# A black and white logo Description automatically generated

# Design Document

**Roomify** is an automated meeting room booking system designed to streamline the process of reserving meeting spaces within a corporate environment. The system provides a user-friendly interface, developed using HTML, CSS, and JavaScript, to allow employees to search and book meeting rooms according to their specific requirements. The backend of Roomify, built in Java using JDBC and MySQL, manages all the core functionalities, including user management, room scheduling, meeting organization, and amenity management. Although the frontend and backend are currently not linked, this document outlines the design principles and architecture of the backend system, ensuring that it adheres to industrial coding standards and robust error handling practices. The project also includes specialized functionality to ensure that certain types of meetings automatically include mandatory amenities, enhancing the overall efficiency of the booking process. Certain user access also allows for viewing the usage statistics of rooms in order to see which room and amenity is used maximally.

## Technology Stack

We used the following technologies:-

**Frontend**

1. HTML
2. CSS
3. JS

**Backend**

1. JAVA
2. JDBC
3. MYSQL
4. Junit
5. Nio

## Scope of the Project

The scope of the Roomify project encompasses the development and implementation of a robust and scalable backend system for an automated meeting room booking application. The primary focus is on designing and building the core functionalities that manage user roles, room scheduling, meeting organization, viewing usage statistics, and the allocation of amenities based on specific meeting requirements.

**Key Functionalities Covered:**

* **User Management:** The system supports three types of users—Admin, Manager, and Member. Each user role has distinct permissions, such as creating rooms, booking meetings, and viewing schedules.
* **Room Management:** Admins can create and configure meeting rooms, specifying attributes such as seating capacity and available amenities (e.g., projectors, conference call facilities, whiteboards).
* **Booking System:** Managers can book meeting rooms based on availability and requirements, with the system automatically handling credits and ensuring mandatory amenities are selected for specific meeting types.
* **Credits System:** The system implements a credits-based booking mechanism, where managers are assigned credits that are deducted when a room is booked. Credits are reset to a default value on a weekly basis.
* **Meeting Management:** The system ensures that each meeting has the necessary amenities based on its type, such as conference calls or training sessions. The schedule for booked meetings is also maintained and displayed.
* **Usage Statistics:** The system implements a graphical usage statistic showing the hours and number of bookings done on each room and which room is the most used one of all.
* **Error Handling and Validation:** The project includes comprehensive error handling mechanisms and input validation to ensure data integrity and system reliability.

**Exclusions:**

* **Frontend-Backend Integration:** While the frontend is developed using HTML, CSS, and JavaScript, the current project scope does not cover the integration of the frontend with the backend. Future development phases will address this linkage. However, the stage that the project is at the frontend is designed to implement API calls and use the same data format as the backend. Backend on the other hand is designed to be ready to incorporate spring boot with all layers in place and annotations and api scripting needed.
* **External API Integration:** Any integration with third-party services or APIs is beyond the scope of this project.

## Data Flow Diagram

A Data Flow Diagram (DFD) is a graphical representation that depicts the flow of data within a system. It illustrates how data moves between various processes, data stores, and external entities, providing a clear and concise overview of the system's structure and operations.

**Entities, Processes, and Data Stores**

**Entities:**

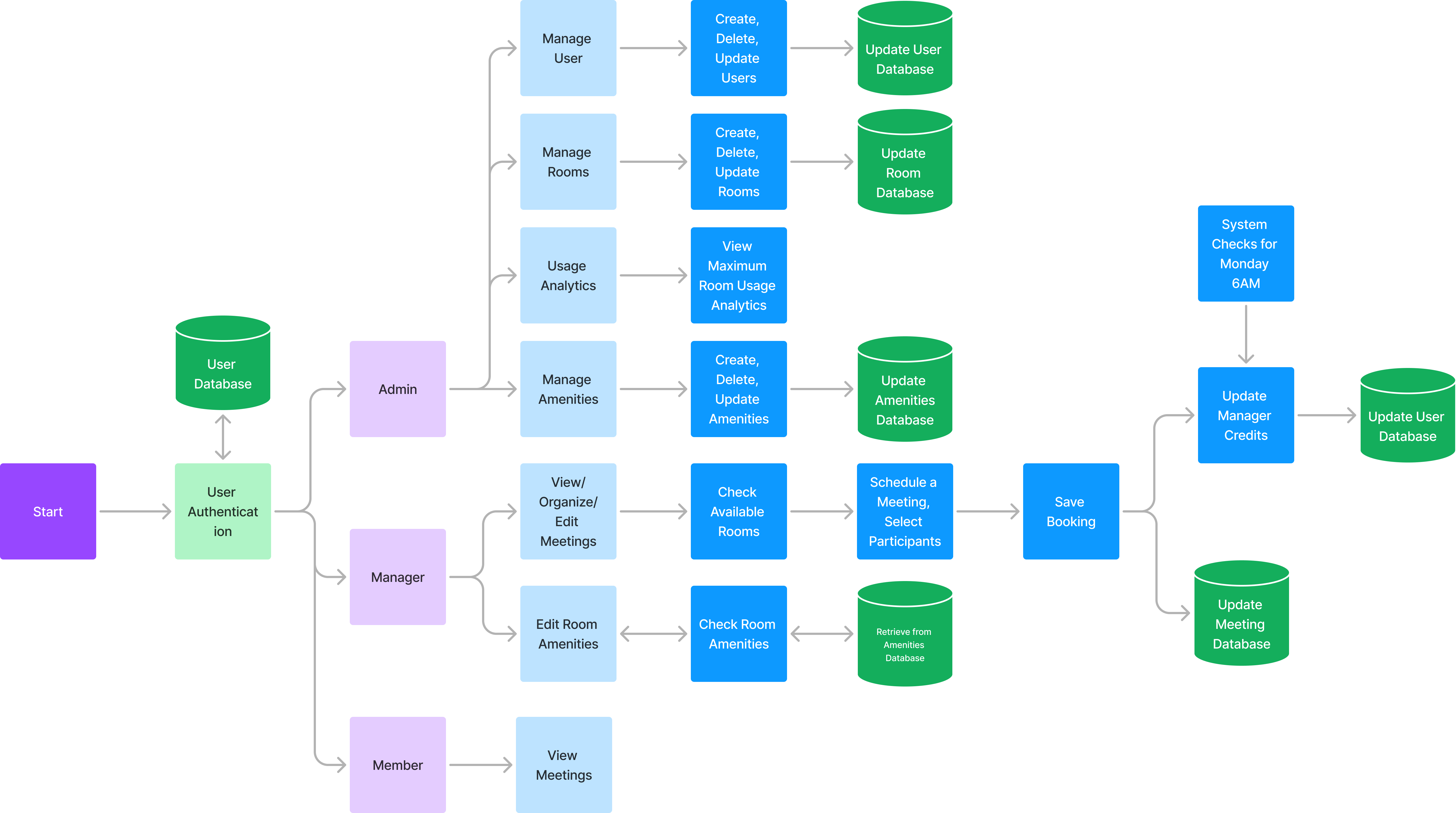
* Admin
* Manager
* Member
* Meetings
* Amenities

**Processes:**

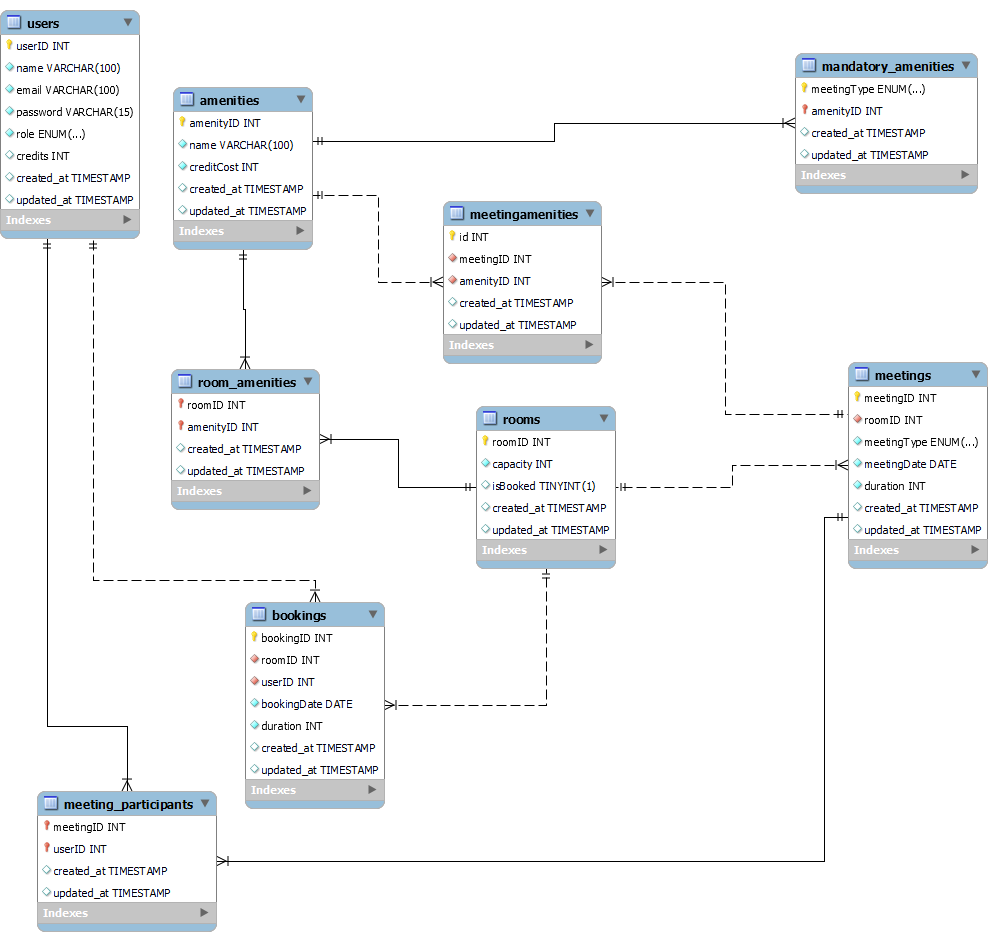
1. Login
2. Manage Users (Admin)
3. Create Room (Admin)
4. Edit Room (Admin)
5. View Schedule (Manager/Member)
6. Book Room (Manager)
7. Organize Meeting (Manager)
8. Update Credits (System)
9. View Meetings (Member)
10. View Usage Statistics for Rooms (Admin)
11. Edit Meeting (Manager)

**Data Stores:**

1. User Database
2. Room Database
3. Booking Database
4. Amenity Database
5. Meeting Database



## Entity-Relationship (ER) Diagram



## Class UML Diagram

A diagram of a meeting

Description automatically generated

## Error Handling and Validation

**1. User Authentication:**

* **Error Handling:**
  + **UserNotFoundException:** If the provided username does not exist in the system, throw this exception and display an error message indicating that the user was not found.
  + **Invalid Credentials:** If a user enters an incorrect password, return an appropriate error message and log the attempt for security monitoring.
  + **Account Lockout:** After a certain number of failed login attempts (e.g., 5), lock the account temporarily and notify the user via email.
* **Validation:**
  + **Username Format:** Ensure the username meets the required format (e.g., email format or alphanumeric, depending on the design).
  + **Password Strength:** Enforce password complexity requirements (e.g., minimum length, inclusion of uppercase, lowercase, numbers, and special characters).
  + **Input Sanitization:** Prevent SQL injection and XSS by sanitizing input fields during login attempts.

**2. Room Management:**

* **Error Handling:**
  + **RoomNotAvailableException:** If a room is unavailable for booking due to maintenance or an existing booking, throw this exception and alert the user.
  + **Room Not Found:** If a room ID referenced does not exist, show an error message and ensure no operations are carried out on a non-existent room.
  + **Duplicate Room Entry:** If an attempt is made to add a room that already exists, prevent the action and alert the user.
* **Validation:**
  + **Room ID:** Validate that the room ID is unique when adding a new room.
  + **Room Capacity:** Ensure that the room capacity is a positive integer and within the allowed limits.
  + **Input Format:** Validate all inputs for room details, such as room name, location, and amenities, ensuring they are in the correct format.

**3. Booking Process:**

* **Error Handling:**
  + **InsufficientCreditException:** If the user does not have enough credits to book a room, throw this exception and notify them to add more credits or adjust the booking.
  + **RoomNotAvailableException:** If the room is already booked or unavailable, throw this exception to prevent double booking and inform the user.
  + **Overlapping Bookings:** Prevent overlapping bookings by checking for conflicts in the Booking Database and alert the manager if a conflict is found.
* **Validation:**
  + **Date and Time Validation:** Ensure the booking date and time are valid and within the operational hours.
  + **Room Availability Check:** Validate that the room is available before confirming the booking.
  + **User Authorization:** Ensure that the user (manager) has the appropriate permissions to make bookings.

**4. Meeting Organization:**

* **Error Handling:**
  + **AmenityNotFoundException:** If a required amenity for a meeting is not available, throw this exception and prevent the meeting from being scheduled.
  + **RoomNotAvailableException:** If the room is unavailable for the desired meeting time, throw this exception and notify the user.
  + **Exceeding Capacity:** If the number of participants exceeds the room capacity, notify the user and prevent the booking.
* **Validation:**
  + **Meeting ID:** Ensure that the meeting ID is unique and properly formatted.
  + **Participant List:** Validate that the list of participants is complete and formatted correctly.
  + **Time Validation:** Check that the meeting start time and duration are valid and do not overlap with other bookings.

**5. Viewing Meetings:**

* **Error Handling:**
  + **UserNotFoundException:** If a user tries to view meetings without proper credentials, throw this exception and log the unauthorized attempt.
  + **Meeting Not Found:** If a meeting ID is referenced that does not exist, show an error message and prevent further actions.
  + **Database Retrieval Failure:** If there’s an error retrieving meeting data from the database, log the error and notify the admin.
* **Validation:**
  + **Meeting ID Input:** Validate that the meeting ID provided by the user exists in the system.
  + **User Permissions:** Ensure the user has the appropriate permissions to view the specific meeting details.

**6. General System-Wide Error Handling:**

* **Database Connection Errors:** Implement a global handler for database connection failures to ensure that users are informed appropriately and that logs are maintained for analysis.
* **Unhandled Exceptions:** Implement a global exception handler to catch and log unhandled exceptions, providing user-friendly error messages.
* **Data Integrity:** Ensure that all database transactions are atomic, meaning that any error during a transaction should result in a rollback to maintain data integrity.

## Future Integration Plans

As **Roomify** continues to evolve, several future integration plans will enhance the system's functionality and user experience:

1. **Front-End and Back-End Integration:**
   * The immediate next step is to integrate the front-end (developed using HTML, CSS, and JavaScript) with the back-end (built using Java, JDBC, and MySQL). This integration will allow users to interact with the system through a web-based interface, making the process of booking rooms and organizing meetings more user-friendly.
2. **Mobile Application Development:**
   * Develop a mobile application that mirrors the functionalities of the web-based platform. A mobile app will provide users with the convenience of booking and managing rooms and meetings on the go, enhancing the overall accessibility of the system.
3. **Role-Based Access Control Enhancement:**
   * Further enhancement of role-based access control (RBAC) to accommodate more complex user roles and permissions. This could involve adding more granular access levels or introducing new roles within the organization as the system scales.
4. **Cloud Integration:**
   * Consider moving the application to a cloud platform to ensure scalability, better performance, and easier maintenance. Cloud integration will also allow for the seamless deployment of new features and updates.

## Conclusion

The **Roomify** project is a robust and scalable meeting room booking system designed to streamline the process of managing meeting spaces within an organization. The system's core functionalities, such as user management, room booking, and meeting organization, are well-structured and cater to the needs of different user roles (Admin, Manager, Member). The project adheres to industrial coding standards, ensuring a reliable and maintainable codebase.

With the proposed future integration plans, **Roomify** is poised to become a comprehensive solution for meeting room management, offering enhanced user experiences, real-time notifications, mobile accessibility, and advanced analytics. The project's solid foundation in Java, JDBC, and MySQL provides a stable backend, while the planned front-end integration will bring the system to life for end users. As the system grows, continuous improvements and integrations will ensure that **Roomify** remains an asset for organizations, optimizing their meeting management processes and improving overall efficiency.