To create a **Spring Boot project with centralized data masking using AOP** and a **REST Controller**, follow this guide:

---

1. Project Setup

1. **Create a Spring Boot Project**:
    - Use **Spring Initializr** or your favorite IDE.
    - Include the following dependencies:
        - **Spring Web**
        - **Spring Boot DevTools**
        - **Lombok** (Optional, for reducing boilerplate code)
        - **Spring Data JPA**
        - **H2 Database** (or any database of your choice)

Directory Structure:

```
src/main/java/com/example/aopmasking
├── annotation
│   └── MaskSensitiveData.java
│   └── MaskResponse.java
├── aspect
│   └── DataMaskingAspect.java
├── controller
│   └── UserController.java
├── dto
│   └── UserResponseDto.java
├── entity
│   └── User.java
├── repository
│   └── UserRepository.java
├── service
│   └── UserService.java
├── util
│   └── DataMaskingUtil.java
└── Application.java
```

2.

---

2. Code Implementation

Annotation for Masking Sensitive Data

To identify fields for masking:

```
package com.example.aopmasking.annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface MaskSensitiveData {
}
```

Annotation for Masking API Responses

To apply masking to specific endpoints:

```
package com.example.aopmasking.annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface MaskResponse {
}
```

Utility for Data Masking

A utility to handle masking logic:

```
package com.example.aopmasking.util;

import java.lang.reflect.Field;

public class DataMaskingUtil {
```

```
    public static void maskFields(Object object) {
        if (object == null) return;

        Field[] fields = object.getClass().getDeclaredFields();
        for (Field field : fields) {
            if
(field.isAnnotationPresent(com.example.aopmasking.annotation.MaskSensitiveData.class)) {
                field.setAccessible(true);
                try {
                    Object value = field.get(object);
                    if (value instanceof String) {
                        field.set(object, maskString((String) value));
                    }
                } catch (IllegalAccessException e) {
                    e.printStackTrace();
                }
            }
        }
    }

    private static String maskString(String value) {
        if (value == null || value.length() <= 4) return "****";
        return value.substring(0, 2) + "****" + value.substring(value.length() - 2);
    }
}
```

Aspect for Centralized Masking

A Spring AOP aspect for centralized masking:

```
package com.example.aopmasking.aspect;

import com.example.aopmasking.util.DataMaskingUtil;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.stereotype.Component;

import java.util.Collection;

@Aspect
@Component
```

```
public class DataMaskingAspect {

    @Around("@annotation(com.example.aopmasking.annotation.MaskResponse)")
    public Object applyMasking(ProceedingJoinPoint joinPoint) throws Throwable {
        Object result = joinPoint.proceed();

        if (result instanceof Collection) {
            ((Collection<?>) result).forEach(DataMaskingUtil::maskFields);
        } else {
            DataMaskingUtil.maskFields(result);
        }

        return result;
    }
}
```

Entity Class

Define the database entity without any masking logic:

```
package com.example.aopmasking.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.Data;

@Entity
@Data
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    private String email;

    private String phoneNumber;
}
```

Repository

Define a JPA repository for database operations:

```
package com.example.aopmasking.repository;

import com.example.aopmasking.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {
}
```

Service Layer

Business logic to fetch data from the database:

```
package com.example.aopmasking.service;

import com.example.aopmasking.entity.User;
import com.example.aopmasking.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    public List<User> getAllUsers() {
        return userRepository.findAll();
    }

    public User getUserById(Long id) {
        return userRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("User not found"));
    }
}
```

Controller

Expose REST endpoints for the client:

```
package com.example.aopmasking.controller;

import com.example.aopmasking.annotation.MaskResponse;
import com.example.aopmasking.entity.User;
import com.example.aopmasking.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController
public class UserController {

    @Autowired
    private UserService userService;

    @MaskResponse
    @GetMapping("/users")
    public List<User> getAllUsers() {
        return userService.getAllUsers();
    }

    @MaskResponse
    @GetMapping("/users/{id}")
    public User getUserById(@PathVariable Long id) {
        return userService.getUserById(id);
    }
}
```

---

3. Test the API

Save Test Data: Use H2 console or JPA methods to insert data like:

```
 [
    {
        "id": 1,
        "name": "John Doe",
```

```
    "email": "johndoe@example.com",
    "phoneNumber": "1234567890"
  },
  {
    "id": 2,
    "name": "Jane Smith",
    "email": "janesmith@example.com",
    "phoneNumber": "9876543210"
  }
]
```


1.

Endpoint Without Masking:
(Remove `@MaskResponse` annotation temporarily for testing.)

```
GET /users
```


```
[
  {
    "id": 1,
    "name": "John Doe",
    "email": "johndoe@example.com",
    "phoneNumber": "1234567890"
  }
]
```


2.

Endpoint With Masking: (With `@MaskResponse` applied.)

```
GET /users
```


```
[
  {
    "id": 1,
    "name": "John Doe",
    "email": "jo****@example.com",
    "phoneNumber": "12******90"
  }
]
```


3.

Benefits of This Approach

1. **Centralized Logic**: Masking is handled by the aspect, ensuring uniformity across APIs.
2. **No Modification to Entity Classes**: Entities remain clean, adhering to separation of concerns.
3. **Reusability**: The masking logic can be applied to multiple controllers or APIs with minimal effort.
4. **Scalability**: Easily handles complex data structures like collections and nested objects.

Let me know if you need additional help or customizations!