## Overview

This system provides functionality to upload an XML file containing product information, process the file to extract product details, and expose endpoints to retrieve product information via a REST API. The system is built using Spring Boot and utilizes JAXB for XML parsing.

## How It Works

Run the application by executing the `UploadXmlApplication` class, which will start a Spring Boot server on the default port `8080`. Next, open a web browser and navigate to `http://localhost:8080`. This will display the main page with a form to upload an XML file.

Once the file is submitted, it is sent to the `/upload` endpoint. The `FileController` handles this request, using the `FileService` to read and parse the XML content. If the file is successfully processed, a message indicating the number of records found will be displayed.

After uploading the file, links and forms become available to interact with the product data. You can view all products by clicking on the "See all products" link, which sends a request to the `/products` endpoint and returns a JSON list of all products. You can also search for a specific product by name. This form sends a request to the `/product/{name}` endpoint. If the product is found, its details are displayed; otherwise, a message indicating that the product does not exist is shown.

## Project Structure

- **uploadxml package**
  - `FileController.java`: Handles HTTP requests and defines endpoints.
  - `FileService.java`: Processes the uploaded XML file.
  - `Product.java`: Represents the product entity.
  - `Products.java`: Represents a collection of products.
  - `UploadXmlApplication.java`: The main application class.
- **resources/static**
  - `index.html`: Front-end page to upload files and search for products.
  - `script.js`: Handles the front-end logic using jQuery.

## Endpoints

### 1. File Upload Endpoint

- **URL**: `/upload`
- **Method**: POST
- **Description**: Uploads an XML file and processes it to extract product information.
- **Request Parameters**:
  - `file`: Multipart file containing the XML data.
- **Response**:
  - `200 OK`: If the file is successfully processed, returns the number of records found.
  - `400 Bad Request`: If no file is uploaded.
  - `500 Internal Server Error`: If an error occurs during file processing.

### 2. Get All Products Endpoint

- **URL**: `/products`
- **Method**: GET
- **Description**: Returns a list of all products in JSON format.
- **Response**:
  - `200 OK`: Returns the list of products.
  - `204 No Content`: If no products are available.

### 3. Get Product by Name Endpoint

- **URL**: `/product/{name}`
- **Method**: GET
- **Description**: Returns the product details for the given product name.
- **Path Variable**:
  - `name`: The name of the product to search for.
- **Response**:

    - `200 OK`: If the product is found, returns the product details.
    - `404 Not Found`: If no product is found with the given name.

## Classes

### FileController.java

- **Annotations**:
    - `@Controller`: Indicates that this class is a Spring MVC controller.
- **Endpoints**:
    - `@PostMapping("/upload")`: Handles the file upload and processes the XML.
    - `@GetMapping("/products")`: Returns all products.
    - `@GetMapping("/product/{name}")`: Returns a product by name.
- **Dependencies**:
    - `FileService`: Autowired service to process the file.
    - `Products`: Holds the processed products.

### FileService.java

- **Annotations**:
    - `@Service`: Indicates that this class is a Spring service.
- **Methods**:
    - `processFile(MultipartFile file)`: Reads and processes the XML file to extract products using JAXB.

### Product.java

- **Annotations**:
    - `@XmlRootElement(name = "Product")`: Specifies the root element for JAXB.
- **Attributes**:
    - `@XmlAttribute`: Marks the id as an XML attribute.
    - `@XmlElement`: Marks other fields as XML elements.
- **Fields**:
    - `int id`: Product ID.
    - `String name`: Product name.
    - `String category`: Product category.
    - `String partNumberNR`: Part number.
    - `String companyName`: Company name.
    - `boolean active`: Indicates if the product is active.
- **Constructors**:
    - Default constructor for JAXB.
    - Parameterized constructor for initialization.
- **Methods**:
    - Getter and setter methods for all fields.
    - `equals`: Overrides to compare products.

### Products.java

- **Annotations**:
    - `@XmlRootElement(name = "Products")`: Specifies the root element for JAXB.
- **Fields**:
    - `List<Product> products`: List of products.
- **Methods**:
    - `getProducts`: Returns the list of products.
    - `setProducts`: Sets the list of products.
    - `productsAmount`: Returns the number of products.
    - `findProduct`: Finds a product by name.

### UploadXmlApplication.java

- **Annotations**:
    - `@SpringBootApplication`: Marks the main class of the Spring Boot application.

- **Methods**:
  - `main(String[] args)`: The entry point of the application.

## Front-End

### index.html

- Provides a form to upload XML files and a form to search for products.
- Uses Thymeleaf for templating and jQuery for AJAX requests.

### script.js

- Handles the file upload form submission.
- Handles the product search form submission.
- Utilizes localStorage to persist messages and state.

## XML Example

Here is an example of the XML file structure expected by the application:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Products>
    <Product id="1">
        <Name>apple</Name>
        <Category>fruit</Category>
        <PartNumberNR>2303-E1A-G-M-W209B-VM</PartNumberNR>
        <CompanyName>FruitsAll</CompanyName>
        <Active>true</Active>
    </Product>
    <Product id="2">
        <Name>orange</Name>
        <Category>fruit</Category>
        <PartNumberNR>5603-J1A-G-M-W982F-PO</PartNumberNR>
        <CompanyName>FruitsAll</CompanyName>
        <Active>false</Active>
    </Product>
    <Product id="3">
        <Name>glass</Name>
        <Category>dish</Category>
        <PartNumberNR>9999-E7R-Q-M-K287B-YH</PartNumberNR>
        <CompanyName>HomeHome</CompanyName>
        <Active>true</Active>
    </Product>
</Products>
```