10 Mistakes Developers Make While Writing REST APIs in Spring Boot

Palmurugan C
@palmuruganc

1. Ignoring HTTP Status Code

Mistake: Always returning 200 OK for every response, even for errors.

For example, sending a 200 response with an error message in the body instead of using 400 Bad Request or 404 Not Found.

Fix: Use appropriate HTTP status code like:

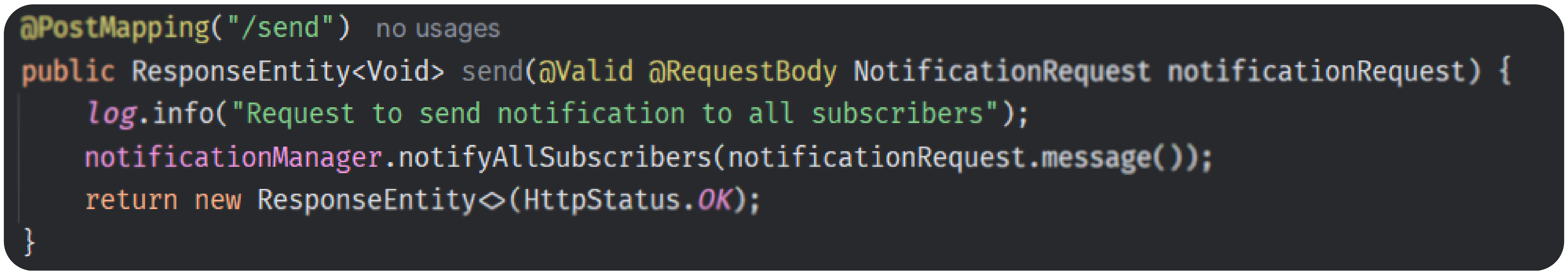
- 201 - Created for successful resource creation
- 400 - Bad Request for validation errors
- 404 - Not Found for missing resources
- 500 - Internal Server Error for unexpected issues.

2. Not Using Proper Request Validation

Mistake: Trusting incoming data without validating it.

Ex: Accepting invalid data without checks, leading to errors downstream..

Fix: Use ***@Valid*** and ***@Validated*** annotations with DTOs, and Spring's BindingResult for detailed error handling



```
@PostMapping("/send")  no usages
public ResponseEntity<Void> send(@Valid @RequestBody NotificationRequest notificationRequest) {
    log.info("Request to send notification to all subscribers");
    notificationManager.notifyAllSubscribers(notificationRequest.message());
    return new ResponseEntity<>(HttpStatus.OK);
}
```

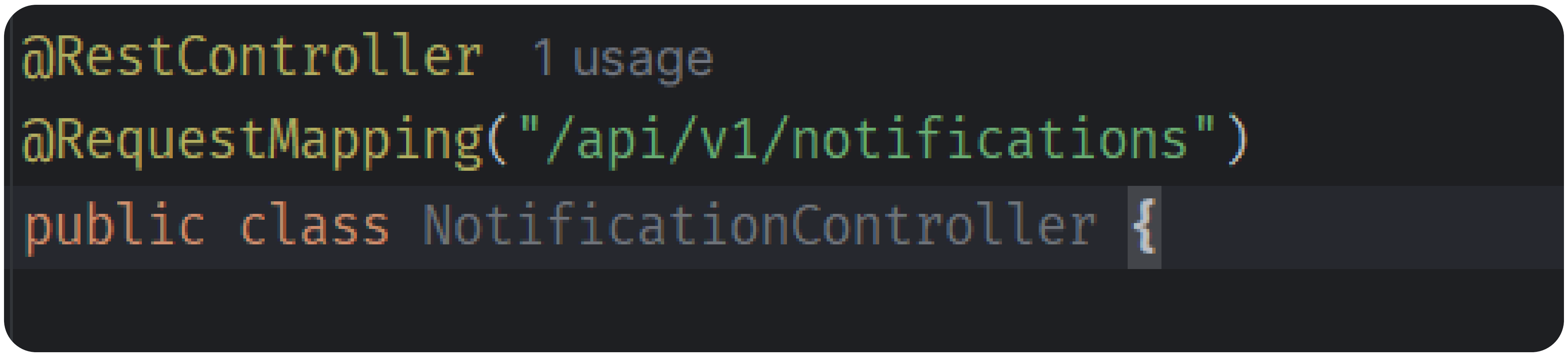
Palmurugan C
@palmuruganc

3. Ignoring API Versioning

Mistake: Developing APIs without versioning makes it hard to manage backward compatibility.

Fix: Implement ***API versioning*** using

- URI versioning (e.g., /v1/users).
- Header versioning (e.g., Accept: application/vnd.company.app-v1+json).



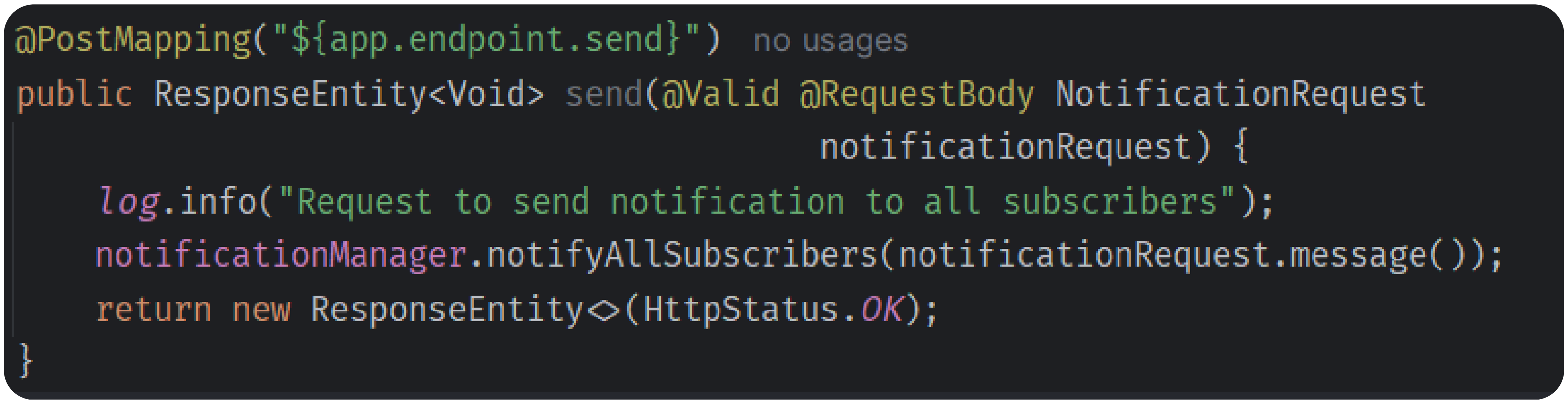
```
@RestController  1 usage
@RequestMapping("/api/v1/notifications")
public class NotificationController {
```

Palmurugan C
@palmurugancc

4. Hardcoding Endpoints and URLs

Mistake: Writing URLs, paths, or service addresses directly in code.

Fix: Use properties files (application.yml) for ***externalizing configurations*** and @Value or Environment to read them.

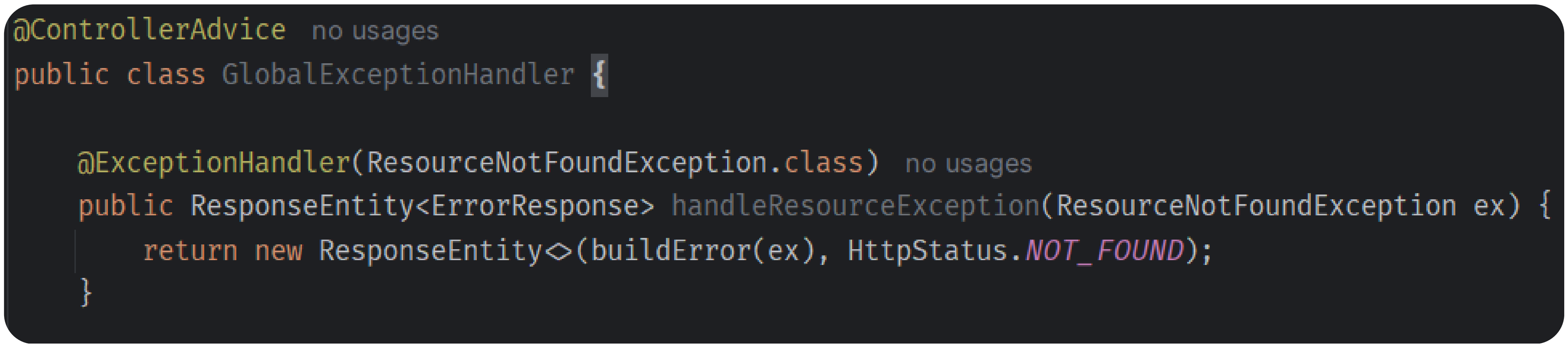


```
@PostMapping("${app.endpoint.send}")  no usages
public ResponseEntity<Void> send(@Valid @RequestBody NotificationRequest
                                          notificationRequest) {
    log.info("Request to send notification to all subscribers");
    notificationManager.notifyAllSubscribers(notificationRequest.message());
    return new ResponseEntity<>(HttpStatus.OK);
}
```

5. Improper Exception Handling

Mistake: Letting exceptions propagate to the client without a structured response.

Fix: Use ***@ControllerAdvice*** and ***@ExceptionHandler*** to standardize error handling.



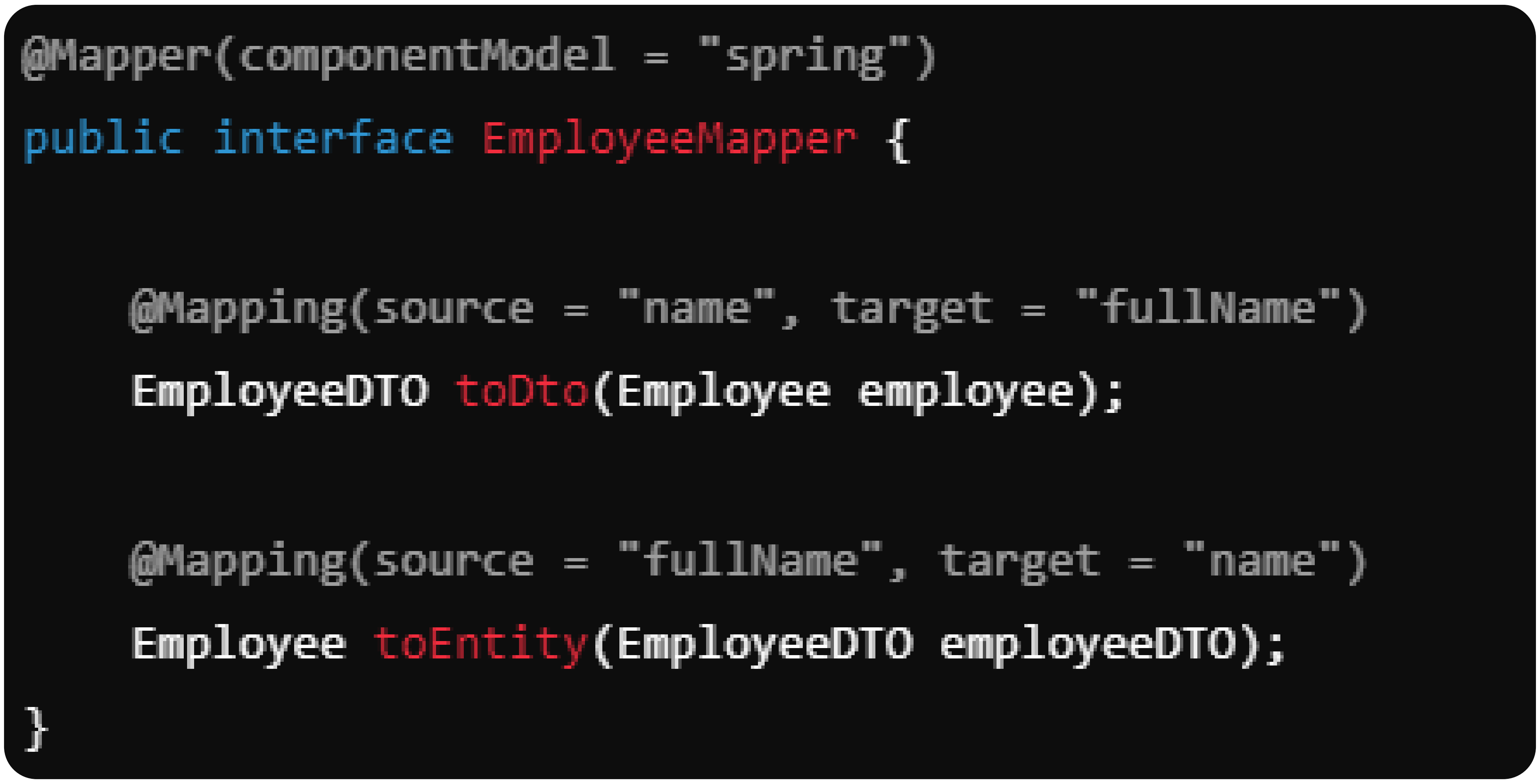
```
@ControllerAdvice  no usages
public class GlobalExceptionHandler {

    @ExceptionHandler(ResourceNotFoundException.class)  no usages
    public ResponseEntity<ErrorResponse> handleResourceException(ResourceNotFoundException ex) {
        return new ResponseEntity<>(buildError(ex), HttpStatus.NOT_FOUND);
    }
```

Palmurugan C
@palmuruganc

6. Complicating DTO & Entity Mapping

Mistake: Mistake: Exposing database entities directly in the API response.

Fix: Use DTOs (Data Transfer Objects) to decouple API layers from the database schema. Use libraries like ***MapStruct*** or ***ModelMapper*** for mapping.



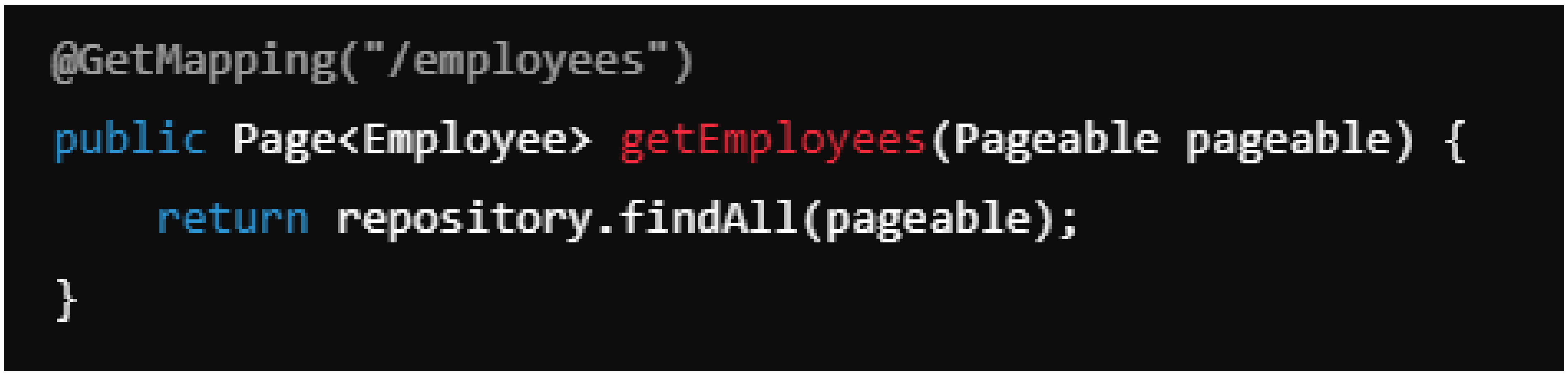
```
@Mapper(componentModel = "spring")
public interface EmployeeMapper {

    @Mapping(source = "name", target = "fullName")
    EmployeeDTO toDto(Employee employee);

    @Mapping(source = "fullName", target = "name")
    Employee toEntity(EmployeeDTO employeeDTO);
}
```

Palmurugan C
@palmuruganc

7. Ignoring Pagination and Filtering

Mistake: Returning all records in a single response, leading to performance issues.

Fix: Implement pagination and filtering using Spring Data's ***Pageable*** and ***query parameters.***



```java
@GetMapping("/employees")
public Page<Employee> getEmployees(Pageable pageable) {
    return repository.findAll(pageable);
}
```

Palmurugan C
@palmurugance

8. Ignoring Security Best Practices

Mistake: Exposing APIs without securing them, allowing unauthorized access.

Fix:

- Use Spring Security to secure endpoints.
- Implement ***OAuth2 or JWT*** for authentication and authorization.
- Avoid exposing sensitive information in responses (e.g., passwords, internal IDs).

Palmurugan C
@palmuruganc

9. Overlooking API Documentation

Mistake: Not documenting APIs, leading to confusion for other developers.

Fix:

- Use tools like ***Swagger/OpenAPI*** for auto-generating documentation.
- Add Swagger dependencies and configure:

10. Forgetting HATEOAS

Mistake: Returning plain JSON without navigational links.

Fix:

- Use Spring HATEOAS to include links for resource actions (e.g., self-link, related resources).

```json
{
    "id": 1,
    "name": "John Doe",
    "role": "Developer",
    "_links": {
        "self": { "href": "http://localhost:8080/employees/1" },
        "all-employees": { "href": "http://localhost:8080/employees" }
    }
}
```

Connect with me

"Stay ahead with cutting-edge technical tips and best practices—connect with me on LinkedIn today!"

@palmuruganc

Palmurugan C
@palmuruganc