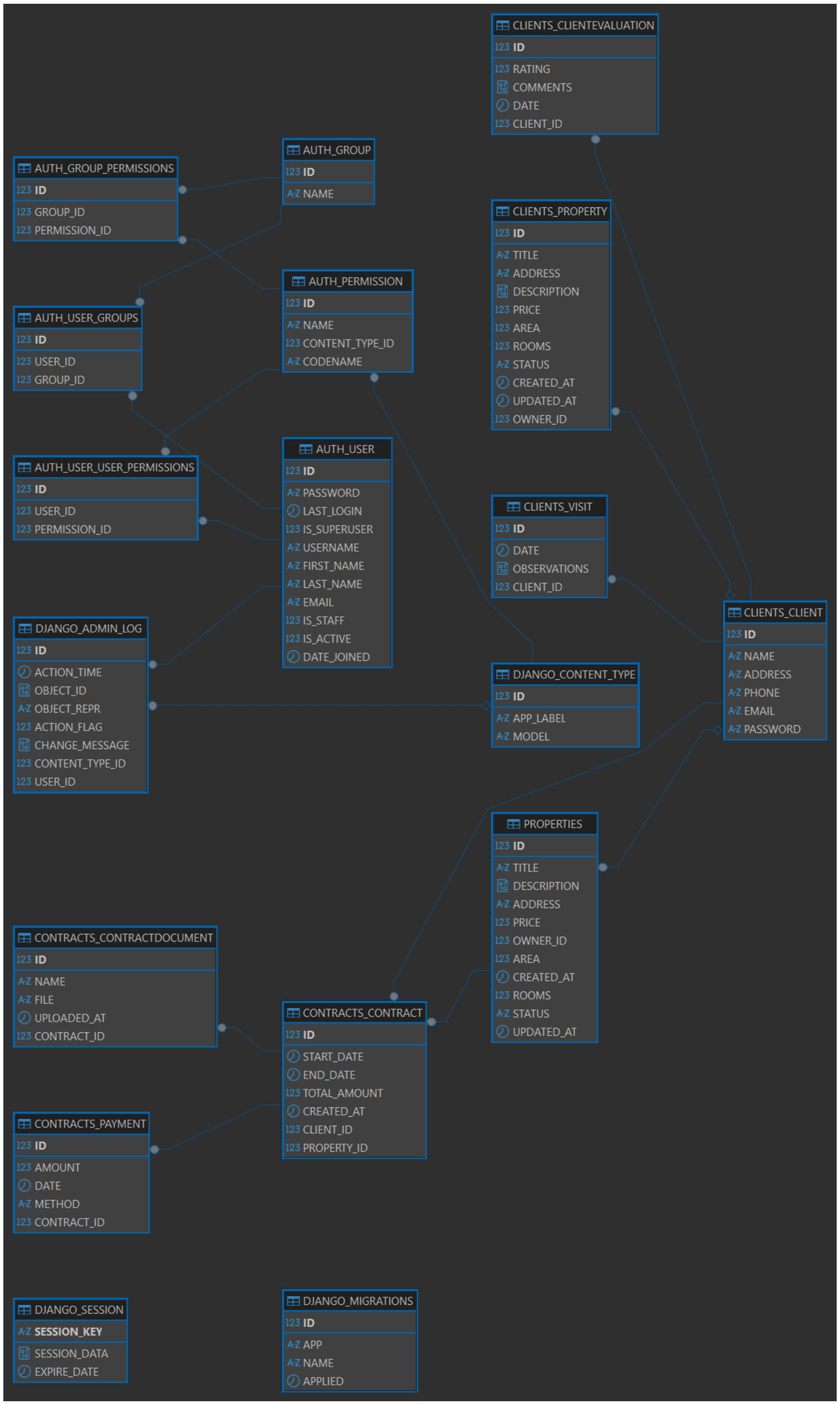

Figure 1: LUXUR

Preconditions:
- User must be authenticated
- User must have permission to create clients
- Email and document must be unique (not already registered)

Postconditions:
- New client record is created in the database
- Client appears in the clients list
- Success notification is displayed
- Audit log is updated with creation details

Main Flow:
1. User navigates to Clients module
2. User clicks "New Client" button
3. System displays client creation form
4. User fills in required fields:
- Full name
- Email address
- Phone number
- Physical address
- Document ID
5. System validates input in real-time
6. User clicks **Save**
7. System sends POST request to API: `/api/clients/`
8. Backend validates data and checks for duplicates
9. Backend creates new client record
10. Backend returns **201 Created**
11. Frontend updates clients list
12. System shows: **"Client created successfully"**

Alternative Flows:
A1: Validation Error
- Step 5: Invalid data detected
- Fields highlighted in red
- User corrects and continues

A2: Duplicate Email/Document
- Step 8: Backend detects duplicate
- Returns **400 Bad Request**
- Frontend shows: *"Email/Document already exists"*

A3: Network Error
- Step 7: Request fails
- System displays: *"Server error. Please try again"*

4.2 Use Case: Read/List Clients

Actor:
Authenticated User

Description:
User views the list of registered clients.

Preconditions:
- User authenticated
- User has permission to view clients

Postconditions:
- Client list displayed
- Actions available based on permissions

Main Flow:
1. User logs in
2. Navigates to **Clients**
3. System sends GET `/api/clients/`
4. Backend retrieves data
5. Returns JSON with client list
6. Frontend renders table
7. Pagination shown if needed
8. User may:
- View details
- Edit
- Delete
- Search/filter

Alternative Flows:
A1: No Clients Found
- Show message: *"No clients registered yet"*

A2: Search/Filter Applied
- GET `/api/clients/?search=term`
- System displays filtered results

4.3 Use Case: Update Client

Actor:
Authenticated User (Administrator or User with permissions)

Description:
Modify existing client information.

Preconditions:
- User authenticated
- User has update permissions
- Client exists
- New email/document must be unique

Postconditions:
- Client data updated
- Success notification displayed
- Audit log updated

Main Flow:
1. User opens Clients list
2. Clicks edit icon
3. GET `/api/clients/{id}/`
4. Backend loads client
5. Form displayed with current data
6. User edits fields
7. Real-time validation
8. User clicks **Update**
9. PUT `/api/clients/{id}/`
10. Backend validates and updates

11. Response **200 OK**
12. Frontend updates table
13. System shows: **“Client updated successfully”**

Alternative Flows:
A1: No Changes Made
- User clicks update/cancel → No API call

A2: Validation Error
- Same as Create flow

A3: Concurrent Modification
- Backend detects conflict
- System shows: *“Record was modified by another user. Please refresh”*

4.4 Use Case: Delete Client

Actor:
Authenticated Administrator

Description:
Delete a client unless active contracts exist.

Preconditions:
- User authenticated
- Admin privileges
- Client exists
- No active contracts

Postconditions:
- Client deleted permanently
- Removed from list
- Deletion logged

Main Flow:
1. User opens Clients list
2. Clicks delete icon
3. System shows confirmation dialog
4. User clicks **Confirm**
5. DELETE `/api/clients/{id}/`
6. Backend checks contracts
7. Backend deletes record
8. Returns **204 No Content**
9. Frontend removes entry
10. System shows: **“Client deleted successfully”**

Alternative Flows:
A1: User Cancels
- No changes

A2: Client Has Active Contracts
- Backend returns **400 Bad Request**
- Show: *“Cannot delete client with active contracts”*

A3: Client Not Found
- Backend returns 404
- Show: *“Client no longer exists”*

4.5 Use Case: Create Property

Actor:
Authenticated User

Description:
Register new property with full details.

Preconditions:
- User authenticated
- User has permission to create properties

Postconditions:
- Property created
- Available for contract assignment

Main Flow:
1. User clicks **New Property**
2. Form displayed
3. User fills in:
- Address
- Type
- Value
- Area
- Rooms
- Status
- Description
4. Validation
5. POST `/api/properties/`
6. Property created
7. System shows success message

4.6 Use Case: Create Contract

Actor:
Authenticated User

Description:
Create a new contract linking a client and property.

Preconditions:
- User authenticated
- At least one client exists
- At least one available property
- Dates must not conflict

Postconditions:
- Contract created
- Property status may change to *occupied*

Main Flow:
1. User clicks **New Contract**
2. Form displayed
3. User selects:
- Client

- Property
- Start date
- End date
- Value
- Terms
- Status
4. System validates:
- End > Start
- Property availability
5. POST `/api/contracts/`
6. Contract created
7. System displays success message

Alternative Flow:
A1: Property Already Contracted
- Date conflict detected
- Error: *"Property already contracted for selected dates"*

5. Backend Documentation

5.1 Authentication API

POST /api/token/

Generate access and refresh tokens.

Request Body:

```
{ "username": "admin", "password": "1234" }
```

Responses: - 200 OK – Tokens returned - 401 Unauthorized – Invalid credentials

---

5.2 Refresh Token

POST /api/token/refresh/

Body:

```
{ "refresh": "your_refresh_token_here" }
```

5.3 API Documentation (Example: Clients Module)

GET /api/clients/

Retrieve list of all clients.

Response:

```
[
  {
    "id": 1,
    "name": "John Doe",
    "email": "john@example.com",
    "phone": "123456789"
  }
]
```

POST /api/clients/

Create a new client.

Request Body:

```
{
  "name": "New Client",
  "email": "new@example.com",
  "phone": "987654321",
  "status": "ACTIVE"
}
```

Responses: - 201 Created
- 400 Bad Request

GET /api/clients/:id

Fetch specific client.

PUT /api/clients/:id

Update client.

DELETE /api/clients/:id

Delete client.

5.4 REST Client Tools

Recommended tools: - cURL
- Postman
- ThunderClient (VSCode)
- Insomnia

Example cURL call:

```
curl -H "Authorization: Bearer <ACCESS_TOKEN>" http://127.0.0.1:8000/api/clients/
```

---

6. Frontend Documentation

6.1 Technical Frontend Documentation

Framework Used: **Angular 17+**

Folder Structure:

```
frontend/
    src/
        app/
            models/
                client.ts
            services/
                client.service.ts
            components/
                client/
                    getall/
                    create/
                    update/
                    delete/
            assets/
            environments/
```

6.2 Models, Services, and Components

Model Example (client.ts)

```
export interface ClientI {
  id?: number;
  name: string;
  address: string;
  phone: string;
  email: string;
  password: string;
  status: "ACTIVE" | "INACTIVE";
}
```

Service Example (client.service.ts)

Handles all HTTP calls to the backend.

Components:

- GetAllComponent
- CreateComponent
- UpdateComponent
- DeleteComponent

6.3 Visual Explanation

Add screenshots of:

- CRUD Clients
- CRUD Properties

- Menu navigation
- Login page
- Table views
- Form views

---

7. Frontend–Backend Integration

Integration uses:

- Angular HttpClientModule
- JWT Interceptor
- Django REST API Endpoints

Flow:

```
Angular → Sends Bearer Token → Django DRF validates → Returns JSON
```

Error handling: - 401 Unauthorized
- 400 Bad Request
- Validation errors

---

8. Conclusions & Recommendations

Conclusions

- Modular structure ensures scalability
- JWT ensures secure authentication
- Angular components improve UI modularity
- DRF standardizes API communication

Recommendations

- Implement RBAC (Role-Based Access Control)
- Add Docker
- Improve logs & monitoring
- Add testing (pytest + Jasmine/Karma)