Digital Twin

Software Design Specification
presented to the academic faculty
by

Nehal Naeem Haji	nh07884
Manal Hasan	mh08438
Eeshal Khalidnadeem Qureshi	eq08433
Muhammad Shawaiz Khan	mk07899

In partial fulfillment of the requirements for
Bachelor of Science
Computer Science

Dhanani School of Science and Engineering

Habib University
Fall 2025

1 Introduction

The purpose of this Software Design Specification (SDS) is to document the architecture, data flow, database design, and algorithms used in the Digital Twin project. This document will serve as a roadmap for the technical implementation, ensuring all project components align to meet the user requirements.

1.1 Scope

The scope of this project is to develop a proof-of-concept digital twin system for two adjacent university laboratory spaces: the Projects Lab and the Digital Instrumentation Lab on the lower-ground floor. The system addresses three critical challenges in campus management that are traditionally handled separately: occupancy monitoring, emergency preparedness, and energy efficiency.

The digital twin will perform real-time, anonymous crowd counting using vision-based analysis of video segments. Occupancy data will be visualized in a 3D virtual environment using avatars and crowd-density heatmaps, providing an intuitive interface for monitoring space usage. The system will automatically generate alert notifications for security personnel during emergency or evacuation scenarios, identifying spaces that remain occupied to support faster, coordinated responses. Additionally, the system will integrate simulated IoT data - such as temperature and lighting - with real-time occupancy levels to identify inefficiencies like lighting or cooling systems operating unnecessarily in empty rooms.

2 Project Architecture

2.1 UML Class Diagram

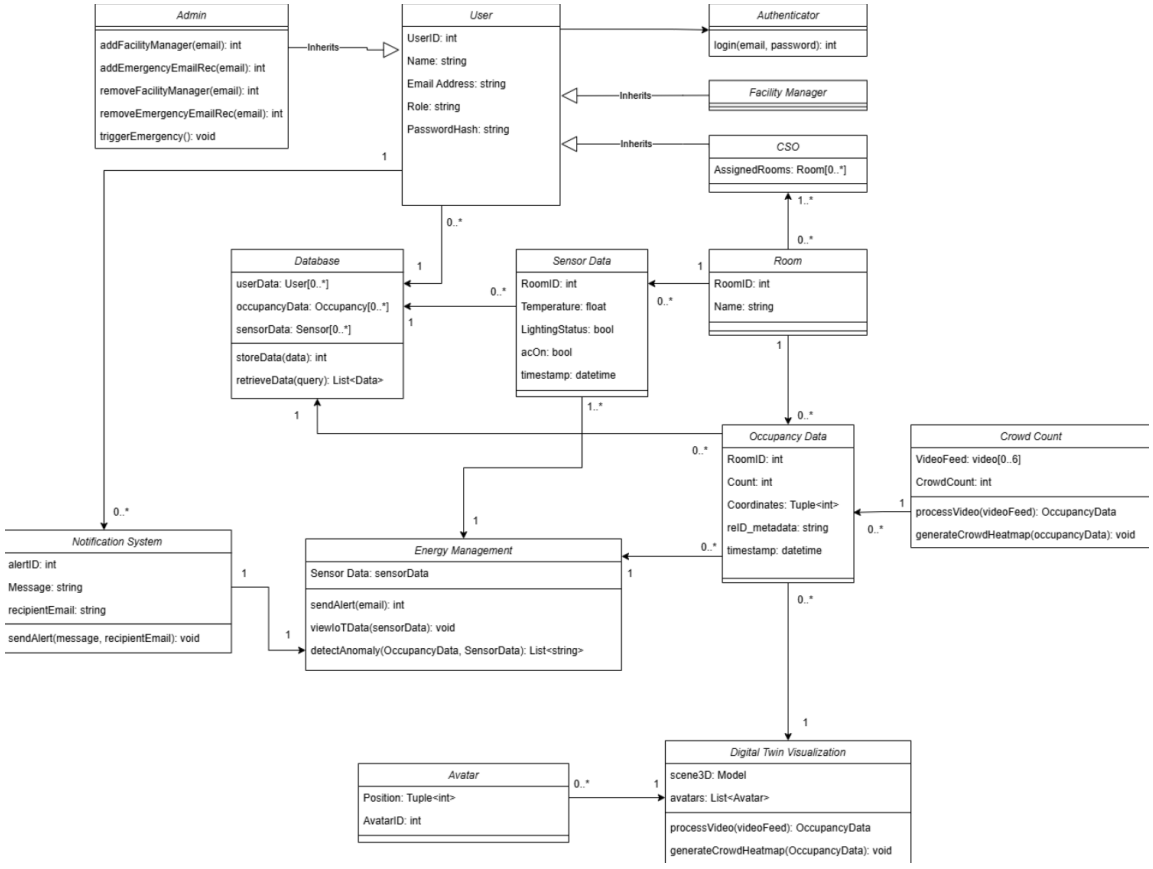

Figure 1: UML Class Diagram

2.2 Description

2.2.1 User

The **User** class represents an individual within the system and contains several key attributes: a unique identifier (`UserID`, stored as an integer), the user's full name (`Name`), their email address (`EmailAddress`), the role assigned to them (such as Facility Manager or CSO), and a secure hash of their password (`PasswordHash`).

While the class itself does not directly manage role assignment, the user's role is administered by the **Admin** class, which establishes an association between Admin and User. Authentication of the User is handled through the **Authenticator** class, for secure login and verification.

2.2.2 Authenticator

The **Authenticator** class is responsible for managing user access within the system. It provides one primary method: `login(email, password)`, which authenticates a user based on their email and password.

2.2.3 Admin

The **Admin** class represents the system administrator and has two attributes: the administrator's name (`Name`) and their email address (`EmailAddress`) inherited from the **User** class. It provides several methods to manage the system and for emergency control. Through `addFacilityManager(email)` and `removeFacilityManager(email)`, the Admin can add or remove Facility Managers from the system. The method `triggerEmergency()` allows the Admin to activate the emergency alert system when needed. Additionally, the Admin can manage emergency communication by using `addEmergencyEmailRec(email)` to add recipients for alerts and `removeEmergencyEmailRec(email)` to remove them. The Admin class assigns roles and manages users through its connection to the **User** class, while also linking to the **EmergencyAlert** class to initiate and control emergency notifications.

2.2.4 Notification System

The **Notification System** class is designed to deliver real-time alerts to users, particularly Facility Managers and designated CSOs. Each alert is defined by an `alertID` (a unique integer identifier), a `Message` containing the content of the alert, and a `recipientEmail` specifying the recipient's address. The class provides the method `sendAlert(message, recipientEmail)`, which generates and transmits the alert to the intended recipient.

2.2.5 Energy Management

The **Energy Management** class is responsible for monitoring and optimizing energy usage within facilities. Each room or facility is identified by a unique `RoomID` (Integer), and the class tracks key environmental parameters such as the current

`Temperature` (Float), the `LightingStatus` (Boolean), and whether the air conditioning system is active (`acOn`, Boolean). To support real-time analysis, the method `viewIoTData()` enables the system to retrieve simulated IoT sensor data, providing continuous visibility into environmental conditions. When lights or the AC is found to be on when the room occupancy is 0, the `sendAlert()` method is used to notify facility managers. In terms of relationships, the Energy Management system integrates with both **CrowdCount** and **Sensor Data** to gain insights into occupancy and environmental patterns, which inform optimization strategies. It also interacts with the **Notification System** to trigger alerts whenever anomalies are detected.

2.2.6 Crowd Count

The **Crowd Count** class is responsible for monitoring and analyzing crowd occupancy within a given area. It processes a `VideoFeed`, represented as an array of video streams, to detect and track people in real time. From this data, it calculates the `CrowdCount`, an integer representing the number of individuals present, and records `Coordinates`, a tuple of integers that captures the geographical location of each individual. The class provides two key methods: `getPeopleCount()`, which returns the total number of people detected, and `generateCrowdHeatmap()`, which produces a visual heatmap based on crowd distribution patterns. In terms of relationships, the Crowd Count class integrates with both the **Energy Management** system and the **Notification System**, supplying occupancy data that informs energy optimization decisions and triggers emergency alerts when people are detected during evacuation.

2.2.7 Database

The **Database** class functions as the central repository for all system data, ensuring that information is securely stored and readily accessible to other components. It maintains three primary categories of data: `userData`, which contains authentication details and role management information; `occupancyData`, which records real-time occupancy metrics; and `sensorData`, which stores simulated IoT data. To manage this information, the class provides two key methods: `storeData(data)`, which saves new entries into the database, and `retrieveData(query)`, which allows data to be fetched based on specific queries. In terms of relationships, the Database class serves as the backbone for data exchange, providing access to User, Occupancy, and Sensor information that supports authentication, energy management, crowd monitoring, and alert generation.

2.2.8 Avatar

The **Avatar** class represents individual agents within the virtual 3D environment. Each avatar is uniquely identified by an `AvatarID` (Integer) and maintains a `Position` (Tuple of integers) that specifies its current location in the simulated space. To support analysis and visualization, the class provides the method `generateCrowdHeatmap()`, which captures avatar movement patterns and produces a heatmap to track crowd density across the environment. In terms of relationships, the Avatar class interacts closely with the **Crowd Count** system and the **Digital Twin Visualization** system, serving as the digital representation of people or agents whose positioning inform occupancy analysis and energy management.

2.2.9 Digital Twin Visualization

The **Digital Twin Visualization** class serves as the graphical interface for the system, which will render a real-time, 3D replica of the labs. It maintains a reference to the static building structure (`scene3D`) and a dynamic list of `avatars` that represent people currently in the building. The class relies on the `processVideo()` method to ingest real-time **OccupancyData**, translating raw coordinates into visual avatar movements. Additionally, it uses `generateCrowdHeatmap()` to generate a heatmap of crowd density.

2.2.10 Facility Manager

The **Facility Manager** class represents a specialized user role responsible for the operational maintenance and energy efficiency of the building. Inheriting from the **User** class, it gains specific authority over building zones. Relationships in the system indicate that a Facility Manager is assigned to monitor one or more **Rooms**. This role is the primary recipient of energy efficiency alerts (e.g., lights left on in empty rooms) generated by the **Energy Management** system.

2.2.11 CSO (Chief Security Officer)

The **CSO** class represents security personnel tasked with monitoring safety and responding to emergencies. Like the Facility Manager, the CSO inherits from the **User** class but is distinguished by the `AssignedRooms` attribute, which links them to specific security zones or sensitive areas within the facility. The CSO is the designated recipient of high-priority alerts triggered by the **Notification System**, particularly when the **Admin** triggers an emergency mode.

2.2.12 Room

The **Room** class acts as the logical representation of a physical space within the facility, identified by a unique `RoomID` and a descriptive `Name`. The diagram illustrates that a Room is the parent entity for `0..*` instances of **Sensor Data** (environmental logs) and **Occupancy Data** (crowd logs), creating a history of what occurs within that specific space. It also maintains direct associations with the **CSO** and **Facility Manager** classes, defining who is responsible for the security and maintenance of that specific area.

2.2.13 Occupancy Data

The **Occupancy Data** class serves as a structured record of crowd analytics obtained from the computer vision processing pipeline. Each instance captures a snapshot of a specific moment, recording the `RoomID`, the total person `Count`, and precise `Coordinates` (tuples) of individuals. It also includes `reID_metadata`, which allows the system to maintain the identity of individuals across video frames for consistent tracking. This data is generated by the **Crowd Count** system and is consumed by the **Digital Twin Visualization** to render avatars and by the **Energy Management** system to correlate occupancy with energy usage.

2.2.14 Sensor Data

Each record is timestamped and linked to a specific `RoomID`. The class tracks critical variables including`Temperature` (Float), `LightingStatus` (Boolean), and the state of the AC system (`acOn`). By providing a historical log of environmental conditions, this class allows the **Energy Management** system to detect inefficiencies - such as cooling systems running in unoccupied rooms - and serves as the data source for the `viewIoTData()` method used for real-time monitoring.

3 Data Model

We will now cover the database design of our project, entailing the Entity-Relationship Diagram (ERD) and its descriptions. This section will allow for efficient storage and allowing the usage of room sensor and occupancy data, enabling model generation, and defining user access for security measures.

3.1 Entity-Relationship Diagram (ERD)

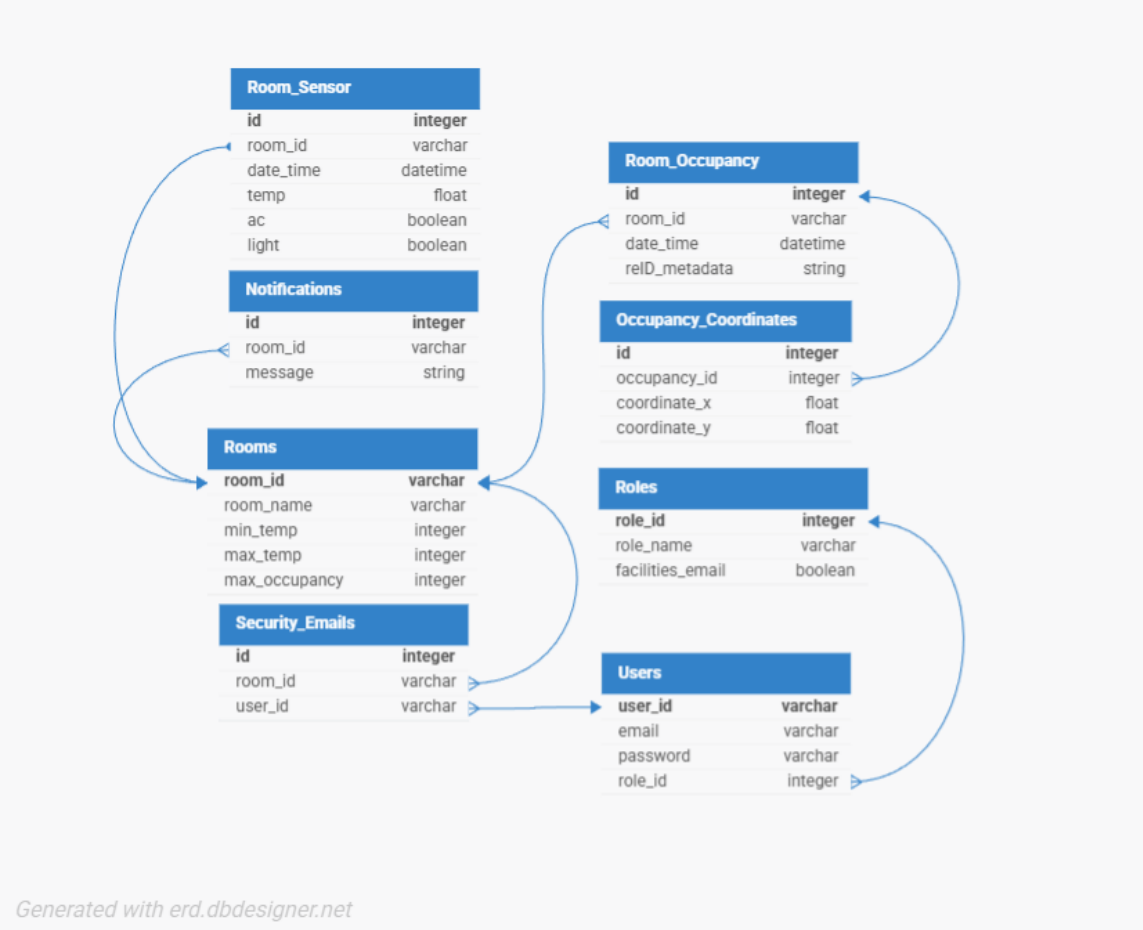

Figure 2: Entity-Relationship Diagram

3.2 Entities

3.2.1 Users

The **Users** entity includes attributes such as `user_id` (Primary Key), `email`, `password`, and `role_id` (foreign key from Roles). It stores information about the users, including their authentication credentials and role IDs, which is useful for identifying users and granting appropriate access to the system's functionalities.

3.2.2 Roles

The **Roles** entity defines the access privileges of the users according to their roles. Attributes include `role_id` (Primary Key), `role_name`, and `facilities_email`. Roles define the department and position of the user, alongside whether they are eligible to receives email regarding facilities alerts.

3.2.3 Rooms

The **Rooms** entity stores data regarding the rooms in the university. Attributes include `room_id` (Primary Key), `room_name`, `min_temp`, `max_temp`, and `max_occupancy`. It acts as a key entity which stores the ideal temperature ranges and maximum occupancy the room can hold before over-crowding, as well as connecting room sensor and occupancy data.

3.2.4 Room Sensor

The **Room Sensor** entity keeps data regarding IoT data of the room at a specific time. Attributes include `id` (Primary Key), `room_id` (foreign key from Rooms), `date_time`, `temp`, `ac` and `light`. It helps in storing data for each room at specific timeframes regarding the temperature, as well as whether the AC and lights are on at a specific time.

3.2.5 Room Occupancy & Occupancy Coordinates

The two entities coupled together store data regarding the occupancy count and coordinates of individuals at a specific time. The **Room Occupancy** attributes include `id` (Primary Key), `room_id` (foreign key from Rooms), `date_time`, and `reID_metadata`. **Occupancy Coordinates'** attributes include `id` (Primary Key), `occupancy_id` (foreign key from Room Occupancy), `coordinate_x` and `coordinate_y`.

This duet helps in storing key occupancy data for each room at specific timeframes, which is reflected onto the 3D model, and giving security alerts regarding overpopulation at a certain area.

3.2.6 Security Emails & Notifications

The two entities help in sending alerts regarding a specific room whenever there is an emergency. The **Security Emails** entity entails of the attributes `id` (Primary Key),

`room_id` (foreign key from Rooms) and `user_id` (foreign key from Users). **Notifications'** attributes include `id` (Primary Key), `room_id` (foreign key from Rooms) and `message`.

Both entities help in controlling the distribution and timing of notifications.

4 Data Pipeline

This section describes the data pipeline, detailing the stages from data collection to final user output. Each step ensures data is processed to meet the requirements of the prediction models and user interface.

4.1 Data Collection

Visual data is gathered through synchronized video footage captured by 5 CCTV cameras installed in specific locations within the Projects Lab and the Digital Instrumentation Lab. The system also ingests simulated IoT data - including temperature readings and lighting status - which supplies data for real-time environmental monitoring.

4.2 Data Preprocessing

Before analysis can occur, raw video inputs undergo preprocessing to ensure the object detection and tracking models perform optimally. Specifically, the lighting and contrast levels of the video streams are adjusted to standardize visual input quality.

4.3 Data Storage

MongoDB will be used to store both structured data, such as occupancy counts and timestamps, and unstructured data, including tracking metadata to provide immediate access to necessary historical and current data.

4.4 Data Processing

For crowd analytics, video data is continuously analyzed using YOLO 12 to detect individuals, while ByteTrack/DeepSORT maintains unique identities across multiple frames to prevent double-counting. Simultaneously, the Energy Management module

correlates IoT sensor readings with occupancy records to identify inefficiencies, such as detecting specific instances where lights are left active in empty rooms.

4.5 Data Visualization

To translate complex data into interpretable insights for campus managers, the system employs a variety of visualization tools. **Three.js** is used to render a 3D digital twin of the facility, overlaying real-time occupancy and crowd-density heatmaps directly onto the virtual model. **Matplotlib** generates the crowd density heatmaps. **React Three Fiber** is used to create the User Interface, allowing users to view their respective dashboards.

4.6 Notifications

SMTP2GO, a SMTP service, triggers real-time email notifications to designated CSOs if individualas are detected during emergency evacuations. Similarly, Facilities Managers receive immediate energy alerts if rooms remain unoccupied for extended periods (e.g., exceeding 25 minutes) while systems are active.

5 Technical Details

5.1 Machine Learning Models

Machine learning models form the backbone of the project's crowd detection and forecasting capabilities, working in tandem to process raw video into actionable data.

YOLO 12 (You Only Look Once)

YOLO 12 serves as the project's primary deep learning engine for real-time object detection. By analyzing video frames directly from university lab CCTV cameras, the model generates bounding boxes and class labels to identify individuals with high precision.

ByteTrack (Multi-Object Tracking)

Alongside the detection layer, ByteTrack provides robust multi-object tracking capabilities. It takes in the detection outputs from YOLO 12 to assign and maintain persistent object IDs across consecutive video frames. This mechanism makes sure that specific individuals are accurately tracked over time, even as they move between

different camera angles or are temporarily obstructed, preserving the integrity and continuity of the occupancy data.

5.2 Notifications

For communication during critical events, the system relies on the **SMTP2GO API** to deliver real-time notifications. Acting on triggers received from the crowd detection or energy management subsystems, the API generates automated email alerts for CSOs and Facility Managers respectively.

5.3 Data Storage

Data persistence is managed through a hybrid approach to balance system performance with ethical compliance. **MongoDB** is used for the high-speed storage and retrieval of structured data, such as real-time occupancy counts and temperature readings. On the other hand, raw video feeds are stored physically on-premises using **Workstation Local Storage**.

5.4 Visualization Algorithms

To interpret and display complex data for campus managers, the system makes use of specific algorithms and visualization libraries. **Matplotlib** is used to generate crowd density heatmaps, offering a clear view of the labs' real-time status. The system also applies **Homography**, a computer vision technique that translates coordinates from 2D video streams into 3D world coordinates. This transformation is essential for accurately mapping physical locations onto the digital twin.