School Inventory Management Backend System

1. Introduction

This document outlines the design and architecture of a robust backend system for managing school inventory. The primary goal is to enhance inventory control and accountability by providing functionalities for item tracking, collector assignment, and automated return reminders. This system is designed for use by school administrators, inventory managers, and other authorized personnel to streamline inventory operations.

2. Project Goal

To develop a robust backend system for managing school inventory, including item tracking, collector assignment, and automated return reminders, thereby enhancing inventory control and accountability.

3. Target Audience

School administrators, inventory managers, and authorized personnel who require efficient tools for managing school assets.

4. Project Scope Breakdown

4.1. Data Modeling and Database Design (PostgreSQL)

The system utilizes a PostgreSQL database to store all inventory-related data. The schema is designed to be normalized, ensuring data integrity and efficient querying. All primary and relevant foreign keys are implemented using `UUID` (Universally Unique Identifiers) for better scalability and distributed system compatibility. Timestamps (`created_at`, `updated_at`) are included in all main tables for auditing purposes, automatically managed by database triggers.

4.1.1. Tables Overview:

- **`categories`**: Stores different types of inventory items (e.g., Electronics, Books, Equipment).
    - `id` (UUID, Primary Key, `DEFAULT gen_random_uuid()`): Unique identifier for the category.
    - `name` (VARCHAR(255), UNIQUE, NOT NULL): The name of the category (e.g., 'ELECTRONICS'). Stored as a string from the `CategoryName` enum.
    - `description` (TEXT): A detailed description of the category.
    - `created_at` (TIMESTAMP WITH TIME ZONE, DEFAULT CURRENT_TIMESTAMP): Timestamp of record creation.
    - `updated_at` (TIMESTAMP WITH TIME ZONE, DEFAULT CURRENT_TIMESTAMP): Timestamp of last record update.

- **`collectors`**: Stores information about individuals to whom items can be assigned (e.g., teachers, staff, students).
    - `id` (UUID, Primary Key, `DEFAULT gen_random_uuid()`): Unique identifier for the collector.
    - `name` (VARCHAR(255), NOT NULL): Full name of the collector.
    - `contact_information` (VARCHAR(255)): General contact details (e.g., phone number).
    - `email` (VARCHAR(255), UNIQUE, NOT NULL): Email address of the collector, used for reminders.
    - `created_at` (TIMESTAMP WITH TIME ZONE, DEFAULT CURRENT_TIMESTAMP): Timestamp of record creation.
    - `updated_at` (TIMESTAMP WITH TIME ZONE, DEFAULT CURRENT_TIMESTAMP): Timestamp of last record update.
- **`assignments`**: Records the assignment of an item to a collector.
    - `id` (UUID, Primary Key, `DEFAULT gen_random_uuid()`): Unique identifier for the assignment.
    - `assignment_date` (DATE, NOT NULL, DEFAULT CURRENT_DATE): The date the item was assigned.
    - `return_due_date` (DATE): Optional date by which the item is expected to be returned.
    - `actual_return_date` (DATE): The actual date the item was returned. `NULL` if not yet returned.
    - `collector_id` (UUID, NOT NULL, Foreign Key to `collectors.id`): The collector to whom the item is assigned.
    - `created_at` (TIMESTAMP WITH TIME ZONE, DEFAULT CURRENT_TIMESTAMP): Timestamp of record creation.
    - `updated_at` (TIMESTAMP WITH TIME ZONE, DEFAULT CURRENT_TIMESTAMP): Timestamp of last record update.
- **`items`**: Stores details about each individual inventory item.
    - `id` (UUID, Primary Key, `DEFAULT gen_random_uuid()`): Unique identifier for the item.
    - `name` (VARCHAR(255), UNIQUE, NOT NULL): The name of the item.
    - `description` (TEXT): A detailed description of the item.
    - `category_id` (UUID, NOT NULL, Foreign Key to `categories.id`): The category the item belongs to.
    - `serial_number` (VARCHAR(255), UNIQUE, NOT NULL): A unique serial number for the item.
    - `status` (VARCHAR(50), NOT NULL, DEFAULT 'AVAILABLE'): Current status of the item (e.g., 'AVAILABLE', 'ASSIGNED', 'RETURNED'). Stored as a string from the `Status` enum.
    - `assignment_id` (UUID, UNIQUE, Foreign Key to `assignments.id`): References the currently active assignment for this item. `NULL` if the item is not currently assigned. This forms a one-to-one relationship with `assignments`.
    - `created_at` (TIMESTAMP WITH TIME ZONE, DEFAULT CURRENT_TIMESTAMP): Timestamp of record creation.
    - `updated_at` (TIMESTAMP WITH TIME ZONE, DEFAULT CURRENT_TIMESTAMP): Timestamp of last record update.
- **`reminders`**: Stores records of reminders sent for overdue assignments.

    - `id` (UUID, Primary Key, `DEFAULT gen_random_uuid()`): Unique identifier for the reminder.
    - `assignment_id` (UUID, NOT NULL, Foreign Key to `assignments.id`): The assignment this reminder is for.
    - `reminder_date` (DATE, NOT NULL, DEFAULT CURRENT_DATE): The date the reminder was generated/sent.
    - `status` (VARCHAR(50), NOT NULL, DEFAULT 'PENDING'): Status of the reminder (e.g., 'PENDING', 'SENT', 'FAILED', 'DISMISSED').
    - `message` (TEXT): The content of the reminder message.
    - `sent_at` (TIMESTAMP WITH TIME ZONE): Timestamp when the reminder was actually sent.
    - `created_at` (TIMESTAMP WITH TIME ZONE, DEFAULT CURRENT_TIMESTAMP): Timestamp of record creation.
    - `updated_at` (TIMESTAMP WITH TIME ZONE, DEFAULT CURRENT_TIMESTAMP): Timestamp of last record update.

4.1.2. Relationships:

- **One-to-Many**:
    - `Category` to `Item`: One category can have many items.
    - `Collector` to `Assignment`: One collector can have many assignments.
    - `Assignment` to `Reminder`: One assignment can have many reminders.
- **One-to-One**:
    - `Item` to `Assignment`: An item can have at most one active assignment at any given time. The `assignment_id` in the `items` table links to the `assignments` table.

4.2. API Development (Spring Boot)

The backend will expose a set of RESTful APIs built with Spring Boot, enabling interaction with the inventory data. The APIs will follow standard HTTP methods (GET, POST, PUT, DELETE) and provide clear, resource-oriented URIs.

4.2.1. Core Endpoints:

- **Inventory Items (`/api/items`)**:
    - `POST /api/items`: Add a new inventory item.
    - `GET /api/items`: Retrieve all inventory items (with optional filtering/pagination).
    - `GET /api/items/{id}`: Retrieve a specific inventory item by its ID.
    - `PUT /api/items/{id}`: Update an existing inventory item.
    - `DELETE /api/items/{id}`: Delete an inventory item.
- **Categories (`/api/categories`)**:
    - `POST /api/categories`: Add a new category.
    - `GET /api/categories`: Retrieve all categories.
    - `GET /api/categories/{id}`: Retrieve a specific category by its ID.
    - `PUT /api/categories/{id}`: Update an existing category.
    - `DELETE /api/categories/{id}`: Delete a category.
- **Collectors (`/api/collectors`)**:

    - `POST /api/collectors`: Add a new collector.
    - `GET /api/collectors`: Retrieve all collectors.
    - `GET /api/collectors/{id}`: Retrieve a specific collector by ID.
    - `PUT /api/collectors/{id}`: Update an existing collector.
    - `DELETE /api/collectors/{id}`: Delete a collector.
- **Assignments (`/api/assignments`)**:
    - `POST /api/assignments`: Create a new assignment for an item to a collector. This will also update the item's status.
    - `GET /api/assignments/{id}`: Retrieve a specific assignment by ID.
    - `PUT /api/assignments/{id}/return`: Mark an assignment as returned (sets `actual_return_date` and updates item status).
    - `PUT /api/assignments/{id}`: Update an existing assignment (e.g., change due date).
    - `GET /api/assignments/overdue`: Retrieve all overdue assignments.
- **Reminders (`/api/reminders`)**:
    - `GET /api/reminders`: Retrieve all reminders.
    - `POST /api/reminders`: Create a new reminder for an assignment to a collector.
    - `GET /api/reminders/{id}`: Retrieve a specific reminder by ID.
    - `POST /api/assignments/{assignmentId}/send-reminder`: Manually trigger sending a reminder for a specific assignment.

4.2.2. Reporting Endpoints:

- `GET /api/reports/inventory-levels`: Provides statistics on inventory, e.g., total items, items per category, available vs. assigned counts.
- `GET /api/reports/collector-assignments`: Provides details on items currently assigned to collectors, including overdue items.

4.3. Persistence Layer (JPA/Hibernate)

JPA (Java Persistence API) with Hibernate as the underlying ORM (Object-Relational Mapping) framework will be used to interact with the PostgreSQL database. Spring Data JPA will further simplify data access by providing repository interfaces.

4.3.1. Entity Classes:

Java classes (`Category`, `Item`, `Collector`, `Assignment`, `Reminder`) will serve as entities, annotated with `@Entity`, `@Table`, `@Id`, `@GeneratedValue`, `@Column`, and relationship annotations (`@ManyToOne`, `@OneToMany`, `@OneToOne`, `@JoinColumn`).

- **UUID Primary Keys**: All entities now use `java.util.UUID` for their primary keys, mapped to `UUID` columns in PostgreSQL, with `GenerationType.AUTO` for automatic generation.
- **Timestamps**: `@CreationTimestamp` and `@UpdateTimestamp` annotations from Hibernate will automatically manage the `createdAt` and `updatedAt` fields (of type `Instant`) for auditing.

- **Enums**: `ReminderStatus, ERole` and `Status` enums are used for type safety and clarity, mapped to `VARCHAR` columns in the database using `@Enumerated(EnumType.STRING)`.
- **Lombok**: The project will leverage Lombok annotations (`@Data`, `@NoArgsConstructor`, `@AllArgsConstructor`) to reduce boilerplate code for getters, setters, constructors, `equals()`, `hashCode()`, and `toString()`.

4.3.2. Spring Data JPA Repositories:

Interfaces extending `JpaRepository` (e.g., `ItemRepository`, `CollectorRepository`) will be created for each entity. These repositories provide out-of-the-box CRUD (Create, Read, Update, Delete) operations and allow for defining custom query methods based on method names (e.g., `findBySerialNumber(String serialNumber)`).

4.4. Security (Spring Security)

Spring Security will be implemented to provide robust authentication and authorization mechanisms, ensuring that only authorized personnel can access and modify inventory data.

- **User Authentication**: Users will authenticate with the system (e.g., via username/password). A secure authentication flow will be implemented.
- **Password Hashing**: User passwords will be securely stored using a strong, one-way hashing algorithm like BCrypt, preventing plaintext storage.
- **Authorization (Role-Based Access Control)**:
    - **Roles**: Define distinct roles such as `ADMIN`, `INVENTORY_MANAGER`, and `COLLECTOR`.
    - **Endpoint Security**: API endpoints will be secured using role-based access control. For example:
        - `ADMIN` might have full access to all CRUD operations across all resources.
        - `INVENTORY_MANAGER` might manage items, categories, and assignments, and view reports.
        - `COLLECTOR` might only be able to view their assigned items and trigger manual reminders for their own items.
    - **Session Management**: Secure session management will be implemented to protect user sessions.

4.5. Collector Tracking and Reminders

A crucial feature of the system is the automated tracking of assigned items and the generation of return reminders.

- **Logic for Tracking Overdue Items**: The system will regularly query the `assignments` table to identify items where the `return_due_date` has passed and `actual_return_date` is `NULL`.
- **Automated Reminder Generation**:
    - A scheduled task will be implemented using Spring's `@Scheduled` annotation. This task will run periodically (e.g., daily at a specific time).

    - It will identify overdue assignments and, for each, either create a new `Reminder` record or update the status of an existing pending reminder.
- **Reminder Sending Mechanism**:
    - **Email**: Reminders will be sent via email using JavaMail Sender. The `Collector` entity's `email` field will be used as the recipient.
    - **Notification System (Future Consideration)**: If time permits, integration with a more comprehensive notification system (e.g., in-app notifications, SMS gateway) can be explored.
- **Manual Reminder API**: An API endpoint will allow authorized users to manually trigger a reminder for a specific assignment, useful for immediate follow-ups.

5. Technical Stack

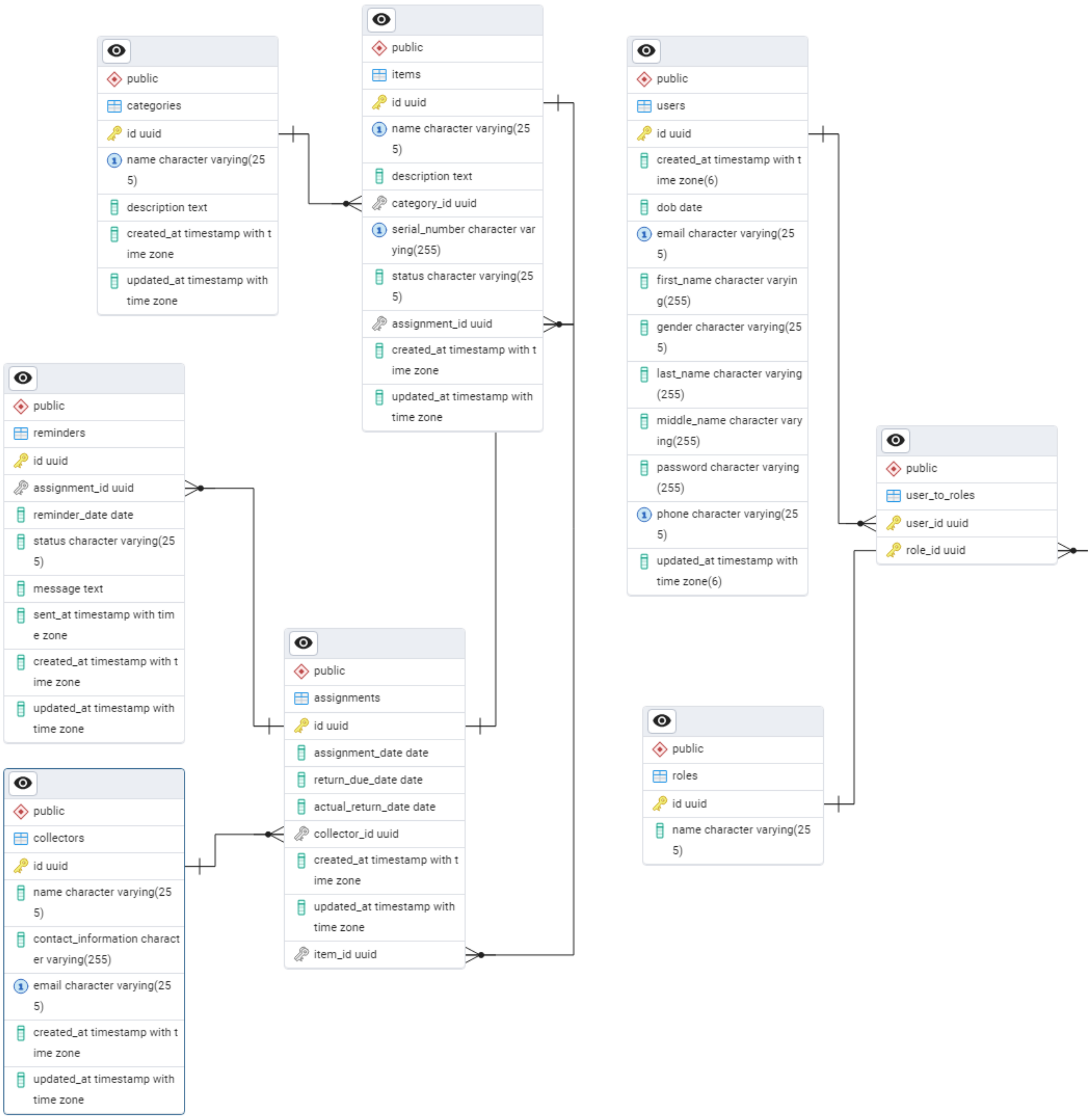
- **Backend Framework**: Spring Boot
- **Database**: PostgreSQL
- **ORM**: JPA / Hibernate
- **Data Access**: Spring Data JPA
- **Security**: Spring Security
- **Utility**: Lombok (for boilerplate code reduction)
- **Email Sending**: JavaMail Sender API (for automated and manual reminders)
- **Build Tool**: Maven
- **Language**: Java

6. Future Enhancements / Considerations

- **Frontend Integration**: Development of a user-friendly web or mobile interface to interact with the backend APIs.
- **Advanced Reporting**: More sophisticated reporting capabilities, including customizable reports, data visualization, and export options (e.g., CSV, PDF).
- **Audit Logging**: Comprehensive logging of all significant actions within the system for compliance and troubleshooting.
- **Notifications**: Integration with SMS gateways or in-app notification services for more diverse reminder delivery options.
- **User Management UI**: A dedicated UI for administrators to manage users, roles, and permissions.
- **Internationalization (i18n)**: Support for multiple languages.
- **Scalability**: Implementing caching, load balancing, and database replication strategies for high-traffic environments.
- **Deployment**: Containerization (Docker) and orchestration (Kubernetes) for easier deployment and management.

7. Conclusion

This backend system provides a robust and scalable solution for managing school inventory. By leveraging modern Spring Boot capabilities, a strong relational database, and comprehensive security features, it aims to significantly improve inventory control, reduce loss, and enhance accountability within educational institutions. The automated reminder system will ensure timely returns and minimize administrative overhead.



public
categories
id uuid
name character varying(255)
description text
created_at timestamp with time zone
updated_at timestamp with time zone
public
items
id uuid
name character varying(255)
description text
category_id uuid
serial_number character varying(255)
status character varying(255)
assignment_id uuid
created_at timestamp with time zone
updated_at timestamp with time zone
public
users
id uuid
created_at timestamp with time zone(6)
dob date
email character varying(255)
first_name character varying(255)
gender character varying(255)
last_name character varying(255)
middle_name character varying(255)
password character varying(255)
phone character varying(255)
updated_at timestamp with time zone(6)
public
user_to_roles
user_id uuid
role_id uuid
public
reminders
id uuid
assignment_id uuid
reminder_date date
status character varying(255)
message text
sent_at timestamp with time zone
created_at timestamp with time zone
updated_at timestamp with time zone
public
assignments
id uuid
assignment_date date
return_due_date date
actual_return_date date
collector_id uuid
created_at timestamp with time zone
updated_at timestamp with time zone
item_id uuid
public
roles
id uuid
name character varying(255)
public
collectors
id uuid
name character varying(255)
contact_information character varying(255)
email character varying(255)
created_at timestamp with time zone
updated_at timestamp with time zone