Assignment Title: PDF Content Segmenter with REST API in Java

Objective:

Develop a Java application that programmatically segments a system-generated PDF into distinct sections based on the whitespace between blocks of text, making exactly X cuts. The application should expose a REST API to accept PDF files, segment them, retrieve metadata, update segmentation details, and delete processed PDFs. The goal is to identify logical sections such as headings, paragraphs, and distinct blocks that are visually separated by increased whitespace, without using image processing techniques.

Requirements:

Input Specifications:

- The application should accept a PDF file as input via a REST API.
- The user should specify the number of cuts (X) to be made.

Processing Details:

- Analyze the PDF to identify sudden changes in whitespaces along the Y-axis that significantly separate text blocks.
- Define "significant whitespace" as vertical spaces that are noticeably larger than the typical line spacing within paragraphs.
- The application should make exactly X cuts based on the largest Y whitespace gaps.

Output Specifications:

- The output should be multiple PDF files, each containing one of the segments created by the cuts, returned via the REST API.
- Ensure that the segmentation does not cut through the middle of text blocks or paragraphs.

Constraints:

- Do not use image processing libraries or convert PDF pages to images for processing.
- Leverage Java PDF manipulation libraries such as Apache PDFBox or iText to analyze and manipulate the PDF content.

Technologies to Use:

- Java 11 or above
- Spring Boot for REST API development
- Maven or Gradle for build management
- Apache PDFBox or iText for PDF processing

Deliverables:

1. Source Code:
    - Include appropriate comments to enhance readability and maintainability.
    - Follow Java coding standards and best practices.
2. README.md File:
    - Document setup instructions.
    - Explain how to run the application.
    - Provide examples of API usage.
3. Unit Tests:
    - Include unit tests to demonstrate the functionality and handle edge cases.
    - Tests should cover both the segmentation logic and REST API functionality.
4. API Documentation:
    - Use Swagger/OpenAPI to document the API endpoints, request and response formats, and error messages.

1. PDF Segmentation Logic:

- **Task:** Develop a method to read and analyze PDF content using Apache PDFBox or iText. The method should identify large vertical whitespace to determine where cuts should be made.
- **Implementation:**
    - Parse text positions to calculate Y-axis gaps between text blocks.
    - Sort the gaps and select the largest X gaps to determine cut positions.
    - Split the PDF into segments at the calculated positions and save each segment as a new PDF.

2. REST API Development:

Set up a Spring Boot application to provide the following REST API endpoints for PDF segmentation:

1. **POST /segment-pdf**
    - **Description:** Accepts a PDF file and the number of cuts to be made, processes the PDF, and returns the segmented sections.
    - **Request:**
        - PDF file (multipart/form-data)
        - Number of cuts (X)
    - **Response:** Returns a zip file containing the segmented PDFs.
2. **GET /pdf-metadata/{id}**

    - **Description:** Retrieves metadata for a processed PDF, such as the number of segments and segment details.
    - **Request:**
        - `{id}`: The unique identifier for the processed PDF.
    - **Response:** JSON containing metadata about the processed PDF, including the number of segments, segment sizes, and positions.
3. **PUT /update-segmentation/{id}**
    - **Description:** Updates the segmentation details of a previously processed PDF by changing the number of cuts or adjusting segmentation positions.
    - **Request:**
        - `{id}`: The unique identifier for the processed PDF.
        - New segmentation details (e.g., updated number of cuts).
    - **Response:** Returns the updated segmented PDF files.
4. **PATCH /modify-segmentation/{id}**
    - **Description:** Partially updates segmentation details, such as modifying specific cut positions without reprocessing the entire PDF.
    - **Request:**
        - `{id}`: The unique identifier for the processed PDF.
        - JSON body with fields to modify (e.g., specific segment positions).
    - **Response:** Returns the modified segmented PDFs.
5. **DELETE /delete-pdf/{id}**
    - **Description:** Deletes a processed PDF and all its associated segments.
    - **Request:**
        - `{id}`: The unique identifier for the processed PDF.
    - **Response:** Confirms deletion of the specified PDF and its segments.

Quality Standards for REST API:

Mandatory Standards:

1. **HTTP Methods and Status Codes:**
    - Use correct HTTP methods:
        - `POST` for creating new segmented PDFs.
        - `GET` for retrieving metadata about processed PDFs.
        - `PUT` for updating entire segmentation details of a PDF.
        - `PATCH` for modifying specific segmentation details.
        - `DELETE` for removing a processed PDF and its segments.
    - Return appropriate status codes:
        - `200 OK` for successful operations.
        - `201 Created` for new resource creation.
        - `204 No Content` for successful deletions.

        - `400 Bad Request` for invalid inputs.
        - `404 Not Found` for non-existent resources.
        - `500 Internal Server Error` for server-side issues.
2. **Input Validation:**
    - Validate all incoming requests, including file type, file size, and parameters like the number of cuts.
    - Provide meaningful error messages with clear explanations for validation errors.
3. **Security:**
    - Implement authentication (API keys, OAuth tokens) to secure API access.
    - Use HTTPS to protect data in transit.
    - Implement input sanitization to prevent security vulnerabilities.
4. **Error Handling:**
    - Provide consistent error responses in JSON format.
    - Include clear error messages and codes that are informative to the user.
    - Handle exceptions gracefully without exposing stack traces.
5. **Performance:**
    - Ensure API endpoints respond promptly, even for large PDFs.
    - Optimize memory usage and processing speed for handling large documents.
    - Use pagination or streaming if applicable for large outputs.
6. **API Documentation:**
    - Use Swagger/OpenAPI for documenting all API endpoints.
    - Include descriptions of request parameters, response formats, and possible error messages.
    - Provide example requests and responses.