In continuation with the last topic assignment on creating a REST API, create a Customer entity with the following fields and apply the given validations.

Field Name	Validations
firstName	Must be non-null and contain at least 3 characters.
lastName	Must be non-null and contain at least 3 characters.
email	Must be non-null and formatted as a valid email address.
password	Must be non-null, 6-20 characters long, and alphanumeric.
confirmPassword	Must be non-null, 6-20 characters long, alphanumeric, and match the password.
mobileNo	Must be non-null and not empty.
salary	Must be between 10,000 and 50,000.

In continuation with Assignment No. 2, create a custom validation for the mobileNo field in the Customer entity.

The validation for mobileNo should:

- Contain only digits.
- Be exactly 10 characters long.

Note: Upload the assignment in PDF format containing screenshots of the code and output screenshots.

Build a CRUD Rest API Project using the MySQL Database and JPA concept on Customer Entity created in the previous assignment.

1. Create a CustomerController to handle CRUD operations for the Customer entity.
2. Create a CustomerService to manage business logic and interact with the CustomerRepository.
3. In the CustomerRepository interface, which should extend JpaRepository, implement the following query methods:
    - Find Customers by First Name
    - Find Customers by Last Name
    - Find Customers by Email
    - Find Customers by Salary Range
    - Find Customers with Salary Greater Than a Specific Value
    - Find Customers by Name Starting With
    - Find Customers by Email Domain
4. Test the Query Methods using Postman

With reference to the Customer entity class created in the previous assignments, apply @CreatedDate and @LastModifiedDate to your Customer entity class.

Hint:

Apply in the following way:

@CreateDate

private Instant CreatedAt;

@LastModifiedDate

private Instant ModifiedAt;

With reference to the Customer Entity class created in the last assignments ,apply @CreatedDate and @LastModifiedDate in your Customer Entity class.

Hint:

Apply in the following way:

@CreateDate

private Instant CreatedAt;

@LastModifiedDate

private Instant ModifiedAt;

Create a new Data Rest Project as per below details given below.

Project Name: Application

Classname: Customer

Attribute Names for Customer class: customerid , firstName, lastName, email, address

Create Repository class for the Customer Entity to map the following APIs with the query methods

HTTP Methods	**routes**	**Action**
POST	/customers	Create a new Customer
GET	/customers	Read all customers
GET	/customers/{customerid}	Read a single Customer
DELETE	/customers/{customerid}	Delete a single Customer
PUT	/customers/{customerid}	Update a single Customer

Create a new Data Rest Project as per below details given below.

Project Name: Application

Classname: Customer

Attribute Names for Customer class: customerid , firstName, lastName, email, address

Create Repository class for the Customer Entity to map the following APIs with the query methods

HTTP Methods	routes	Action
POST	/customers	Create a new Customer
GET	/customers	Read all customers
GET	/customers/{customerid}	Read a single Customer
DELETE	/customers/{customerid}	Delete a single Customer
PUT	/customers/{customerid}	Update a single Customer

Implement paging and sorting for the Customer entity as follows:

Routes:

- http://localhost:8080/customers?pageSize=5
- http://localhost:8080/customers?pageSize=5&pageNo=1
- http://localhost:8080/customers?pageSize=5&pageNo=2
- http://localhost:8080/customers?pageSize=5&pageNo=1&sortBy=email
- http://localhost:8080/customers?pageSize=5&pageNo=1&sortBy=firstName

Using the previously created Customer entity and the new Orders entity, create a one-to-one mapping for both entities.

Attribute for Orders Class (orders.java):

- private Integer orderId
- private String orderDate
- private Double amount

Create repository classes for the Order entity.

Create One-to-Many and Many-to-One mappings for Category and Product.

Attributes for Product class (product.java):

- private Integer id
- private String name
- private String manufacture
- private Double price
- private String description

Attributes for Category class (category.java):

- private Integer id
- private String name
- private String description

Using the previously created Orders and Product classes, create a many-to-many mapping between them.

Implements Concepts of projection UserWithNoPassword on Your Customer Entity.

Hint:

1. Projection Name:@Projection(name="nopassword",types=(User.class))
2. Interface Name: UserWithNoPassword
   - String getName()
   - String getEmail()
   - String getMobile()

Assignment Last

Objective:

Design and develop a full-stack web application for managing books using React.js for the frontend and Spring Boot for the backend. The system should allow users to register, log in, and perform CRUD (Create, Read, Update, Delete) operations on a collection of books.

Requirements:

User Authentication:

Register:

Allow new users to create an account using a registration form that requires a username, email, and password. Implement password hashing and validation on the backend.

Login: I

Implement a login functionality that requires a username and password.

Logout:

Provide a logout option to clear the user's session.

Book Management:

Add a Book: Authenticated users should be able to add a new book by providing details such as the book title, author, publication date, genre, and a brief description.

View Books: Implement a feature to display a list of all books. The list should be paginated and include basic details of each book.

View Book Details: Allow users to click on a book in the list to view its detailed information.

Update a Book: Authenticated users should be able to update the details of a book they have added.

Delete a Book: Authenticated users should be able to delete a book they have added.

Frontend Requirements (React.js):

Use React.js to create a responsive and user-friendly interface.

Implement routing to navigate between different pages (e.g., Login, Register, Book List, Book Details, Add/Edit Book).

Use state management (React Context API) to manage application state, including user authentication status and book data.

Implement form validation for all input fields.

Display error messages and success notifications where appropriate.

Backend Requirements (Spring Boot Rest API):

Develop a RESTful API using Spring Boot Framework to handle all backend logic.

Create models for User and Book with appropriate fields.

Implement API endpoints for user registration, login, and logout.

Implement API endpoints for CRUD operations on books (e.g., /api/books/, /api/books/<id>/).

Database:

Use a relational database (MySQL) to store user and book data.

Ensure that appropriate relationships (e.g., one-to-many between User and Book) are defined in the models.