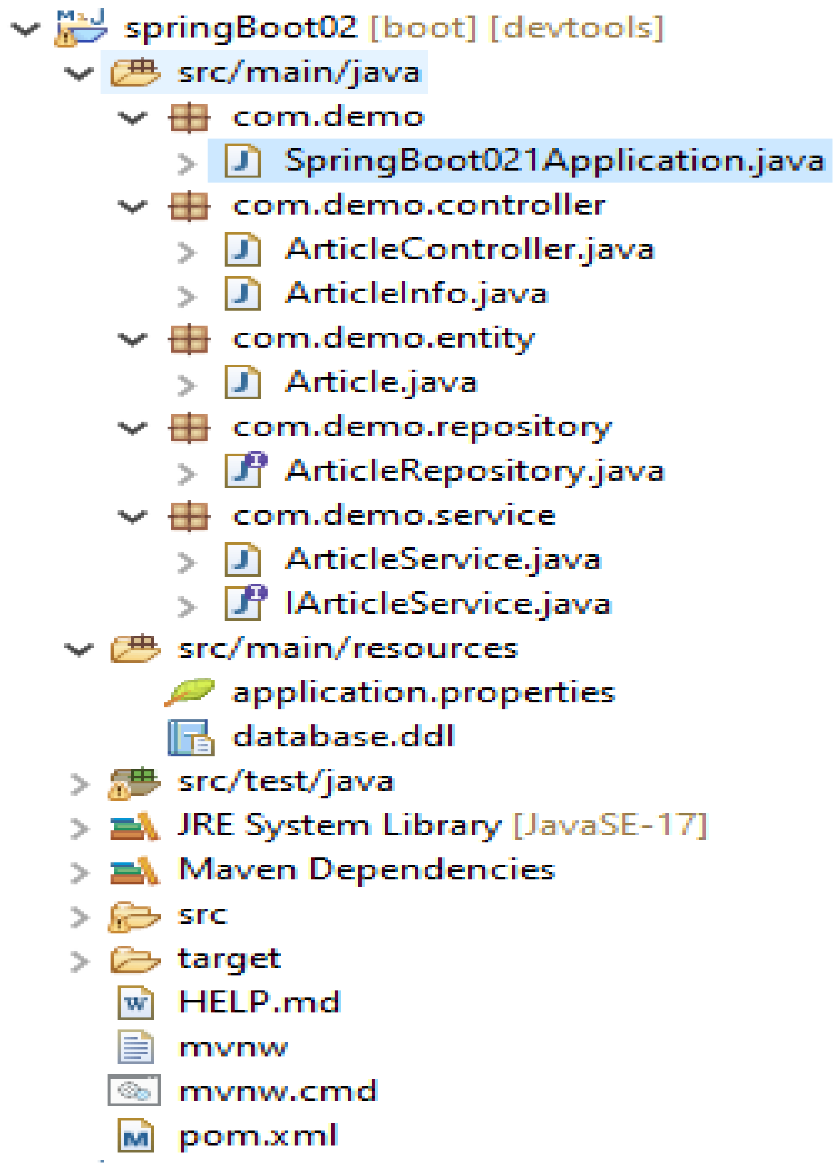
Spring Boot
REST CRUD + DATA JPA + MySql
Json Response

springboot02

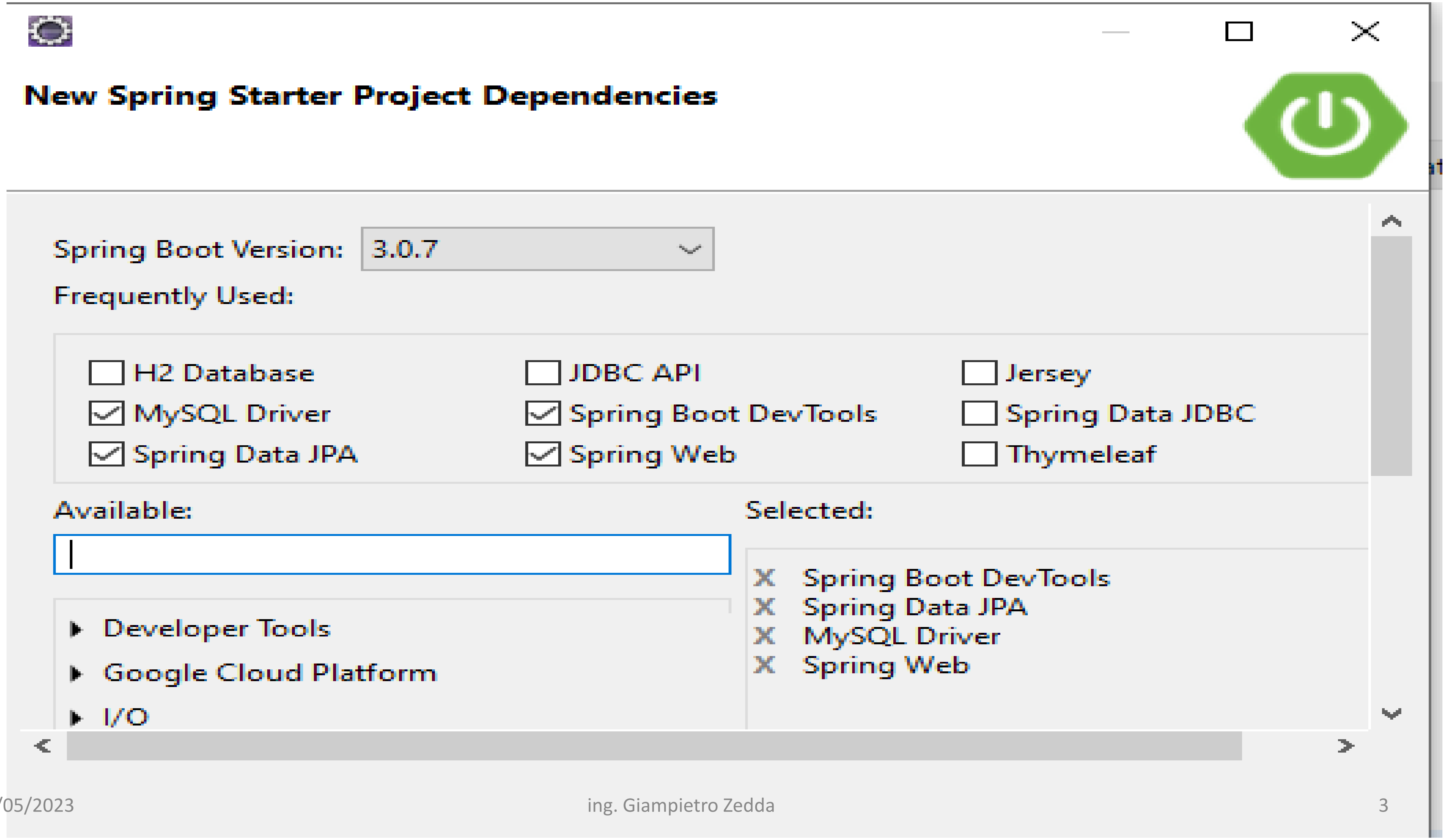
Spring Boot Eclipse Project Structure



```
springBoot02 [boot] [devtools]
  src/main/java
    com.demo
      SpringBoot021Application.java
    com.demo.controller
      ArticleController.java
      ArticleInfo.java
    com.demo.entity
      Article.java
    com.demo.repository
      ArticleRepository.java
    com.demo.service
      ArticleService.java
      IArticleService.java
  src/main/resources
    application.properties
    database.ddl
  src/test/java
  JRE System Library [JavaSE-17]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

spring-boot-starter-parent: POM principale per la gestione delle dipendenze.
spring-boot-starter-web: strumento di avviamento per la creazione di applicazioni *Web* e *REST*. Utilizza il server *Tomcat* come server incorporato predefinito.
spring-boot-starter-data-jpa: Starter per i dati di spring JPA con hibernate.
spring-boot-devtools: fornisce strumenti per sviluppatori. Questi strumenti sono utili nella modalità di sviluppo dell'applicazione. Una delle funzionalità dello strumento per sviluppatori è il riavvio automatico del server per qualsiasi modifica al codice.
spring-boot-maven-plugin: viene utilizzato per creare JAR eseguibile dell'applicazione.

Spring Boot Modules



New Spring Starter Project Dependencies

Spring Boot Version: 3.0.7

Frequently Used:

- [ ] H2 Database
- [ ] JDBC API
- [ ] Jersey
- [✓] MySQL Driver
- [✓] Spring Boot DevTools
- [ ] Spring Data JDBC
- [✓] Spring Data JPA
- [✓] Spring Web
- [ ] Thymeleaf

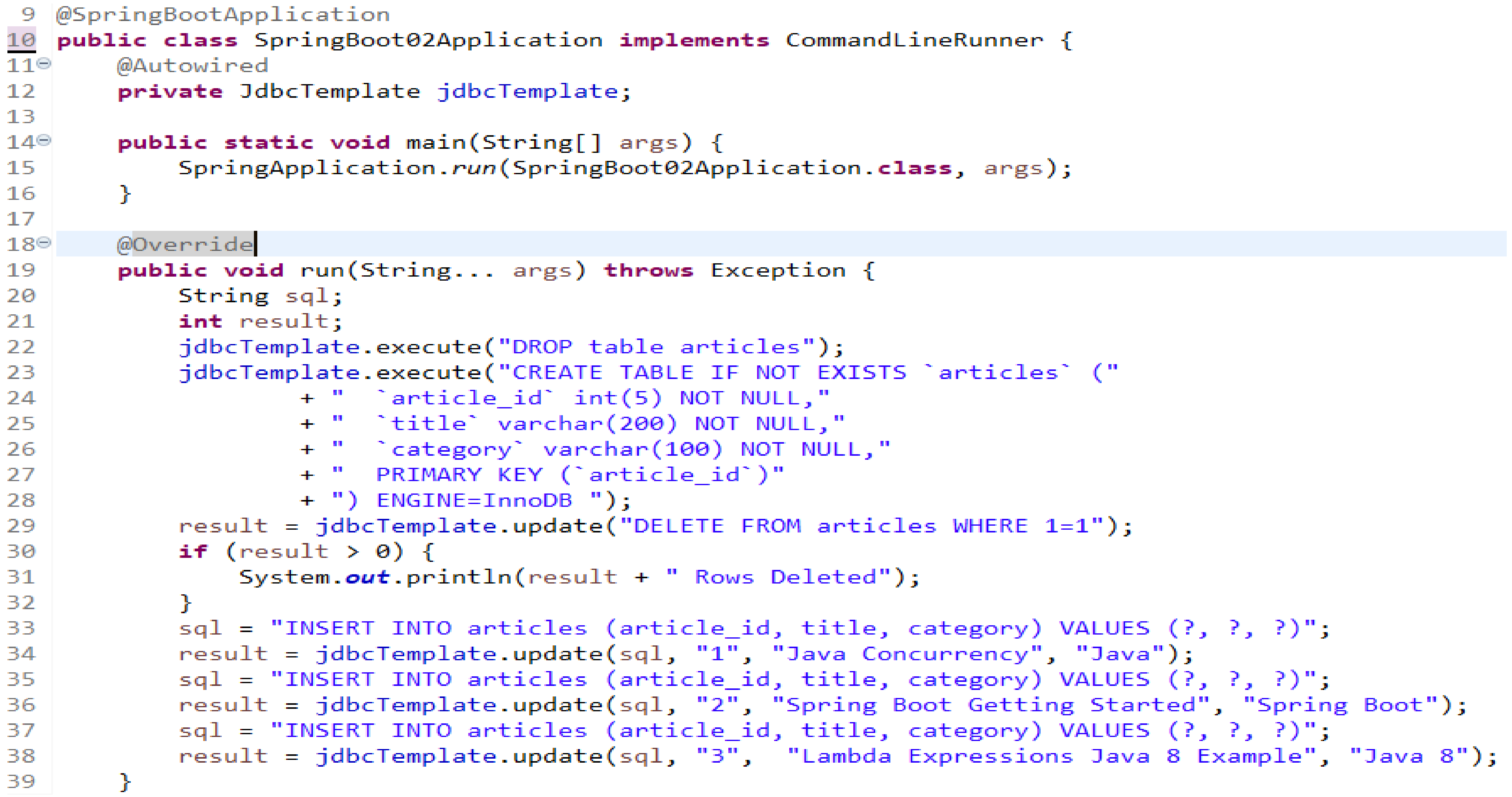
Available:

- Developer Tools
- Google Cloud Platform
- I/O

Selected:

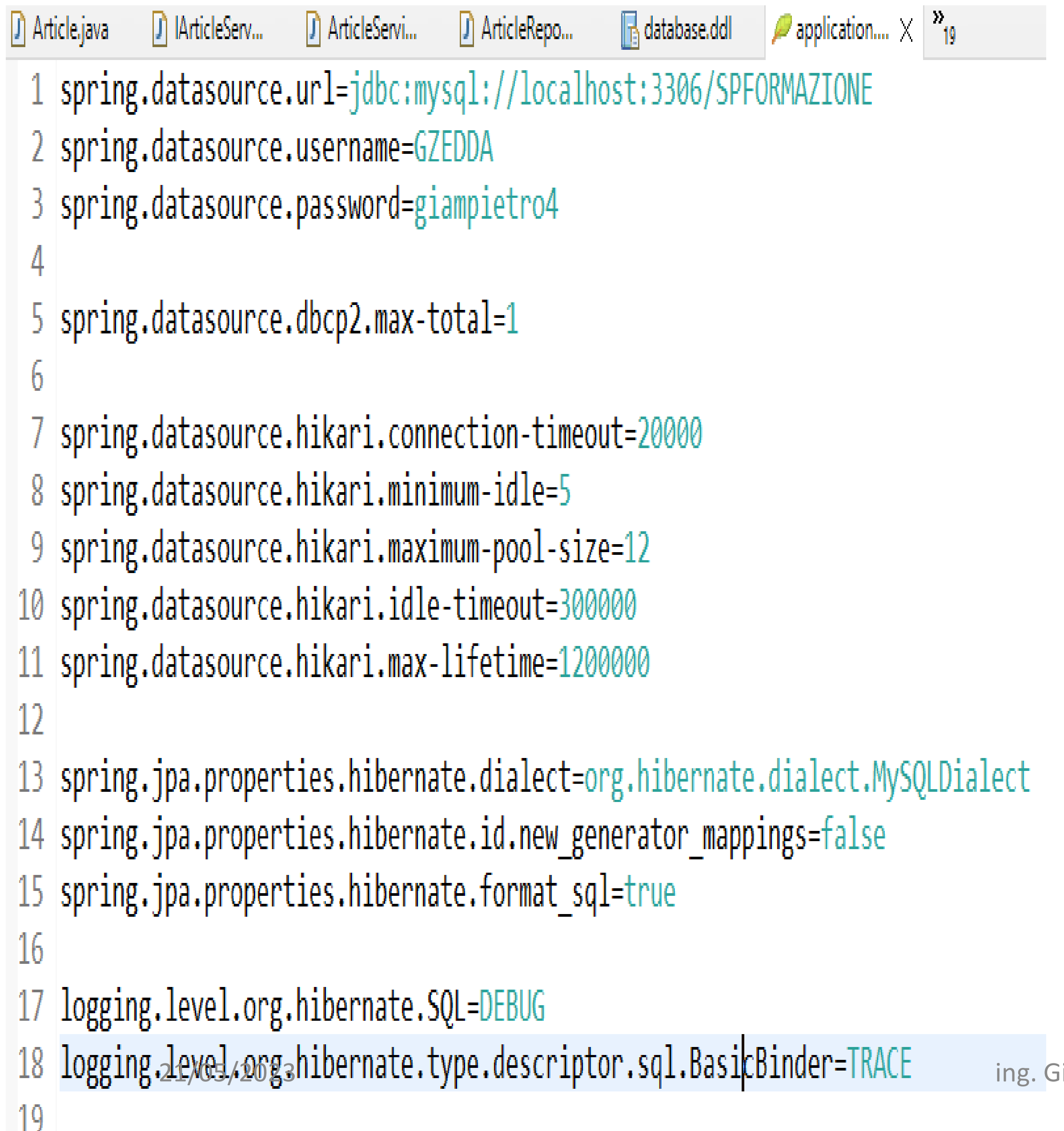
- Spring Boot DevTools
- Spring Data JPA
- MySQL Driver
- Spring Web

Spring Boot DATA JPA Application



```java
@SpringBootApplication
public class SpringBoot02Application implements CommandLineRunner {
    @Autowired
    private JdbcTemplate jdbcTemplate;

    public static void main(String[] args) {
        SpringApplication.run(SpringBoot02Application.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        String sql;
        int result;
        jdbcTemplate.execute("DROP table articles");
        jdbcTemplate.execute("CREATE TABLE IF NOT EXISTS `articles` ("
                + "  `article_id` int(5) NOT NULL,"
                + "  `title` varchar(200) NOT NULL,"
                + "  `category` varchar(100) NOT NULL,"
                + "  PRIMARY KEY (`article_id`)"
                + ") ENGINE=InnoDB ");
        result = jdbcTemplate.update("DELETE FROM articles WHERE 1=1");
        if (result > 0) {
            System.out.println(result + " Rows Deleted");
        }
        sql = "INSERT INTO articles (article_id, title, category) VALUES (?, ?, ?)";
        result = jdbcTemplate.update(sql, "1", "Java Concurrency", "Java");
        sql = "INSERT INTO articles (article_id, title, category) VALUES (?, ?, ?)";
        result = jdbcTemplate.update(sql, "2", "Spring Boot Getting Started", "Spring Boot");
        sql = "INSERT INTO articles (article_id, title, category) VALUES (?, ?, ?)";
        result = jdbcTemplate.update(sql, "3",  "Lambda Expressions Java 8 Example", "Java 8");
    }
```

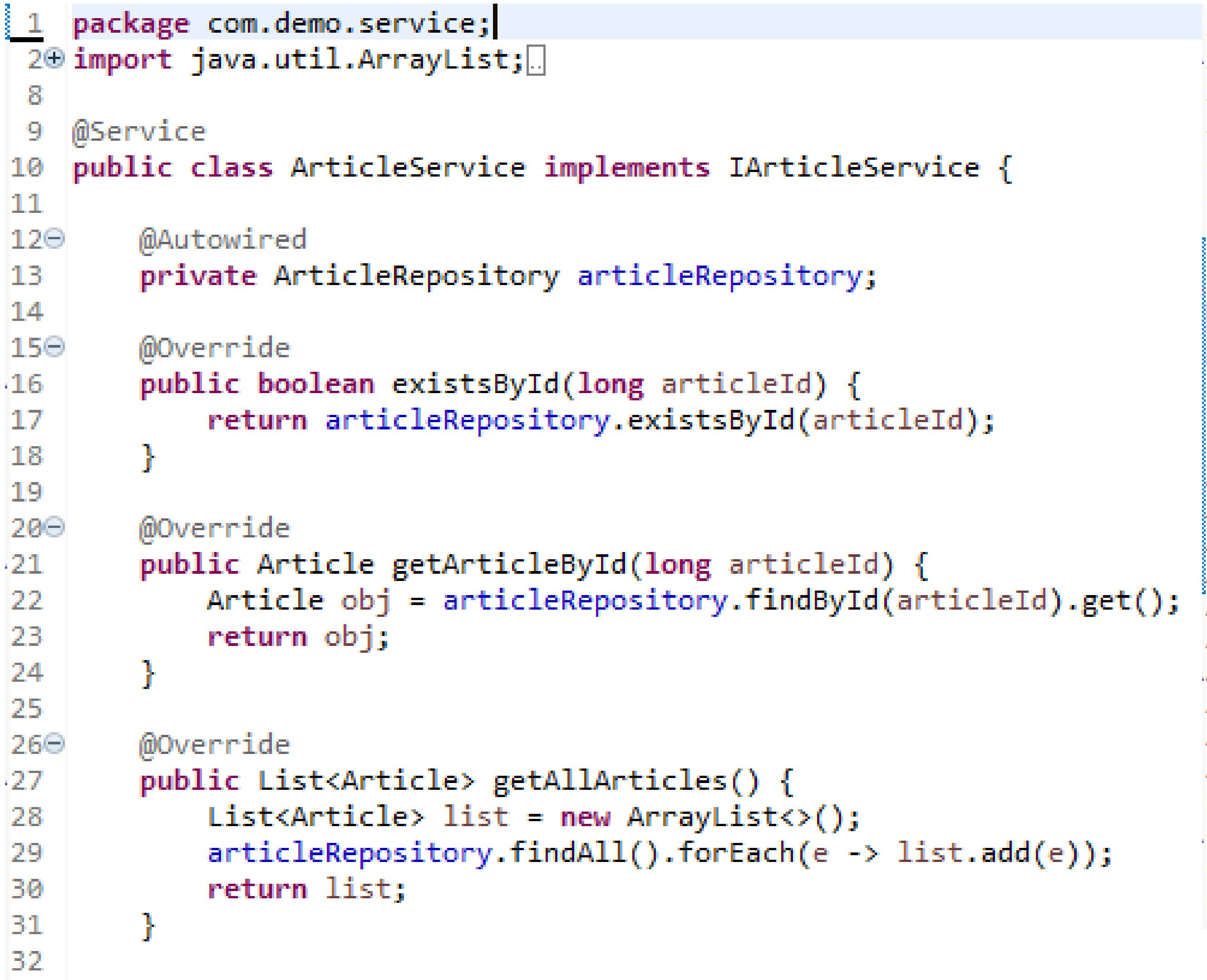
Spring Boot Eclipse Property File



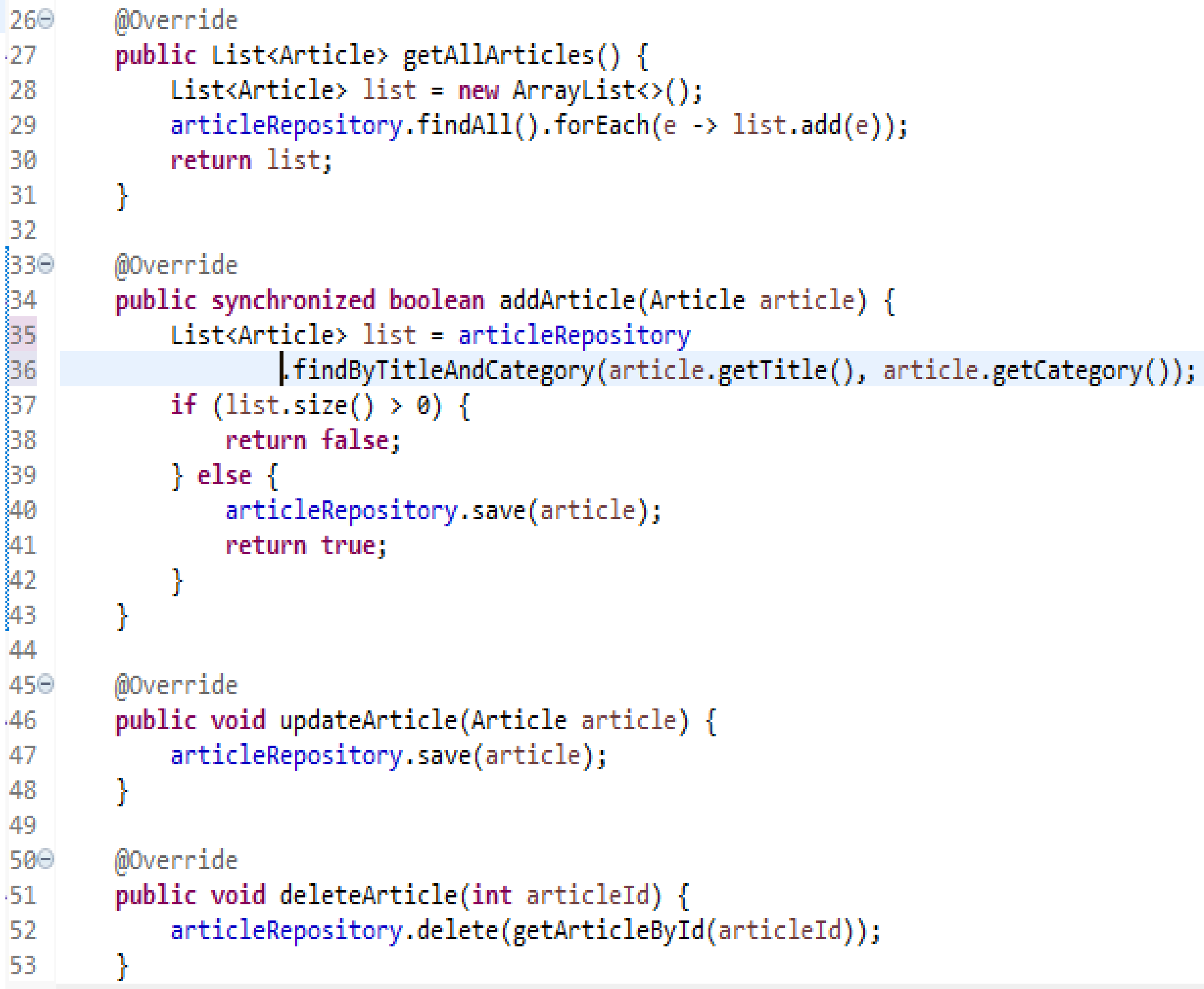
```
spring.datasource.url=jdbc:mysql://localhost:3306/SPFORMAZIONE
spring.datasource.username=GZEDDA
spring.datasource.password=giampietro4

spring.datasource.dbcp2.max-total=1

spring.datasource.hikari.connection-timeout=20000
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.maximum-pool-size=12
spring.datasource.hikari.idle-timeout=300000
spring.datasource.hikari.max-lifetime=1200000

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.properties.hibernate.id.new_generator_mappings=false
spring.jpa.properties.hibernate.format_sql=true

logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
```

- Da Spring Boot 2.0, la tecnologia di pooling del database predefinita è stata spostata da Tomcat Pool a **HikariCP.** *spring-boot-starter-jdbc* e *spring-boot-starter-data-jpa* risolvono la dipendenza HikariCP per impostazione predefinita e la proprietà
- **spring.datasource.type** ha HikariDataSource come valore predefinito. Le proprietà dell'origine dati che iniziano con spring.datasource.* verranno lette automaticamente da Spring boot JPA.
- Per modificare le proprietà di Hibernate useremo il prefisso **spring.jpa.properties.*** con i nomi delle proprietà di **Hibernate**.
- Sulla base dell'URL dell'origine dati specificato, Spring Boot può identificare automaticamente la classe del driver dell'origine dati senza necessità di configurazione.

Spring Boot DATA JPA CRUD Service



```java
package com.demo.service;
import java.util.ArrayList;

@Service
public class ArticleService implements IArticleService {

    @Autowired
    private ArticleRepository articleRepository;

    @Override
    public boolean existsById(long articleId) {
        return articleRepository.existsById(articleId);
    }

    @Override
    public Article getArticleById(long articleId) {
        Article obj = articleRepository.findById(articleId).get();
        return obj;
    }

    @Override
    public List<Article> getAllArticles() {
        List<Article> list = new ArrayList<>();
        articleRepository.findAll().forEach(e -> list.add(e));
        return list;
    }

```



```java
    @Override
    public List<Article> getAllArticles() {
        List<Article> list = new ArrayList<>();
        articleRepository.findAll().forEach(e -> list.add(e));
        return list;
    }

```

```java
    @Override
    public synchronized boolean addArticle(Article article) {
        List<Article> list = articleRepository
                .findByTitleAndCategory(article.getTitle(), article.getCategory());
        if (list.size() > 0) {
            return false;
        } else {
            articleRepository.save(article);
            return true;
        }
    }

    @Override
    public void updateArticle(Article article) {
        articleRepository.save(article);
    }

    @Override
    public void deleteArticle(int articleId) {
        articleRepository.delete(getArticleById(articleId));
    }
```

Spring Boot REST CRUD JPA

❑Viene creata una applicazione RESTful web service. Si eseguiranno operazioni **CRUD** sulla tabella **articles**.

❑Alla partenza dell'applicazione Spring Boot, via **`CommandLineRunner`** interface e **`JdbcTemplate`** si effettuerà la creazione della tabella e il suo caricamento

❑Verranno forniti nel controller **ArticleController** endpoint **Spring REST**, per create, read, update e delete di articles.

Spring Boot REST CRUD JPA

- ❑Il controller ArticleController smisterà le richieste di **GET, PUT, POST e DELETE** alla classe service **ArticleService**

- ❑ArticleService attraverso la classe **ArticleRepository** effettuera l’accesso al database sfruttando le features di Spring Data Jpa.

- ❑**ArticleRepository** eredita da CRUDRepository le operazioni di persistenza e definisce le sue query personalizzate

Spring Boot REST CRUD JPA

- Il controller ArticleController smisterà specifiche richieste di **GET** per

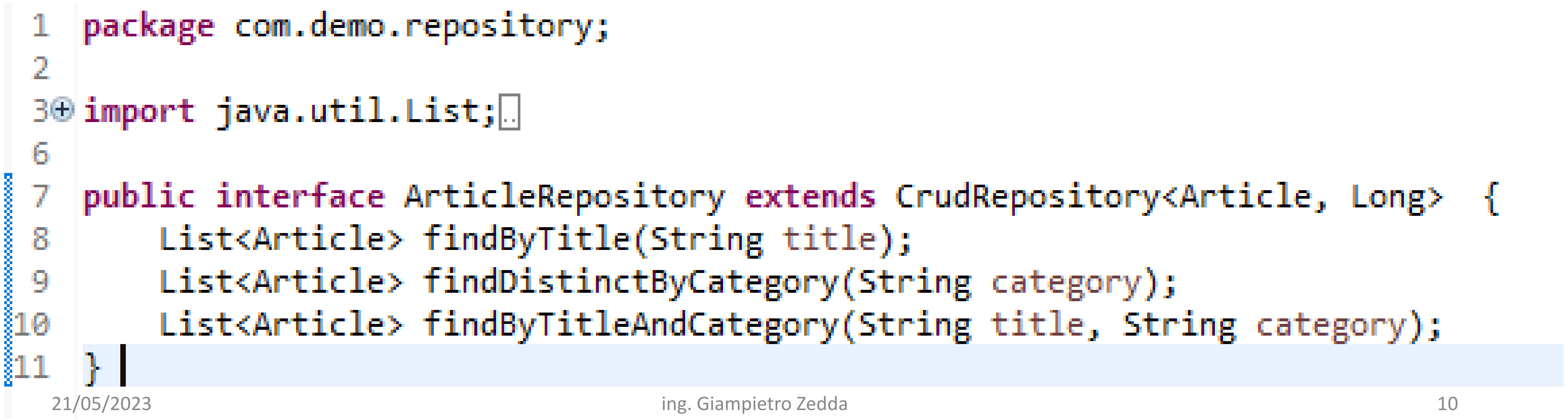
  - restituire I dati dell’articolo in diversi formati
    - **JSON**
    - **XML** con diverse modalità di implementazione
    - **TEXT** come da toString() della classe Article

  - Visualizzare questo **PDF** nel browser

  - Effettuare il **download** di questo PDF

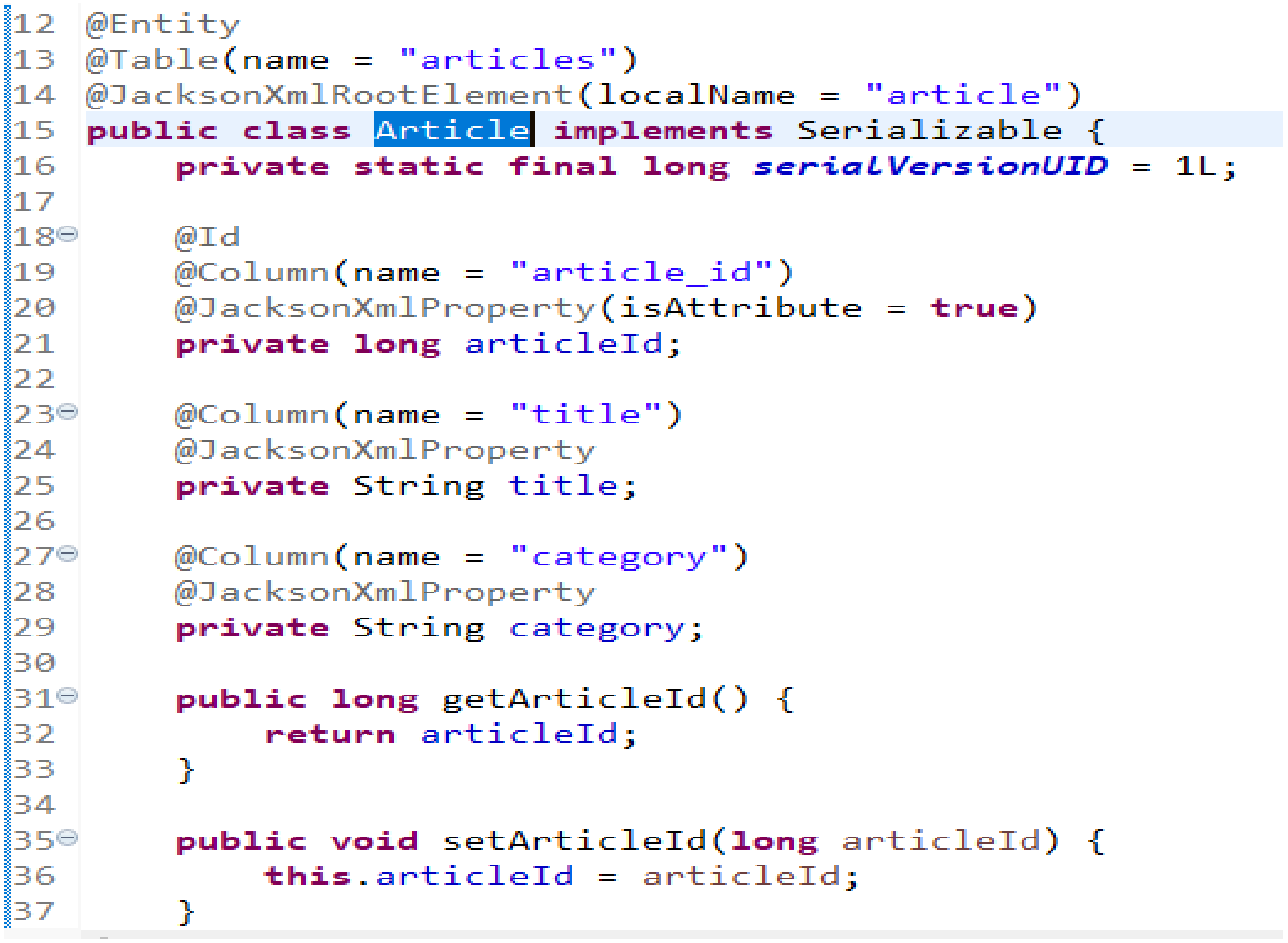
Spring Boot DATA JPA Repository

- Spring Data definisce una serie di interface, chiamate **Repository**, attraverso cui, senza scrivere codice, si effettuano tutte le operazioni di accesso e di persistenza al database
  - **CrudRepository** per le operazioni di gestione CRUD
  - **PagingAndSortingRepository** per gestire la paginazione
  - **JpaRepository/MongoRepository** per specifiche tecnologie di persistenza



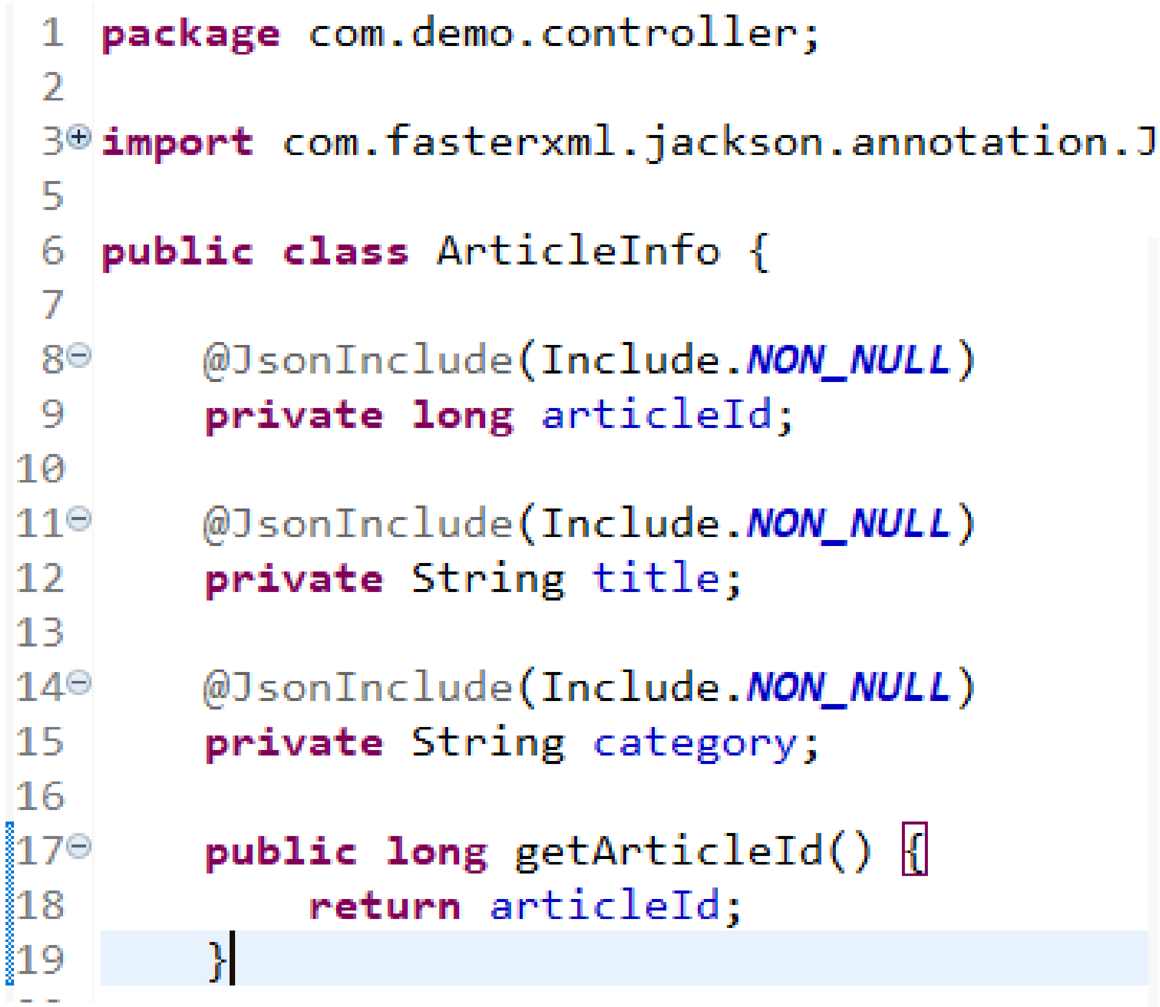
```java
package com.demo.repository;

import java.util.List;...

public interface ArticleRepository extends CrudRepository<Article, Long>  {
    List<Article> findByTitle(String title);
    List<Article> findDistinctByCategory(String category);
    List<Article> findByTitleAndCategory(String title, String category);
}
```

Spring Boot DATA JPA Entity & Xml Model

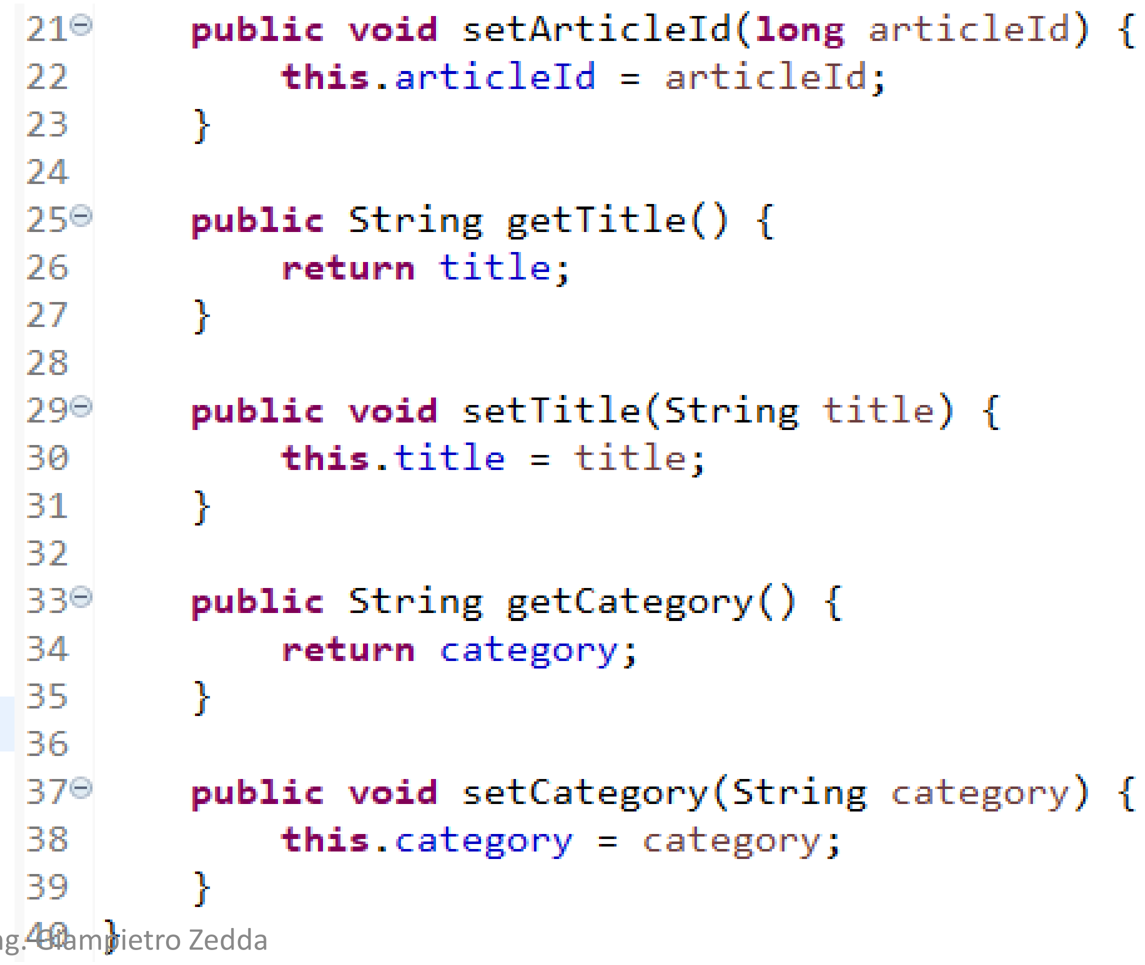


```
@Entity
@Table(name = "articles")
@JacksonXmlRootElement(localName = "article")
public class Article implements Serializable {
	private static final long serialVersionUID = 1L;

	@Id
	@Column(name = "article_id")
	@JacksonXmlProperty(isAttribute = true)
	private long articleId;

	@Column(name = "title")
	@JacksonXmlProperty
	private String title;

	@Column(name = "category")
	@JacksonXmlProperty
	private String category;

	public long getArticleId() {
		return articleId;
	}

	public void setArticleId(long articleId) {
		this.articleId = articleId;
	}
```

Spring Boot DATA JPA Json Mapping

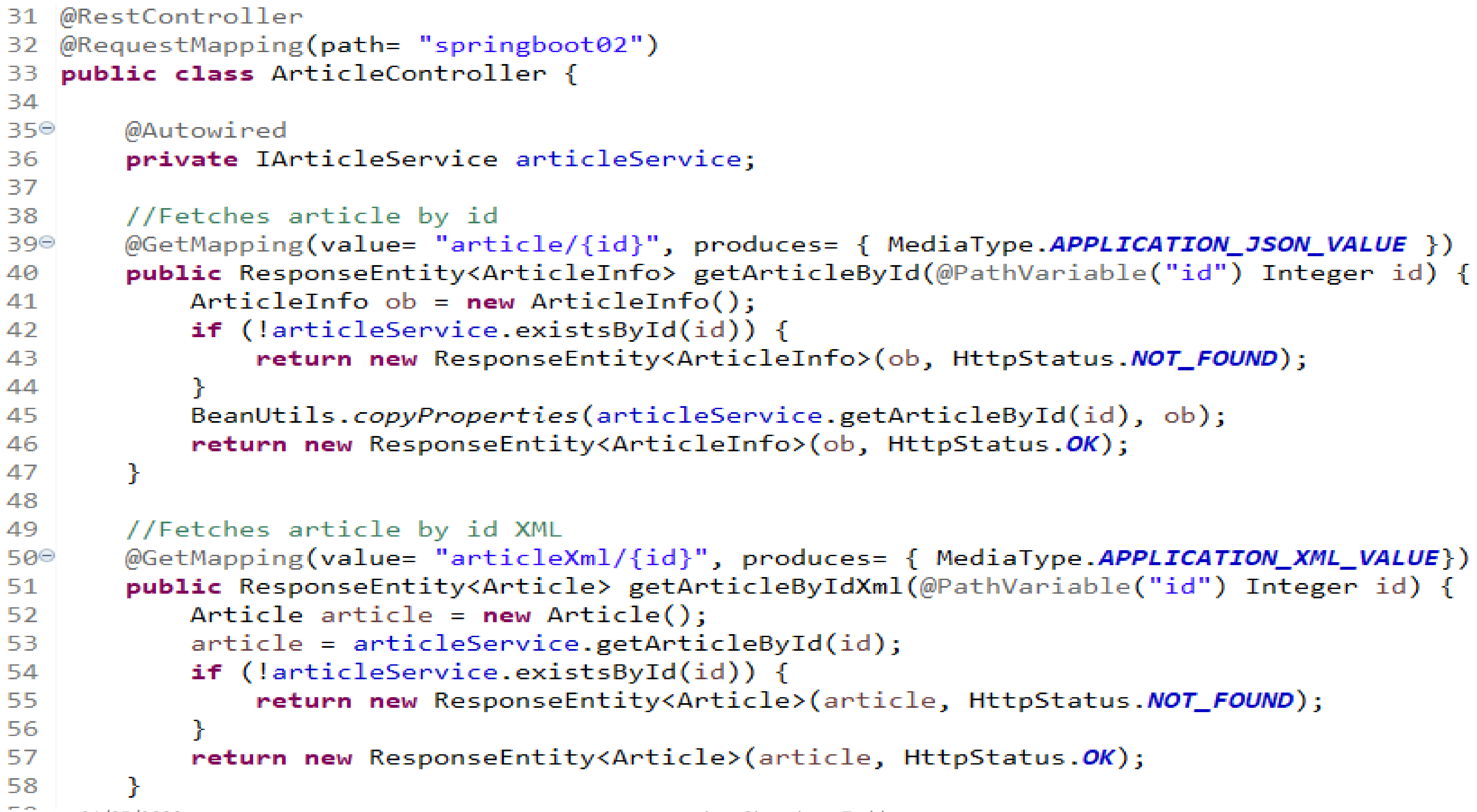


```java
package com.demo.controller;

import com.fasterxml.jackson.annotation.JsonInclude;

public class ArticleInfo {

    @JsonInclude(Include.NON_NULL)
    private long articleId;

    @JsonInclude(Include.NON_NULL)
    private String title;

    @JsonInclude(Include.NON_NULL)
    private String category;

    public long getArticleId() {
        return articleId;
    }
```



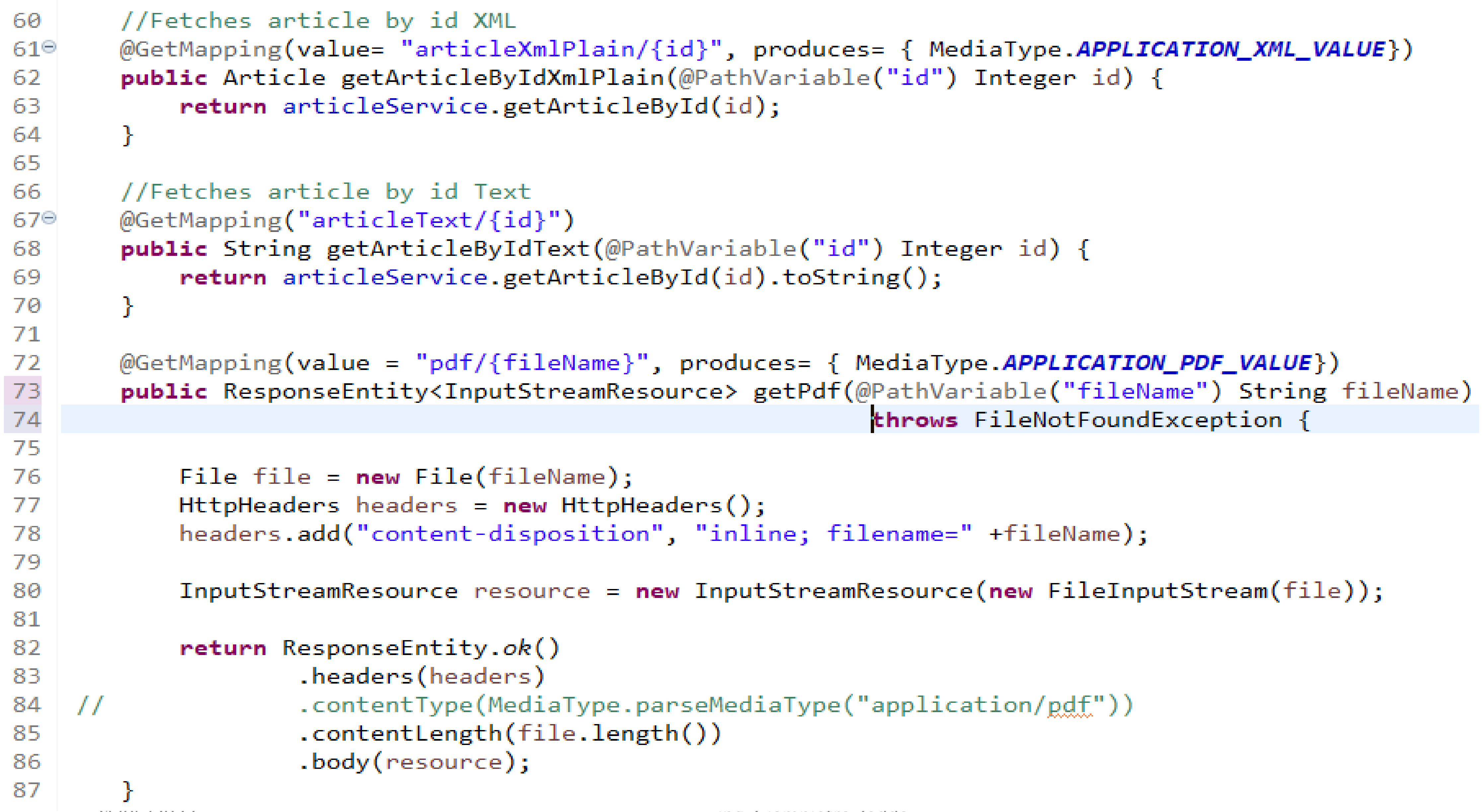
```java
    public void setArticleId(long articleId) {
        this.articleId = articleId;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }
}
```

Spring Boot DATA JPA Controller



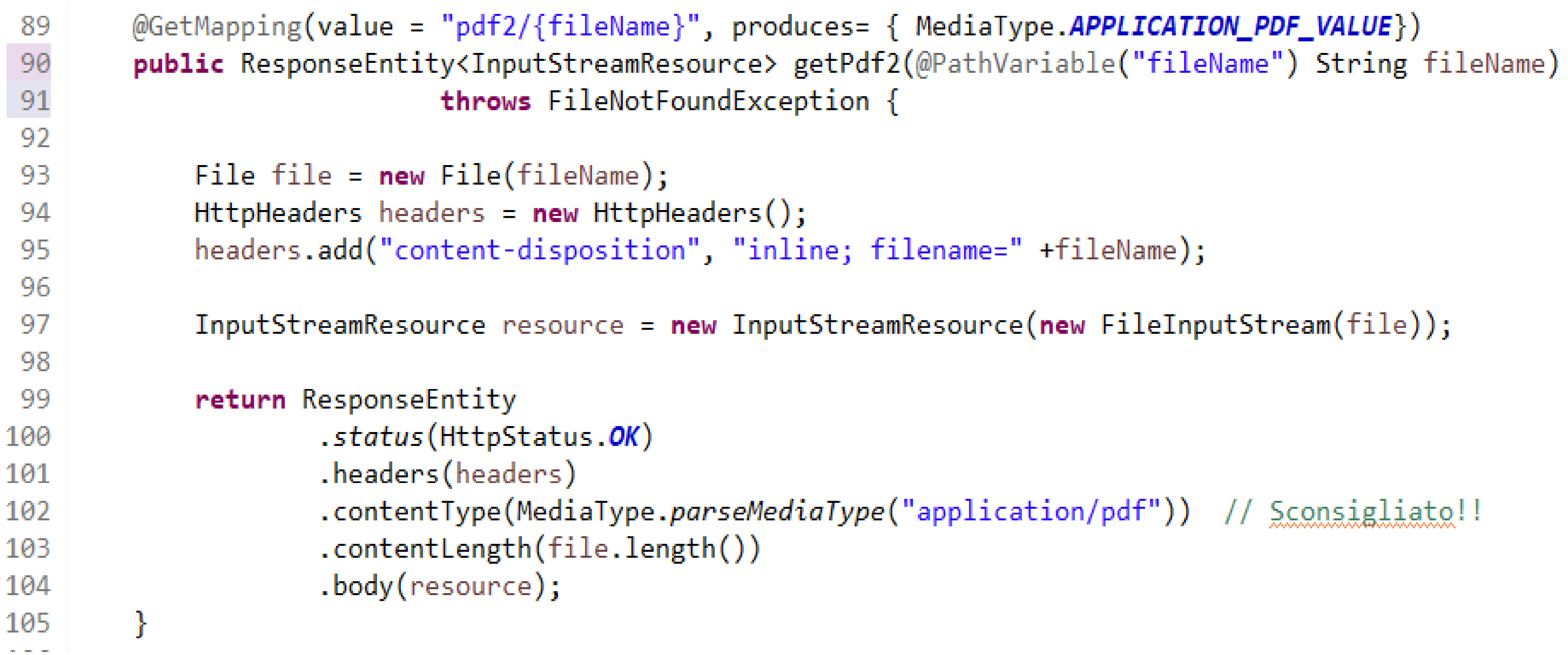
```java
@RestController
@RequestMapping(path= "springboot02")
public class ArticleController {

    @Autowired
    private IArticleService articleService;

    //Fetches article by id
    @GetMapping(value= "article/{id}", produces= { MediaType.APPLICATION_JSON_VALUE })
    public ResponseEntity<ArticleInfo> getArticleById(@PathVariable("id") Integer id) {
        ArticleInfo ob = new ArticleInfo();
        if (!articleService.existsById(id)) {
            return new ResponseEntity<ArticleInfo>(ob, HttpStatus.NOT_FOUND);
        }
        BeanUtils.copyProperties(articleService.getArticleById(id), ob);
        return new ResponseEntity<ArticleInfo>(ob, HttpStatus.OK);
    }

    //Fetches article by id XML
    @GetMapping(value= "articleXml/{id}", produces= { MediaType.APPLICATION_XML_VALUE})
    public ResponseEntity<Article> getArticleByIdXml(@PathVariable("id") Integer id) {
        Article article = new Article();
        article = articleService.getArticleById(id);
        if (!articleService.existsById(id)) {
            return new ResponseEntity<Article>(article, HttpStatus.NOT_FOUND);
        }
        return new ResponseEntity<Article>(article, HttpStatus.OK);
    }
```

Spring Boot DATA JPA Controller



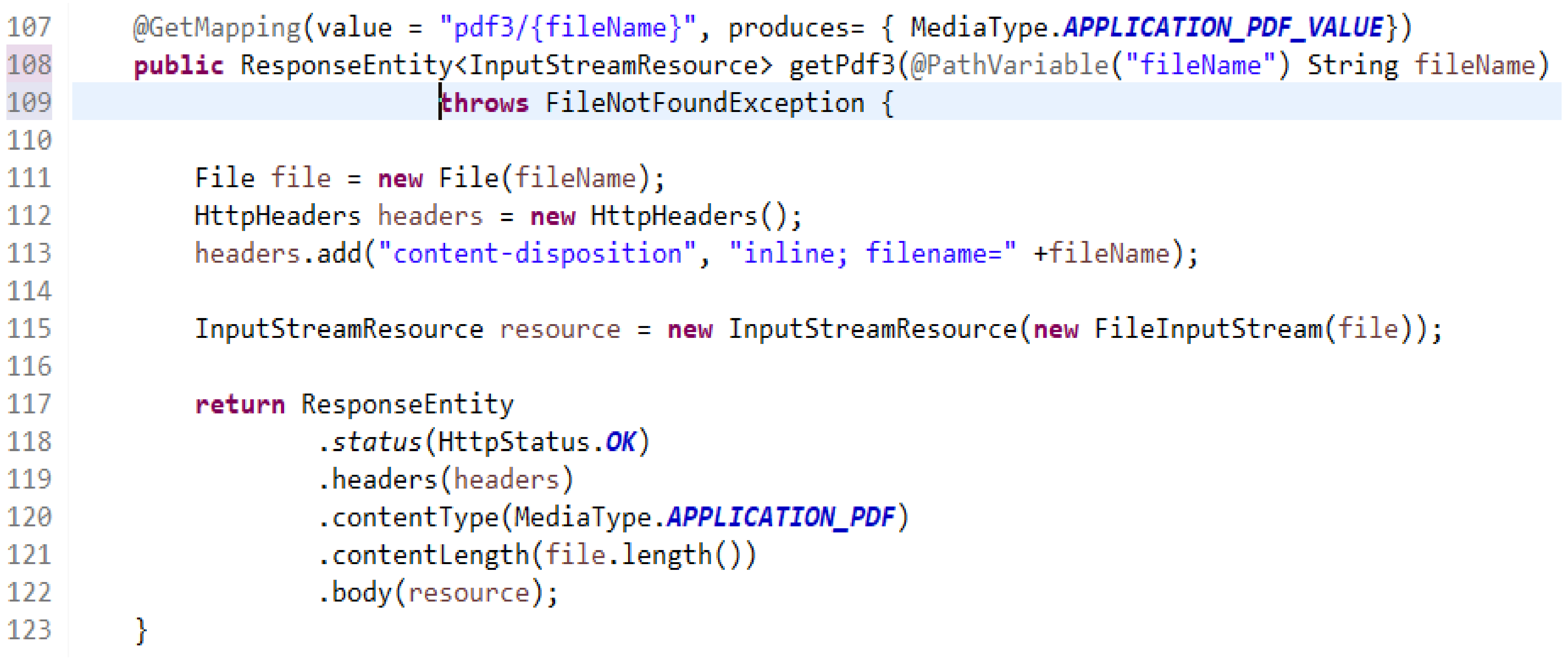
```java
    //Fetches article by id XML
    @GetMapping(value= "articleXmlPlain/{id}", produces= { MediaType.APPLICATION_XML_VALUE})
    public Article getArticleByIdXmlPlain(@PathVariable("id") Integer id) {
        return articleService.getArticleById(id);
    }

    //Fetches article by id Text
    @GetMapping("articleText/{id}")
    public String getArticleByIdText(@PathVariable("id") Integer id) {
        return articleService.getArticleById(id).toString();
    }

    @GetMapping(value = "pdf/{fileName}", produces= { MediaType.APPLICATION_PDF_VALUE})
    public ResponseEntity<InputStreamResource> getPdf(@PathVariable("fileName") String fileName)
                                                        throws FileNotFoundException {

        File file = new File(fileName);
        HttpHeaders headers = new HttpHeaders();
        headers.add("content-disposition", "inline; filename=" +fileName);

        InputStreamResource resource = new InputStreamResource(new FileInputStream(file));

        return ResponseEntity.ok()
                .headers(headers)
//              .contentType(MediaType.parseMediaType("application/pdf"))
                .contentLength(file.length())
                .body(resource);
    }
```

Spring Boot DATA JPA Controller



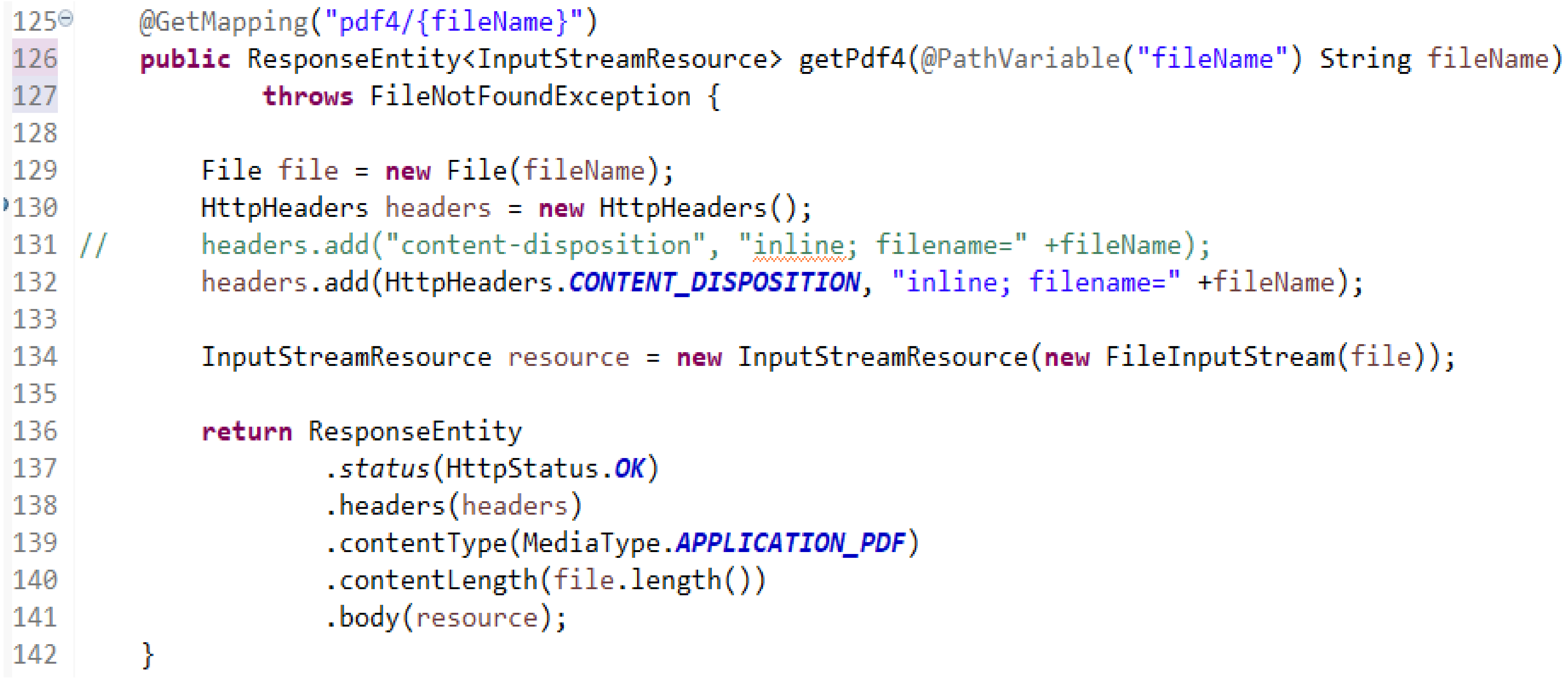
```
    @GetMapping(value = "pdf2/{fileName}", produces= { MediaType.APPLICATION_PDF_VALUE})
    public ResponseEntity<InputStreamResource> getPdf2(@PathVariable("fileName") String fileName)
                        throws FileNotFoundException {

        File file = new File(fileName);
        HttpHeaders headers = new HttpHeaders();
        headers.add("content-disposition", "inline; filename=" +fileName);

        InputStreamResource resource = new InputStreamResource(new FileInputStream(file));

        return ResponseEntity
                .status(HttpStatus.OK)
                .headers(headers)
                .contentType(MediaType.parseMediaType("application/pdf"))  // Sconsigliato!!
                .contentLength(file.length())
                .body(resource);
    }
```

Spring Boot DATA JPA Controller



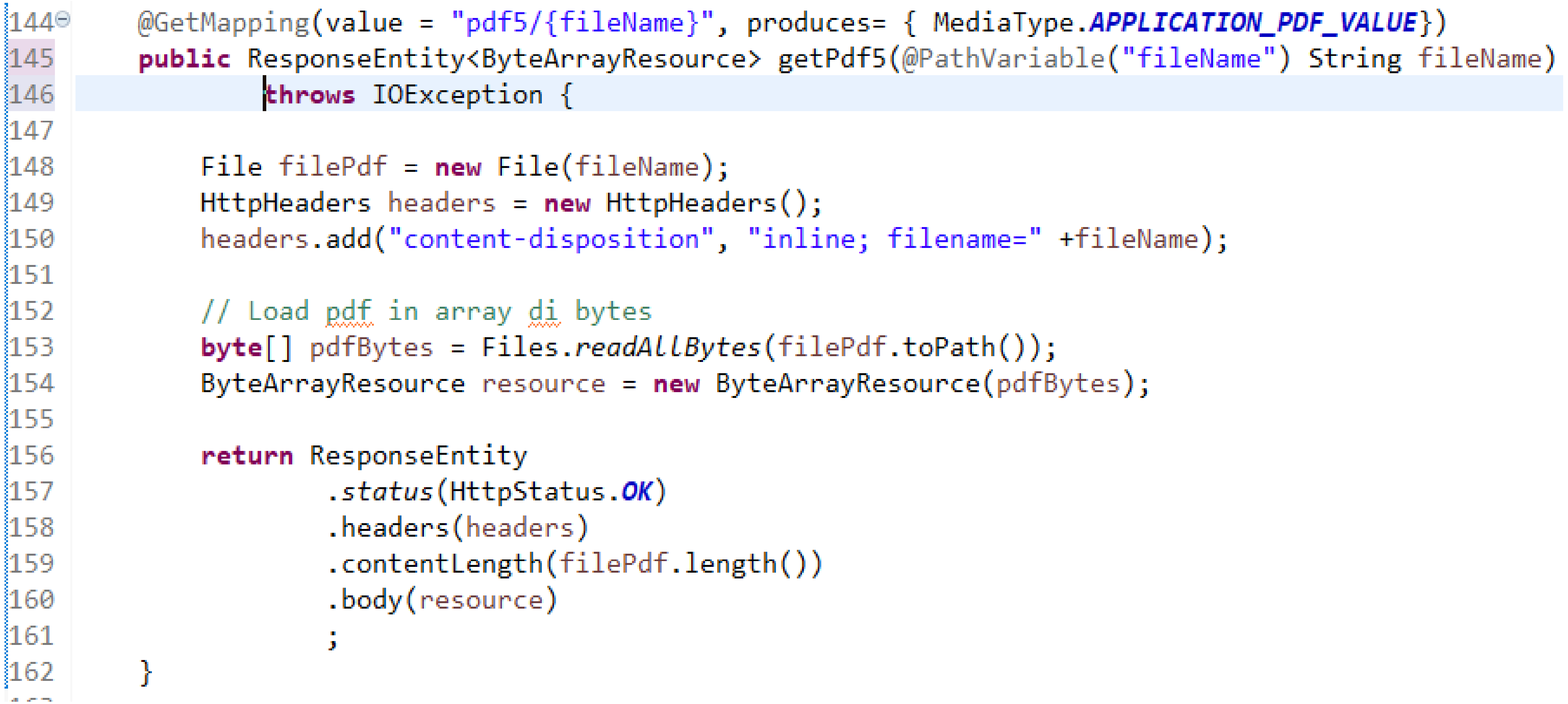
```java
    @GetMapping(value = "pdf3/{fileName}", produces= { MediaType.APPLICATION_PDF_VALUE})
    public ResponseEntity<InputStreamResource> getPdf3(@PathVariable("fileName") String fileName)
                        throws FileNotFoundException {

        File file = new File(fileName);
        HttpHeaders headers = new HttpHeaders();
        headers.add("content-disposition", "inline; filename=" +fileName);

        InputStreamResource resource = new InputStreamResource(new FileInputStream(file));

        return ResponseEntity
                .status(HttpStatus.OK)
                .headers(headers)
                .contentType(MediaType.APPLICATION_PDF)
                .contentLength(file.length())
                .body(resource);
    }
```

Spring Boot DATA JPA Controller



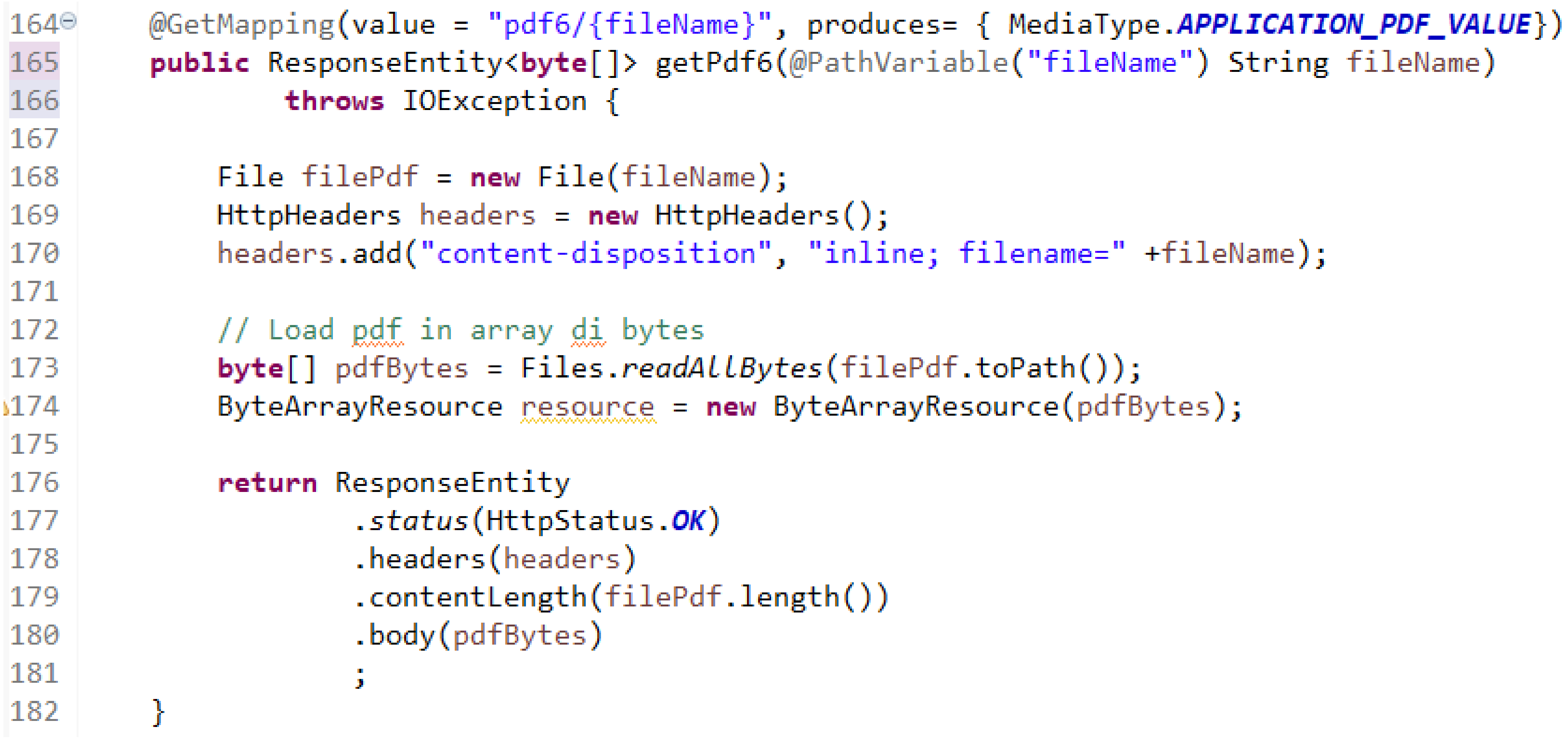
```java
    @GetMapping("pdf4/{fileName}")
    public ResponseEntity<InputStreamResource> getPdf4(@PathVariable("fileName") String fileName)
            throws FileNotFoundException {

        File file = new File(fileName);
        HttpHeaders headers = new HttpHeaders();
//      headers.add("content-disposition", "inline; filename=" +fileName);
        headers.add(HttpHeaders.CONTENT_DISPOSITION, "inline; filename=" +fileName);

        InputStreamResource resource = new InputStreamResource(new FileInputStream(file));

        return ResponseEntity
                .status(HttpStatus.OK)
                .headers(headers)
                .contentType(MediaType.APPLICATION_PDF)
                .contentLength(file.length())
                .body(resource);
    }
```

Spring Boot DATA JPA Controller



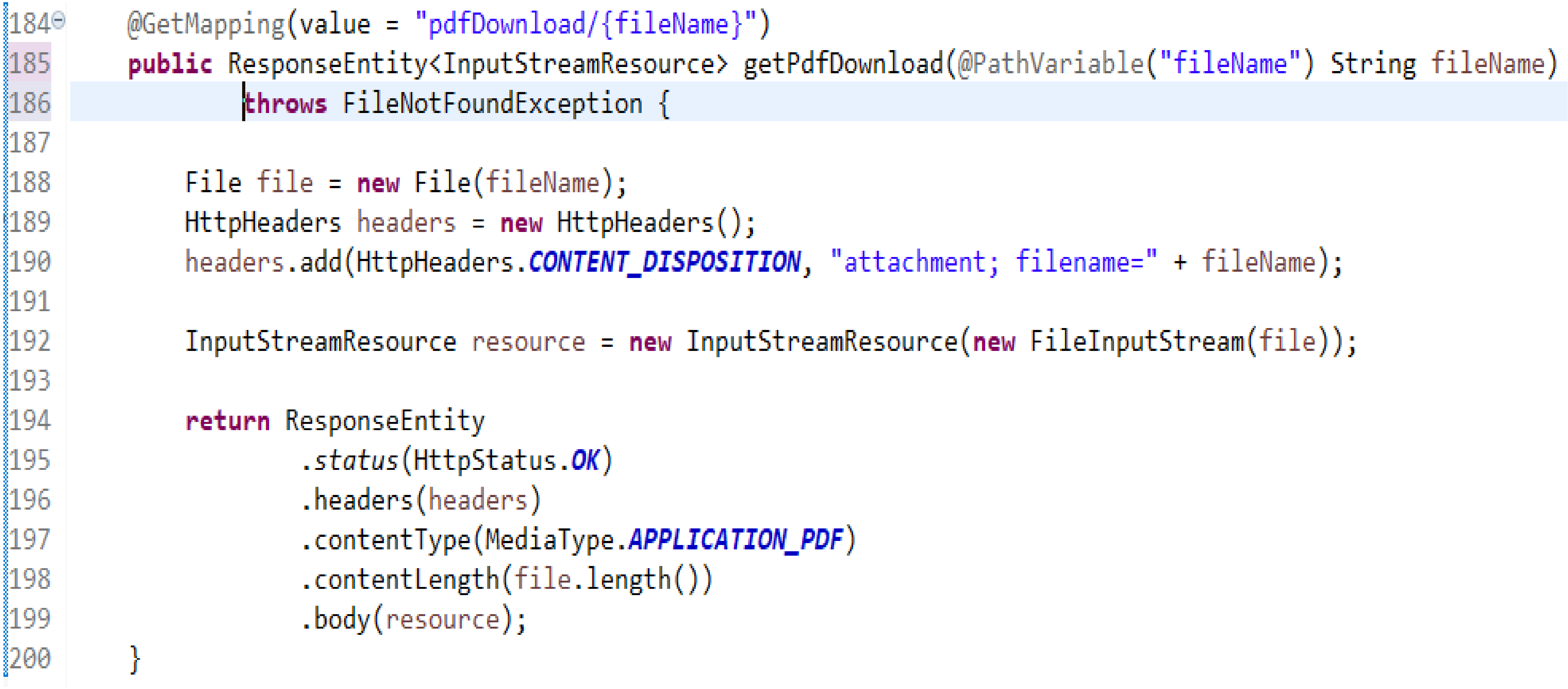
```java
    @GetMapping(value = "pdf5/{fileName}", produces= { MediaType.APPLICATION_PDF_VALUE})
    public ResponseEntity<ByteArrayResource> getPdf5(@PathVariable("fileName") String fileName)
            throws IOException {

        File filePdf = new File(fileName);
        HttpHeaders headers = new HttpHeaders();
        headers.add("content-disposition", "inline; filename=" +fileName);

        // Load pdf in array di bytes
        byte[] pdfBytes = Files.readAllBytes(filePdf.toPath());
        ByteArrayResource resource = new ByteArrayResource(pdfBytes);

        return ResponseEntity
                .status(HttpStatus.OK)
                .headers(headers)
                .contentLength(filePdf.length())
                .body(resource)
                ;
    }
```

Spring Boot DATA JPA Controller



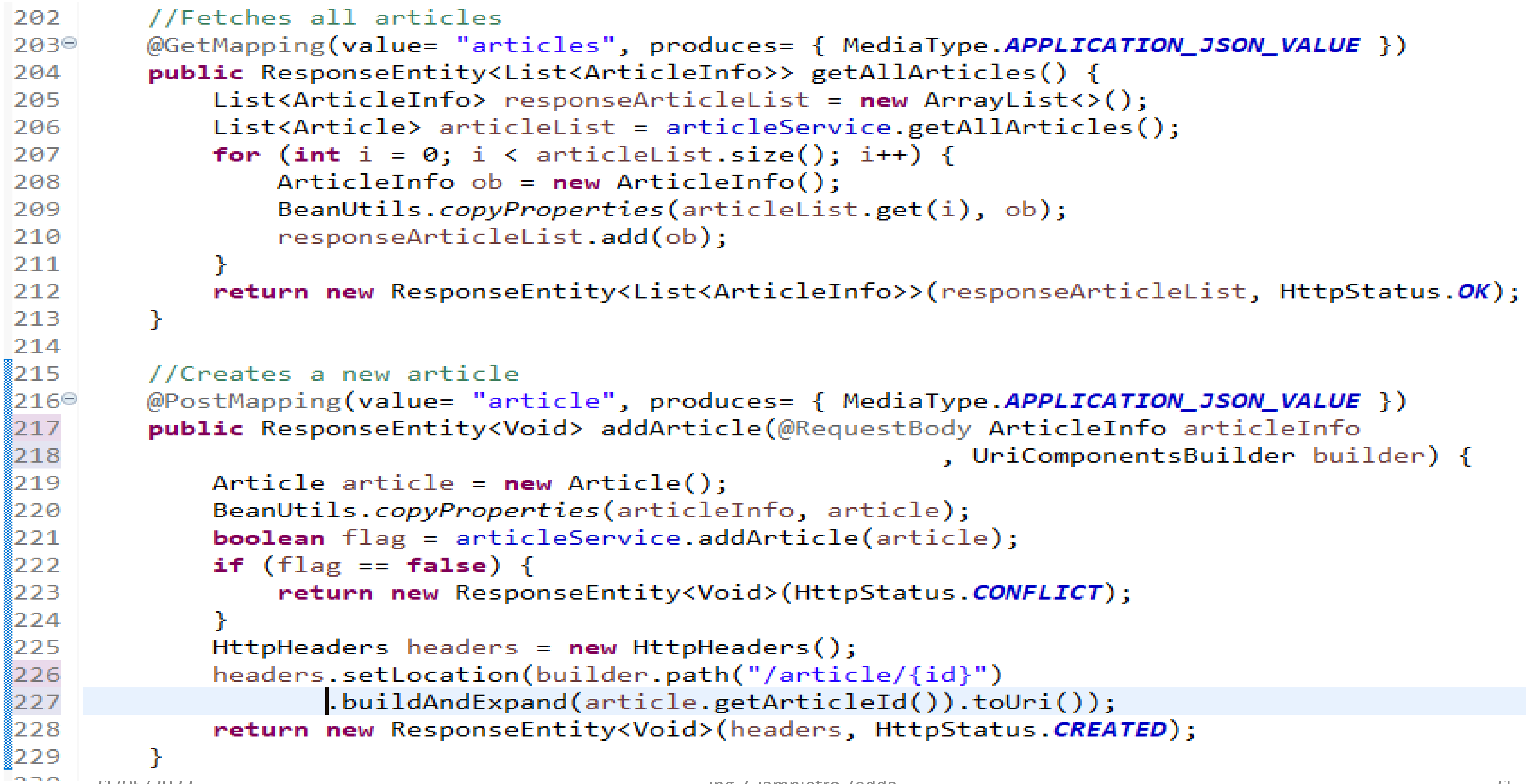
```java
    @GetMapping(value = "pdf6/{fileName}", produces= { MediaType.APPLICATION_PDF_VALUE})
    public ResponseEntity<byte[]> getPdf6(@PathVariable("fileName") String fileName)
            throws IOException {

        File filePdf = new File(fileName);
        HttpHeaders headers = new HttpHeaders();
        headers.add("content-disposition", "inline; filename=" +fileName);

        // Load pdf in array di bytes
        byte[] pdfBytes = Files.readAllBytes(filePdf.toPath());
        ByteArrayResource resource = new ByteArrayResource(pdfBytes);

        return ResponseEntity
                .status(HttpStatus.OK)
                .headers(headers)
                .contentLength(filePdf.length())
                .body(pdfBytes)
                ;
    }
```

Spring Boot DATA JPA Controller



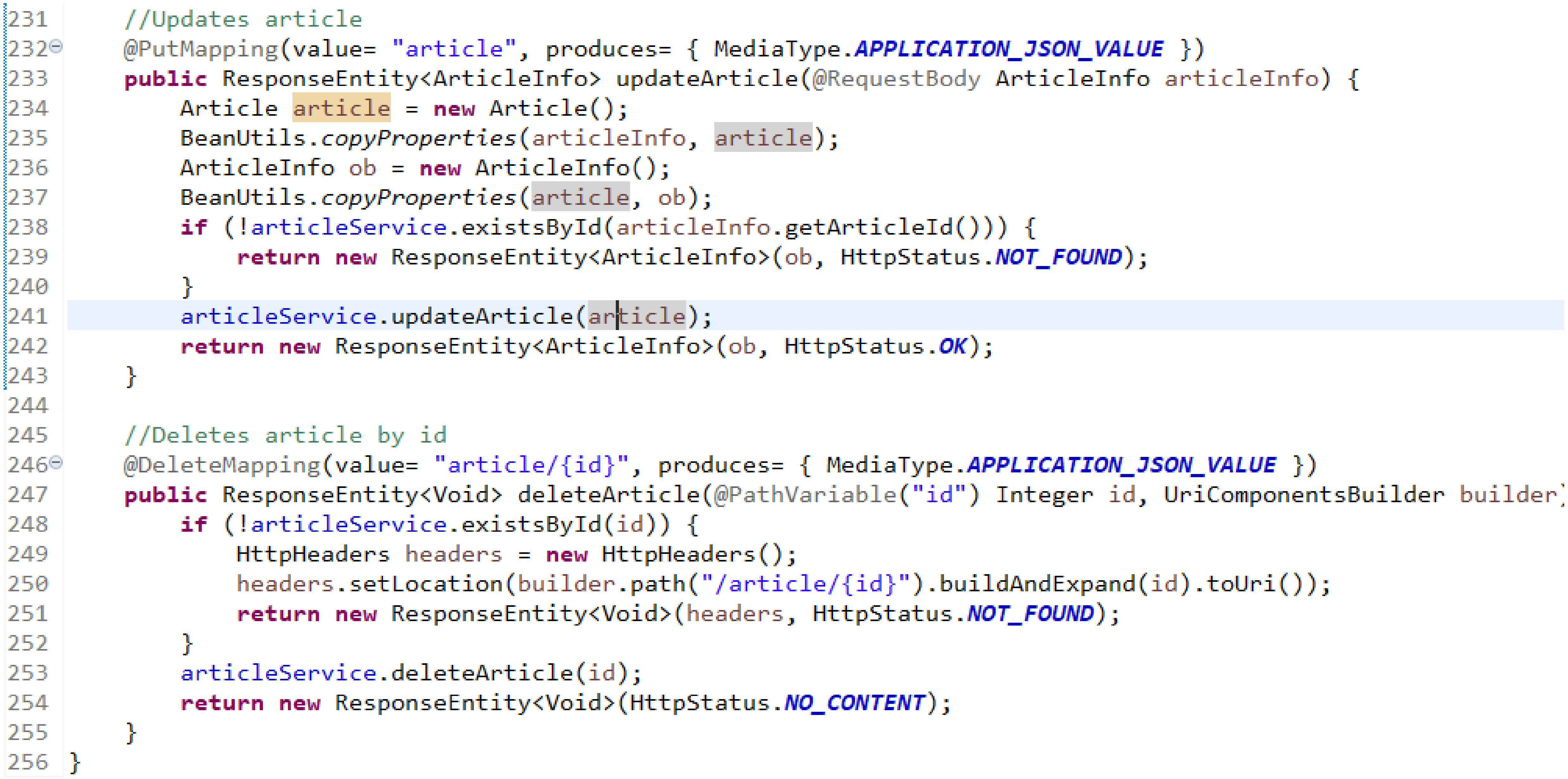
```java
    @GetMapping(value = "pdfDownload/{fileName}")
    public ResponseEntity<InputStreamResource> getPdfDownload(@PathVariable("fileName") String fileName)
            throws FileNotFoundException {

        File file = new File(fileName);
        HttpHeaders headers = new HttpHeaders();
        headers.add(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename=" + fileName);

        InputStreamResource resource = new InputStreamResource(new FileInputStream(file));

        return ResponseEntity
                .status(HttpStatus.OK)
                .headers(headers)
                .contentType(MediaType.APPLICATION_PDF)
                .contentLength(file.length())
                .body(resource);
    }
```

Spring Boot DATA JPA Controller



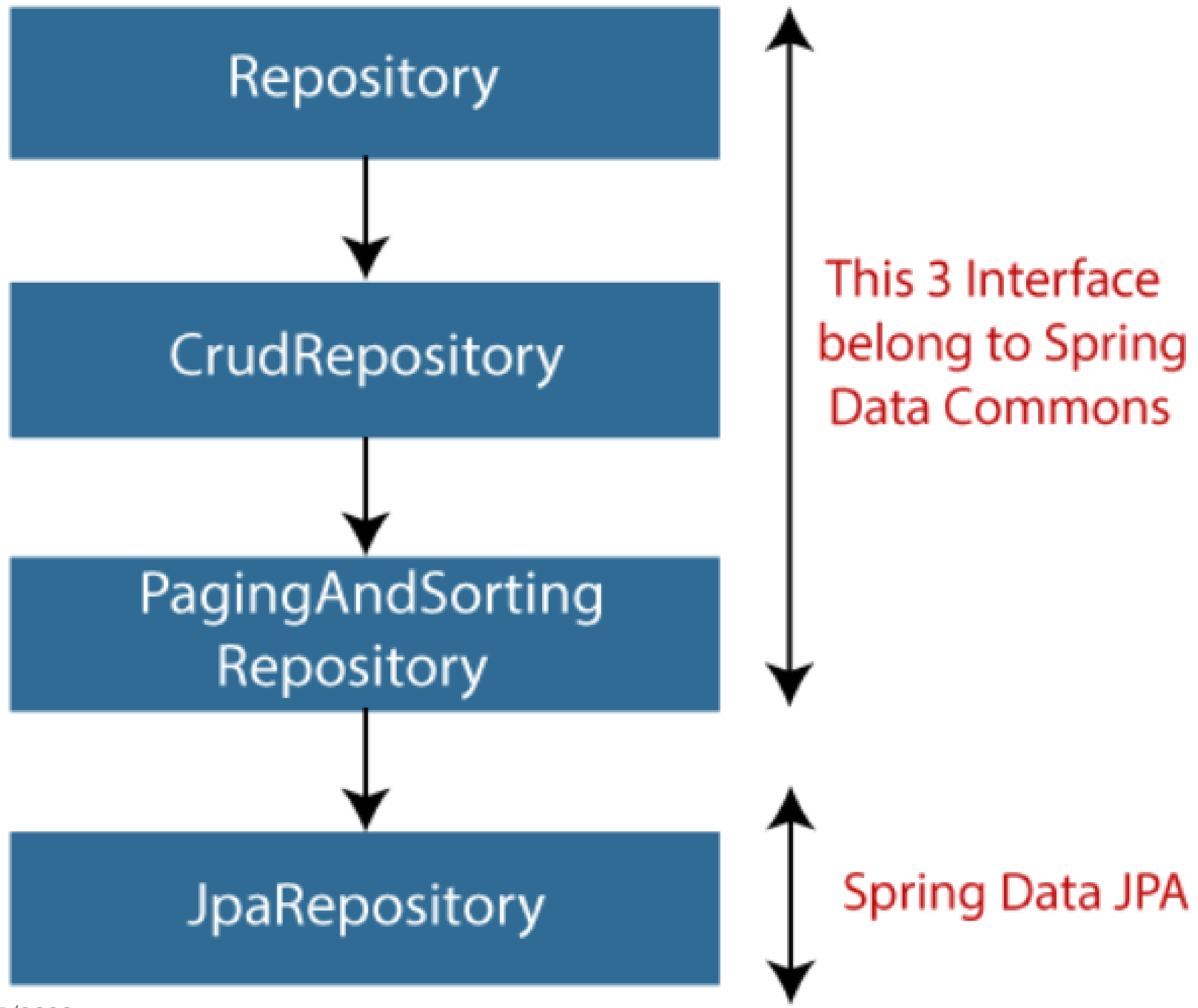
```java
    //Fetches all articles
    @GetMapping(value= "articles", produces= { MediaType.APPLICATION_JSON_VALUE })
    public ResponseEntity<List<ArticleInfo>> getAllArticles() {
        List<ArticleInfo> responseArticleList = new ArrayList<>();
        List<Article> articleList = articleService.getAllArticles();
        for (int i = 0; i < articleList.size(); i++) {
            ArticleInfo ob = new ArticleInfo();
            BeanUtils.copyProperties(articleList.get(i), ob);
            responseArticleList.add(ob);
        }
        return new ResponseEntity<List<ArticleInfo>>(responseArticleList, HttpStatus.OK);
    }

    //Creates a new article
    @PostMapping(value= "article", produces= { MediaType.APPLICATION_JSON_VALUE })
    public ResponseEntity<Void> addArticle(@RequestBody ArticleInfo articleInfo
                                            , UriComponentsBuilder builder) {
        Article article = new Article();
        BeanUtils.copyProperties(articleInfo, article);
        boolean flag = articleService.addArticle(article);
        if (flag == false) {
            return new ResponseEntity<Void>(HttpStatus.CONFLICT);
        }
        HttpHeaders headers = new HttpHeaders();
        headers.setLocation(builder.path("/article/{id}")
                .buildAndExpand(article.getArticleId()).toUri());
        return new ResponseEntity<Void>(headers, HttpStatus.CREATED);
    }
```

Spring Boot DATA JPA Controller

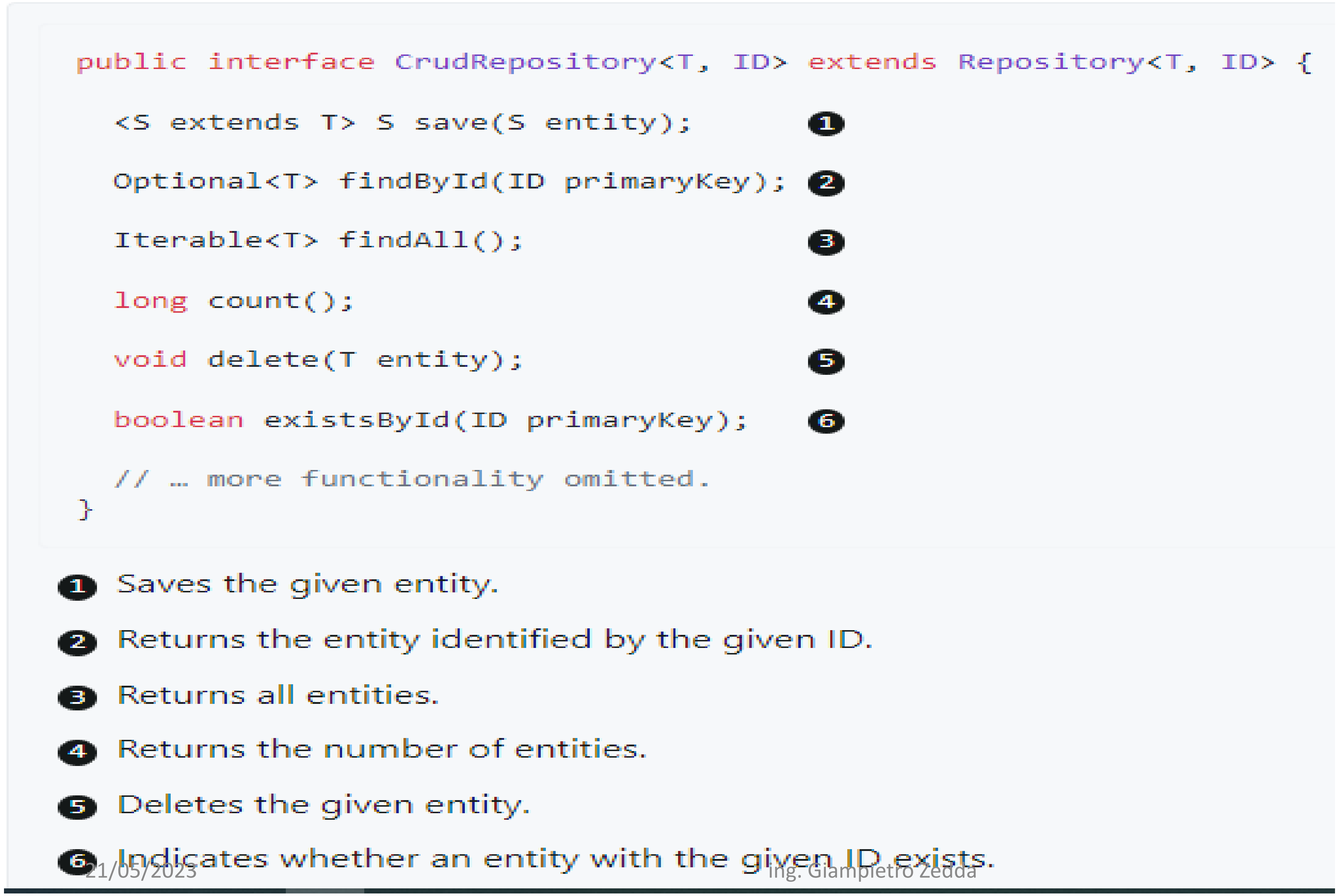


```java
    //Updates article
    @PutMapping(value= "article", produces= { MediaType.APPLICATION_JSON_VALUE })
    public ResponseEntity<ArticleInfo> updateArticle(@RequestBody ArticleInfo articleInfo) {
        Article article = new Article();
        BeanUtils.copyProperties(articleInfo, article);
        ArticleInfo ob = new ArticleInfo();
        BeanUtils.copyProperties(article, ob);
        if (!articleService.existsById(articleInfo.getArticleId())) {
            return new ResponseEntity<ArticleInfo>(ob, HttpStatus.NOT_FOUND);
        }
        articleService.updateArticle(article);
        return new ResponseEntity<ArticleInfo>(ob, HttpStatus.OK);
    }

    //Deletes article by id
    @DeleteMapping(value= "article/{id}", produces= { MediaType.APPLICATION_JSON_VALUE })
    public ResponseEntity<Void> deleteArticle(@PathVariable("id") Integer id, UriComponentsBuilder builder
        if (!articleService.existsById(id)) {
            HttpHeaders headers = new HttpHeaders();
            headers.setLocation(builder.path("/article/{id}").buildAndExpand(id).toUri());
            return new ResponseEntity<Void>(headers, HttpStatus.NOT_FOUND);
        }
        articleService.deleteArticle(id);
        return new ResponseEntity<Void>(HttpStatus.NO_CONTENT);
    }
}
```

Spring Boot DATA JPA Repository Interface


Repository
CrudRepository
PagingAndSorting Repository
JpaRepository
This 3 Interface belong to Spring Data Commons
Spring Data JPA

Spring Boot DATA JPA CRUD Repository



```
public interface CrudRepository<T, ID> extends Repository<T, ID> {

  <S extends T> S save(S entity);      ❶

  Optional<T> findById(ID primaryKey); ❷

  Iterable<T> findAll();               ❸

  long count();                        ❹

  void delete(T entity);               ❺

  boolean existsById(ID primaryKey);   ❻

  // … more functionality omitted.
}
```

❶ Saves the given entity.

❷ Returns the entity identified by the given ID.

❸ Returns all entities.

❹ Returns the number of entities.

❺ Deletes the given entity.

❻ Indicates whether an entity with the given ID exists.

Spring Boot DATA JPA CRUD Repository

Operation	SQL	HTTP verbs	RESTful Web Service
Create	INSERT	PUT/POST	POST
Read	SELECT	GET	GET
Update	UPDATE	PUT/POST/PATCH	PUT
Delete	DELETE	DELETE	DELETE

CrudRepository vs. JpaRepository

CrudRepository	JpaRepository
CrudRepository does not provide any method for pagination and sorting.	JpaRepository extends PagingAndSortingRepository. It provides all the methods for implementing the pagination.
It works as a **marker** interface.	JpaRepository extends both **CrudRepository** and **PagingAndSorting Repository**.
It provides CRUD function only. For example **findById(), findAll(),** etc.	It provides some extra methods along with the method of PagingAndSortingRepository and CrudRepository. For example, **flush(), deleteInBatch().**
It is used when we do not need the functions provided by JpaRepository and PagingAndSortingRepository.	It is used when we want to implement pagination and sorting functionality in an application.

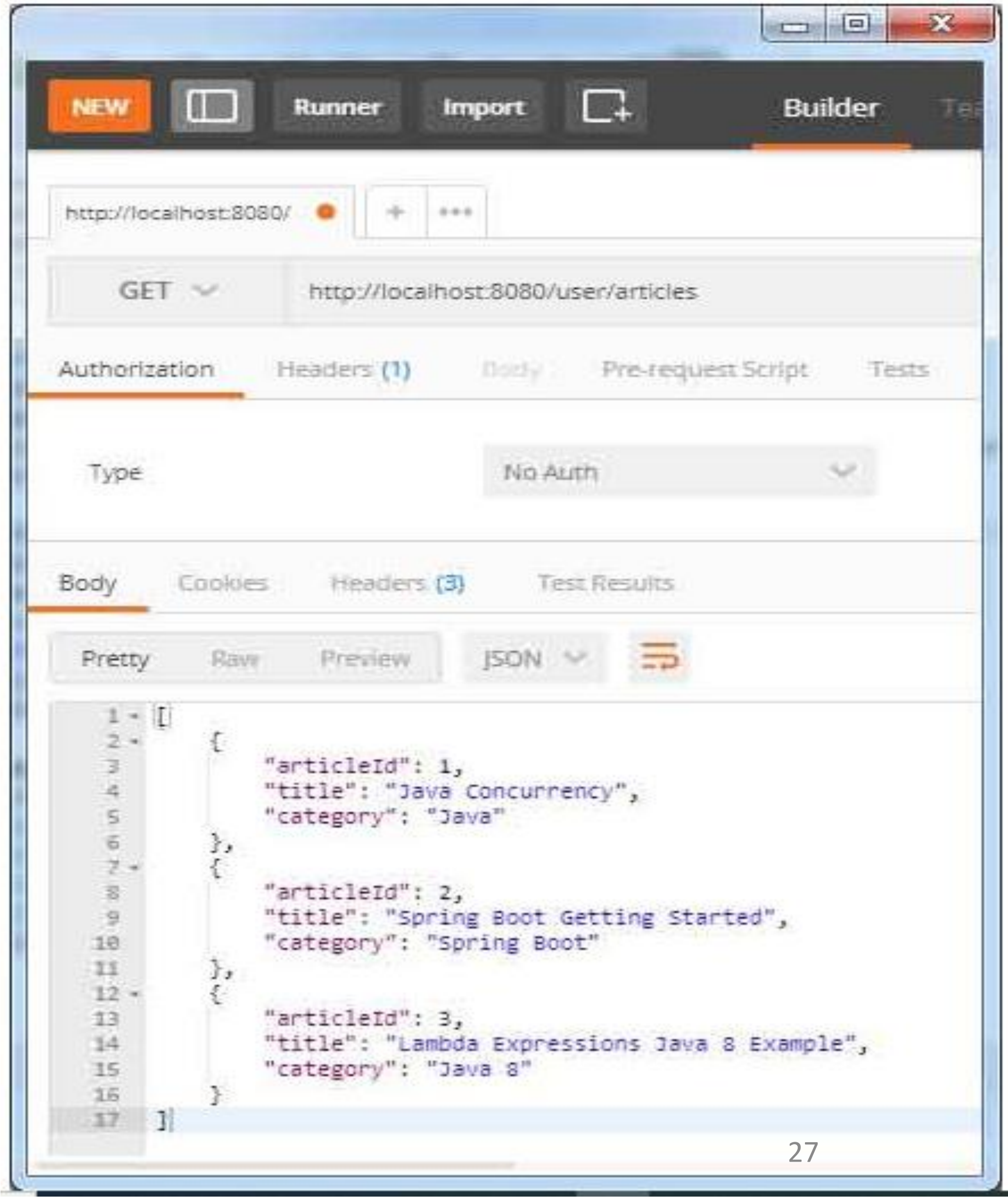
Spring Boot RESTFul Web Service URLs, Status Code, ..

1. Create :
HTTP Method: **POST**, URL: **/springboot02/article**
HTTP Response Status Code: **201 CREATED** & **409 CONFLICT**

2. Read : (Fetches article by id/All)
HTTP Method: **GET**, URL: **/ springboot02 /article/{id}**
HTTP Method: **GET**, URL: **/user/articles**
HTTP Response Status Code: **200 OK** & **404 Notfound**

3. Update :
HTTP Method: **PUT**, URL: **/springboot02/article**
HTTP Response Status Code: **200 OK**

4. Delete :
HTTP Method: **DELETE**, URL: **/springboot02 /article/{id}**
HTTP Response Status Code: **204 NO CONTENT**

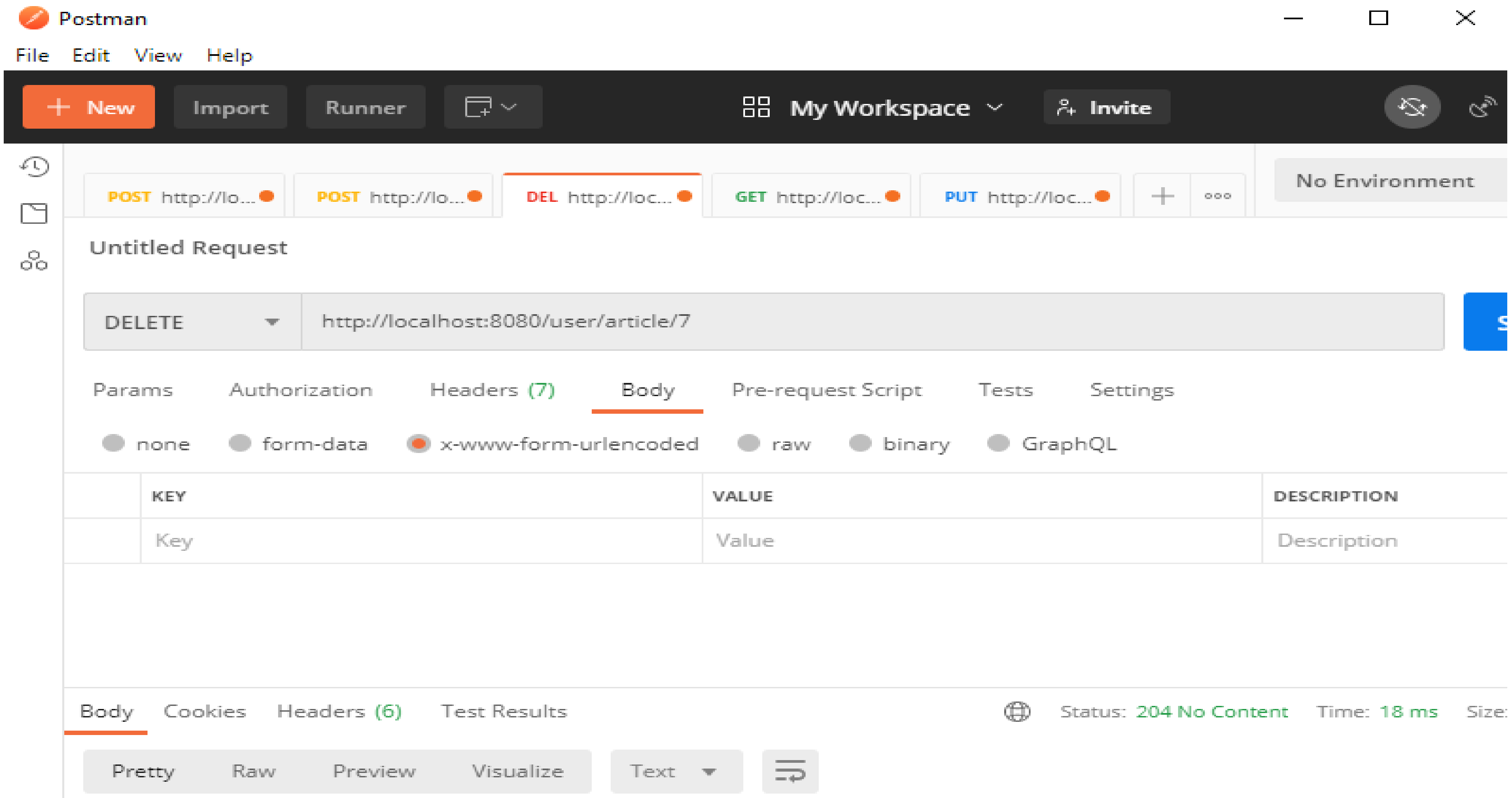


GET http://localhost:8080/user/articles

Type: No Auth

```
[
    {
        "articleId": 1,
        "title": "Java Concurrency",
        "category": "Java"
    },
    {
        "articleId": 2,
        "title": "Spring Boot Getting Started",
        "category": "Spring Boot"
    },
    {
        "articleId": 3,
        "title": "Lambda Expressions Java 8 Example",
        "category": "Java 8"
    }
]
```

Spring Boot RESTFul Testing con Postman



Untitled Request

DELETE http://localhost:8080/user/article/7

Params | Authorization | Headers (7) | Body | Pre-request Script | Tests | Settings

none | form-data | [✓ x-www-form-urlencoded] | raw | binary | GraphQL

	KEY	VALUE	DESCRIPTION
	Key	Value	Description

Body | Cookies | Headers (6) | Test Results — Status: 204 No Content Time: 18 ms Size:

Pretty | Raw | Preview | Visualize | Text

Spring Boot POST Mapping RESTFul Testing con client

POST Client Code: Codice lato client, per run di web service RESTful. Si usa il metodo postForLocation di RestTemplate

```
public void addArticleDemo() {
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    RestTemplate restTemplate = new RestTemplate();

    String url = "http://localhost:8080/springboot02/article";

    Article objArticle = new Article();
    objArticle.setArticleId(4);
    objArticle.setTitle("Spring REST Security using Hibernate");
    objArticle.setCategory("Spring");
    HttpEntity<Article> requestEntity = new HttpEntity<Article>(objArticle, headers);
    URI uri = restTemplate.postForLocation(url, requestEntity);
    System.out.println(uri.getPath());
}
```

Spring Boot GET mapping RESTFul Testing con client

GET Client Code: Codice lato client per run di web service RESTful, si usa il metodo exchange di RestTemplate con HTTP GET.

```
public void getArticleByIdDemo() {
   HttpHeaders headers = new HttpHeaders(); headers.setContentType(MediaType.APPLICATION_JSON);
   RestTemplate restTemplate = new RestTemplate();

   String url = "http://localhost:8080/ springboot02 /article/{id}";

   HttpEntity<String> requestEntity = new HttpEntity<String>(headers);
   ResponseEntity<Article> responseEntity = restTemplate.exchange(url, HttpMethod.GET, requestEntity,
                                                                   Article.class, 101);
   Article article = responseEntity.getBody();
   System.out.println("Id:"+article.getArticleId()+", Title:"+article.getTitle() +",
                              Category:"+article.getCategory()); }
```

Spring Boot PUT mapping RESTFul Testing con client

PUT Client Code: Per creare codice client, per run RESTful web service, si usa il metod put di RestTemplate.

```
public void updateArticleDemo() {
   HttpHeaders headers = new HttpHeaders();
   headers.setContentType(MediaType.APPLICATION_JSON);
   RestTemplate restTemplate = new RestTemplate();

   String url = "http://localhost:8080/ springboot02 /article";

   Article objArticle = new Article();
   objArticle.setArticleId(1);
   objArticle.setTitle("Update:Java Concurrency");
   objArticle.setCategory("Java"); HttpEntity<Article> requestEntity = new HttpEntity<Article>(objArticle, headers);
   restTemplate.put(url, requestEntity);
}
```

Spring Boot DELETE mapping RESTFul Testing con client

DELETE Client Code: Per creare codice client, per run RESTful web service, si usa il metodo exchange di RestTemplate con HTTP DELETE.

```
public void deleteArticleDemo() {
   HttpHeaders headers = new HttpHeaders();
   headers.setContentType(MediaType.APPLICATION_JSON);
   RestTemplate restTemplate = new RestTemplate();

   String url = "http://localhost:8080/ springboot02 /article/{id}";

   HttpEntity<Article> requestEntity = new HttpEntity<Article>(headers);
   restTemplate.exchange(url, HttpMethod.DELETE, requestEntity, Void.class, 4);
}
```