Spring Boot Cat Café

Objective:

Develop a **REST API** for a **Cat Café Shop** using **Spring Boot**.

The system has **two roles**:

- **ROLE_ADMIN**: Manages café reservations and cat adoptions.
- **ROLE_USER**: Makes café reservations and applies to adopt cats.

You can use `Basic Auth` authentication method, that is, username and password, for all requests.

Requirements

Database Tables

- Create a new database called `cat_cafe`
- Implement `users`, `reservations`, and `cat_adoptions` tables with the following columns:

  `users` (Represents regular users (customers) and admins)

  - `id`: `BIGINT`, `PRIMARY KEY`, `AUTO_INCREMENT`
  - `username`: `VARCHAR(255)`, `NOT NULL`, `UNIQUE`
  - `password`: `VARCHAR(255)`, `NOT NULL`

  `reservations` (Customers book café visits)

  - `id`: `BIGINT`, `PRIMARY KEY`, `AUTO_INCREMENT`
  - `user_id`: `BIGINT`, `FK`
  - `date_of_reservation`: `DATE`, `NOT NULL`
  - `time_slot`: `VARCHAR(50)`, `NOT NULL`
    * Example: `10:00 AM - 11:00 AM`
  - `num_guests`: `INTEGER`, `NOT NULL`

  `cat_adoptions` (Customers apply to adopt café cats)

  - `id`: `BIGINT`, `PRIMARY KEY`, `AUTO_INCREMENT`
  - `user_id`: `BIGINT`, `FK`

    - `cat_name`: `VARCHAR(50)`, `NOT NULL`
    - `status`: `VARCHAR(50)`, `NOT NULL`
        - Will either be `PENDING`, `APPROVED` or `REJECTED`
    - `application_date`: `TIMESTAMP`, `NOT NULL`

- You must also add `roles` and `users_roles` tables - you should already know how to do this

- **Relationships:**

    - A customer can make one or many reservations. One particular reservation belongs to only one person.
    - A customer can submit one or many cat adoption applications. One particular cat adoption application belongs to only one person.

Entity Models

- Based on the tables you have created, create Entity classes in your Java code

---

DTOs & Validation

- Use **DTOs** for transfering data from client to server, and vice versa; you will need separate DTOs for request and response, for example, `UserRequestDTO` and `UserResponseDTO`
- Apply **Bean Validation** annotations for input validation.
- Minimum DTOs you'll need to implement, their fields and validation (may need more later):
    - `UserRequestDTO`:
        - `username`
            - Cannot be null
            - Must be at least 4 characters long
        - `password`
            - Cannot be null
            - Must be at least 10 characters long
        - `roles`:

        - Cannot be null
  - `UserResponseDTO`:
    * `id`
    * `username`
    * `roles`
  - `ReservationRequestDTO`:
    * `dateOfReservation`
      - Cannot be null
      - Must be in the future; not today
    * `timeSlot`
      - Cannot be null
    * `numGuests`
      - Cannot be null
      - Must be between 1 and 4
  - `ReservationResponseDTO`:
    * `id`
    * `user`
    * `dateOfReservation`
    * `timeSlot`
    * `numGuests`
  - `CatAdoptionRequestDTO`:
    * `catName`
      - Not null
      - Starts with uppercase letter
  - `CatAdoptionResponseDTO`:
    * `id`
    * `user`
    * `catName`
    * `status`
    * `applicationDate`

REST API Endpoints

Public Endpoints

- `POST /auth/register` → New user registers (default ROLE_USER role).
  - Username must be unique

  - **Request:**

    ```
    {
      "username": "catlover123",
      "password": "securepass",
      "roles": [
        {
          "id": 1
        }
      ]
    }
    ```

    Response (Success - 201 Created):

    ```
    {
      "id": 1,
      "username": "catlover123",
      "roles": [
        {
          "id": 1,
          "name": "ROLE_USER"
        }
      ]
    }
    ```

  - Keep in mind that if you want to register ROLE_ADMIN, you should not forget to include ROLE_USER
  - Client should expect to see validation errors if some field does not match validation, `400 Bad Request`

Customer Endpoints

- `POST /reservations` → Make a café reservation.
  - **Request:**

    ```
    {
      "dateOfReservation": "2025-03-01",
      "timeSlot": "14:00 - 15:00",
      "numGuests": 2
    }
    ```

Response (Success - 201 Created):

```
{
  "id": 10,
  "user": {
    "id": 1,
    "username": "catlover123",
    "roles": [
      {
        "id": 1,
        "name": "ROLE_USER"
      }
    ]
  },
  "dateOfReservation": "2025-03-01",
  "timeSlot": "14:00-15:00",
  "numGuests": 2
}
```

  - Client should expect to see validation errors if some field does not match validation, `400 Bad Request`

- `GET /reservations` → View own reservations.
  - Show reservations of only currently logged in user
  - **Response (Success - 200 OK):**

```
[
  {
    "id": 10,
    "user": {
      "id": 1,
      "username": "catlover123",
      "roles": [
        {
          "id": 1,
          "name": "ROLE_USER"
        }
      ]
    },
    "dateOfReservation": "2025-03-01",
    "timeSlot": "14:00-15:00",
    "numGuests": 2
  },
  {
    "id": 12,
    "user": {
      "id": 1,
      "username": "catlover123",
      "roles": [
        {
          "id": 1,
          "name": "ROLE_USER"
        }
      ]
    },
    "dateOfReservation": "2025-03-05",
    "timeSlot": "10:00-11:00",
    "numGuests": 1
  }
]
```

  - If list of reservations is empty, should still return `200 OK`

- `POST /adoptions/apply` → Apply to adopt a cat.
  - User cannot apply for adoption of same cat twice
  - **Request:**

```
{
  "catName": "Whiskers"
}
```

Response (Success - 201 Created):

```
{
  "id": 5,
  "user": {
    "id": 1,
    "username": "catlover123",
    "roles": [
      {
        "id": 1,
        "name": "ROLE_USER"
      }
    ]
  },
  "catName": "Whiskers",
  "status": "PENDING",
  "applicationDate": "2025-02-11T14:23:00"
}
```

  - Client should expect to see validation errors if some field does not match validation, `400 Bad Request`

- `GET /adoptions` → View adoption status.
  - Only show for currently authenticated user, his own adoptions and status, not all users
  - **Response (Success - 200 OK):**

```
[
  {
    "id": 5,
    "user": {
      "id": 1,
      "username": "catlover123",
      "roles": [
        {
          "id": 1,
          "name": "ROLE_USER"
        }
      ]
    },
    "catName": "Whiskers",
    "status": "PENDING",
    "applicationDate": "2025-02-11T14:23:00"
  },
  {
    "id": 8,
    "user": {
      "id": 1,
      "username": "catlover123",
      "roles": [
        {
          "id": 1,
          "name": "ROLE_USER"
        }
      ]
    },
    "catName": "Mittens",
    "status": "APPROVED",
    "applicationDate": "2025-02-11T11:15:38"
  }
]
```

  - If the user has no adoptions, an empty list and `200 OK` should be returned

Admin Endpoints

- `GET /adoptions/pending` → View all pending adoptions, of **all** users in the system.
  - **Response (Success - 200 OK):**

```
[
  {
    "id": 5,
    "user": {
      "id": 1,
      "username": "catlover123",
      "roles": [
        {
          "id": 1,
          "name": "ROLE_USER"
        }
      ]
    },
    "catName": "Whiskers",
    "status": "PENDING"
  },
  {
    "id": 7,
    "user": {
      "id": 3,
      "username": "meowfan99",
      "roles": [
        {
          "id": 1,
          "name": "ROLE_USER"
        }
      ]
    },
    "catName": "Fluffy",
    "status": "PENDING"
  }
]
```

- `PUT /adoptions/{adoptionId}/approve` → Approve an adoption (**Admin only**).
  - **Request**

    ```
    {
      "status": "APPROVED"
    }
    ```

    Response (Success - 200 OK):

    ```
    {
      "id": 5,
      "catName": "Whiskers",
      "status": "APPROVED"
    }
    ```

  - If an adoption id cannot be found, return `404 Not Found`, alongside a message:

    ```
    {
    ```

```
      "error": "Adoption request not found."
    }
```

- `DELETE /reservations/{reservationId}` → Cancel a reservation (**Admin only**).
  - **Response (Success - 200 OK):**
    ```
    {
      "message": "Reservation successfully canceled."
    }
    ```
  - If reservation provided by id cannot be found, return `404 Not Found`

Security & Authorization

- Implement **Basic Authentication.**
- Users have roles: **ROLE_USER** or **ROLE_ADMIN**.
- Only **ADMINs** can approve adoptions and cancel reservations.

Extra endpoint

- Implement an endpoint `GET /adoptions/approved` to list all approved adoptions (**Admin only**).