Java Mini Project

Problem Statement

Kickdrum is building a **Smart mobile application**.
The customer wants to build an application allowing the users to manage the devices using the same application.

Requirements

Following are the basic requirements carved out by the customer.

1. The user should be able to create one or more houses for him or her
2. The user who creates the house becomes the admin for the house.
3. Only the admin of the house should be able to add more users to his/her house.
4. The admin should be able to create rooms in the house.
5. The admin should be able to add a device to the house.
6. Users should be able to move the device from one room to another in the *same* house.
7. Users should be able to list all the houses
8. Users should be able to list all rooms and devices in the house.
9. Users should be able to add/update the address of the house.
10. The customer also maintains an inventory of all the devices which are manufactured which has the following details
    a. kickston_id // A six-digit hexadecimal number starts at 000001 to FFFFFF
    b. device_username
    c. device_password
    d. manufactore_date_time
    e. manufactore_factory_place //Example: china hub1, china hub2, etc
11. When adding a device to a house by the user, the user should provide the username and password of the device including the kickston_id which will be verified against the inventory and will be registered only if it matches.

Tasks

- Define Schema for the database which will be able to handle the above-mentioned use cases.
- Create a DDL statement to create the database, tables, and indexes, so you can recreate multiple times in the future without doing it manually
- Create a DML statement to populate the inventory table.
- Create Rest APIs that will be able to handle all the above-mentioned use cases.

General Notes

As you are developing make sure you adhere to the following best practices:-

- Naming conventions should be followed for Java Class names, variables, methods, etc
- Naming conventions should be followed for table names and table column names
- The packaging structure should be followed for the java spring application
- All the entities should have created date, modified date, and deleted date, so we can know when these entities are created/modified/deleted from the table.
    - **Reference**: https://thorben-janssen.com/persist-creation-update-timestamps-hibernate/
- Exceptions should be handled and validation of request parameters and request payload should be handled
- **All the rest APIs should be authentication required**

Test Requirements

API Endpoints

POST /api/v1/auth/register

Description : This API endpoint is used to register a new user.

Request:-

- ➔ Method: POST
- ➔ Request Body:
    - ◆ **Content-Type:** application/json
    - ◆ **Attributes:**
        - username (Type: String) - The username of the user.
        - password (Type: String) - The password for the user.
        - name (Type: String) - The full name of the user.
        - firstName (Type: String) - The first name of the user.
        - lastName (Type: String) - The last name of the user.
        - emailId (Type: String) - The email ID of the user.

Response:-

- **Success Response:** HTTP Status: 200 OK
- Body:
    - message (Type: String) - A success message.
    - token (Type: String) - The authentication token for the registered user.

POST /api/v1/house

Description : This API endpoint is used to add a new house.

Request:-

- ➔ Method: POST
- ➔ Request Body:
  - ◆ Content-Type: application/json
  - ◆ Attributes:
    - address (Type: String) - The address of the house.
    - house_name (Type: String) - The name of the house.

Response:-

- Success Response: HTTP Status: 200 OK
- Body:
  - message (Type: String) - A success message.
  - house (Type: House) - Details of the added house. *House object must have an 'id' attribute.*
  - httpStatus (Type: HttpStatus) - HTTP status of the response.

POST /api/v1/house/{houseId}/add-user

Description : This API endpoint is used to add a user to a house.

Request:-

- ➔ Method: POST
- ➔ Path Parameters:
  - ◆ houseId (Type: String) - The identifier of the house to which the user will be added.
- ➔ Request Body:
  - ◆ Content-Type: application/json
  - ◆ Attributes:
    - username (Type: String) - The username of the user to be added to the house

Response:-

- Success Response: HTTP Status: 200 OK
- Body:
  - message (Type: String) - A success message.
  - object (Type: String) - Additional information about the added user.
  - httpStatus (Type: HttpStatus) - HTTP status of the response.

GET /api/v1/house/

Description : This API endpoint retrieves a list of houses.

Request:-

- ➔ Method: GET
- ➔ Request Body:
  - ◆ **Content-Type:** application/json

Response:-

- Success Response: HTTP Status: 200 OK
- Body:
  - message (Type: String) - A success message.
  - houses (Type: String) - JSON representation of the list of houses. *Must have houses' names and addresses.*
  - httpStatus (Type: HttpStatus) - HTTP status of the response.

PUT /api/v1/house

Description : This API endpoint is used to update the address of a house.

Request:-

- ➔ Method: PUT
- ➔ Query Parameter:
  - ◆ houseId (Type: String) - The identifier of the house for which the address will be updated.
- ➔ Request Body:
  - ◆ **Content-Type:** application/json
  - ◆ Attributes:
    - newAddress (Type: String) - The new address to be set for the house.

Response:-

- Success Response: HTTP Status: 200 OK
- Body:
  - message (Type: String) - A success message.
  - object (Type: String) - Additional information about the added house.
  - httpStatus (Type: HttpStatus) - HTTP status of the response.

GET /api/v1/house/{houseId}

Description : This API endpoint retrieves all rooms and devices associated with a specific house.

Request:-

- ➔ Method: GET
- ➔ Path Parameter:
  - houseId (Type: String) - The identifier of the house for which the address will be updated.

Response:-

- Success Response: HTTP Status: 200 OK
- Body:
  - message (Type: String) - A success message.
  - roomsAndDevices (Type: String) - JSON representation of rooms and devices in the specified house. *Must have : House id, name and address; Room id and name; Device kickstone id and name.*
  - httpStatus (Type: HttpStatus) - HTTP status of the response.

POST /api/v1/room

Description : This API endpoint is used to add rooms to a house.

Request:-

- ➔ Method: POST
- ➔ Query Parameter:
  - houseId (Type: String) - The identifier of the house to which the rooms will be added.
- ➔ Request Body:
  - Content-Type: application/json
  - Attributes:
    - room_name (Type: String) - The name of the room to be added.

Response:-

- Success Response: HTTP Status: 200 OK
- Body:
  - message (Type: String) - A success message.
  - room (Type: Room) - Details of the added room. *Room object must have an 'id' attribute.*
  - httpStatus (Type: HttpStatus) - HTTP status of the response.

GET /api/v1/inventory

Description : This API endpoint retrieves the list of items in the inventory.

Request:-

➔ Method: GET

Response:-

- Success Response: HTTP Status: 200 OK
- Body:
    - inventory (Type: String) - JSON representation of the list of items in the inventory. *Must have Device kickstoneId, name and password*
    - httpStatus (Type: HttpStatus) - HTTP status of the response.

POST /api/v1/inventory

Description : This API endpoint is used to add an item to the inventory.

Request:-

➔ Method: POST

➔ Request Body:

◆ Content-Type: application/json

◆ Attributes:

- kickston_id (Type: String) - The identifier for the inventory item.
- device_username (Type: String) - The username associated with the device.
- device_password (Type: String) - The password associated with the device.
- manufacture_date_time (Type: String) - The manufacturing date and time of the device (format: "yyyy-MM-dd'T'HH:mm:ss").
- manufacture_factory_place (Type: String) - The place where the device was manufactured.

Response:-

- Success Response: HTTP Status: 200 OK
- Body:
    - message (Type: String) - A success message.
    - object (Type: String) - Additional information about the added item.
    - httpStatus (Type: HttpStatus) - HTTP status of the response.

POST /api/v1/device/register

Description : This API endpoint is used to register a new device.

Request:-

- ➔ Method: POST
- ➔ Request Body:
  - ◆ Content-Type: application/json
  - ◆ Attributes:
    - kickston_id (Type: String) - The identifier for the device.
    - device_username (Type: String) - The username associated with the device.
    - device_password (Type: String) - The password associated with the device.

Response:-

- Success Response: HTTP Status: 200 OK
- Body:
  - message (Type: String) - A success message.
  - object (Type: String) - Additional information about the added item.
  - httpStatus (Type: HttpStatus) - HTTP status of the response.

POST /api/v1/device/add

Description : This API endpoint is used to add a device to a house.

Request:-

- ➔ Method: POST
- ➔ Request Body:
  - ◆ Content-Type: application/json
  - ◆ Attributes:
    - houseId (Type: String) - The identifier of the house to which the device will be added.
    - roomId (Type: String) - The identifier of the room in the house where the device will be placed.
    - kickstonId (Type: String) - The identifier of the device to be added.

Response:-

- Success Response: HTTP Status: 200 OK
- Body:
  - message (Type: String) - A success message.
  - object (Type: String) - Additional information about the added item.
  - httpStatus (Type: HttpStatus) - HTTP status of the response.

Debugging

- Go through the test cases and modify your API responses accordingly.
- Every test case performs “andExpect” where it matches some string data in “jsonPath”
  - Make sure your responses expose the same json path formats.
  - Log your response or test cases accordingly to debug any failures.