                                `

**Advanced Java Programming**

PROJECT REPORT

**TEXT SUMMARIZATION TOOL USING JSP**

**ABSTRACT**

This project aims to build a tool that can summarize long pieces of text using Java Server Pages (JSP). The tool helps create short and meaningful summaries of large texts by following the concept of **extractive summarization**. This means the tool selects important sentences directly from the original text, instead of rewriting or rephrasing them. It uses Natural Language Processing (NLP) techniques to pick out the most important parts of the text while leaving out unnecessary information. The tool leverages the **TF-IDF (Term Frequency-Inverse Document Frequency)** algorithm, a popular technique in NLP. This algorithm works by analyzing how often words appear in the text (term frequency) and how important they are based on their occurrence across multiple documents (inverse document frequency). Sentences that contain the most significant terms are chosen to be part of the summary. Powered by popular NLP libraries like Apache OpenNLP or Stanford NLP, the system processes the text by first breaking it down into smaller parts (called tokenizing) and then analyzing these parts to identify key sentences. Unnecessary or repetitive content is removed based on the TF-IDF scores. This tool will have a user-friendly web interface built using JSP. Users can upload documents or paste text, and the tool will process the text and generate a shorter version using extractive summarization techniques. The summary is then displayed for the user to read. This kind of tool can be very useful in areas like journalism, education, and research, where people need to quickly understand large amounts of information. By using JSP, the tool can be easily used on websites, and it has the potential to be part of bigger systems for companies or organizations.

**TABLE OF CONTENTS**

|  |
| --- |
| Chapter 1: Introduction  Chapter 2: System Design  2.1: Technologies used  2.2: Modules  Chapter 3: Database Design  Chapter 4: Implementation  Chapter 5: Source Code  Chapter 6: Conclusion   * Future Enhancements * References |

**CHAPTER 1: INTRODUCTION**

The main goal of this project is to create a web-based tool using JavaServer Pages (JSP) that automatically summarizes large amounts of text into concise, easy-to-read summaries. The system uses a frequency-based algorithm to pick out key sentences and important information from the input text. It will have a simple, user-friendly interface for easy interaction and efficient summarization. The tool is designed to keep the most important content while cutting out redundancy, making it a fast and scalable solution for summarizing lengthy documents. It can be applied to various areas, such as research papers, news articles, and reports. This project will showcase the combination of web technologies and natural language processing (NLP) to automate the text summarization process.

**CHAPTER 2: SYSTEM DESIGN**

The system design of the Text Summarization Web Application follows a three-tier architecture that includes the Presentation Layer, Business Logic Layer, and Data Layer. The goal is to provide users with a smooth and efficient way to upload text or documents, summarize them, and download the result as a PDF. The system's architecture ensures scalability, security, and ease of use.

* Presentation Layer: The front end of the system, where users interact with the application by submitting documents or text, uses JSP (Java Server Pages) for dynamic web pages and HTML/CSS for styling and structure.
* Business Logic Layer: This layer handles the summarization logic. The application uses Java Servlets to process user inputs, store sessions, and perform text summarization. The generated summary is then stored and displayed to the user.
* Data Layer: Responsible for managing session data and temporarily storing user-generated summaries. A database can be integrated in the future for storing user profiles and historical summaries.

The input and output design of this system focuses on simplicity, efficiency, and user-friendliness. The design ensures seamless interaction between the user and the application while handling text inputs and generating outputs in an organized manner.

* **Input Design**: Users are provided with two options for data input:
  1. **Text Input**: Users can paste the text they wish to summarize directly into a form field.
  2. **File Upload**: Users can upload a text-based document, such as a PDF. The application extracts the content of the PDF using the Apache PDFBox library.

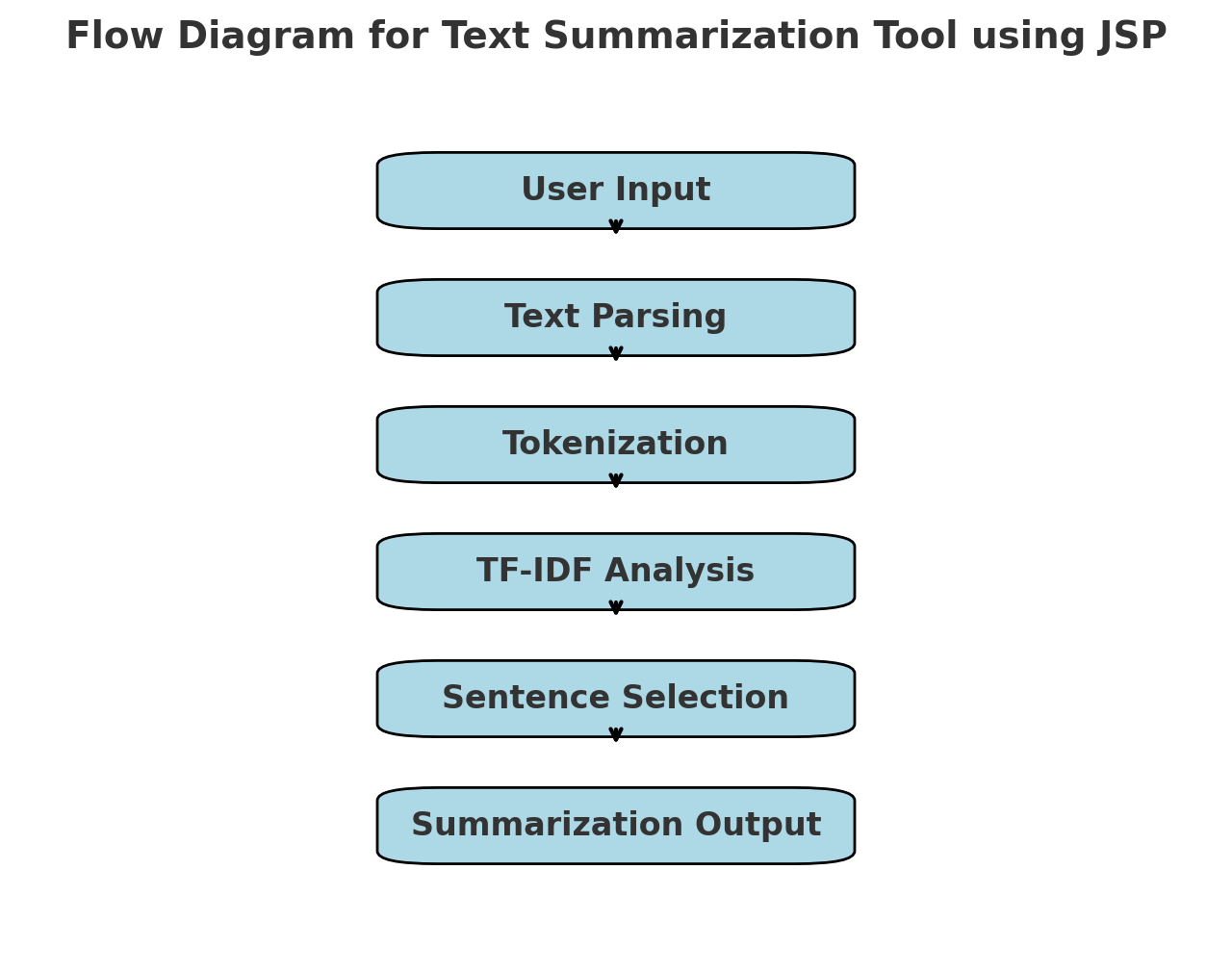
The input design ensures that the system can effectively handle both free text and structured document inputs, making it flexible and accessible.

* **Output Design**: The output is a summarized version of the input content, displayed on the web interface and also available for download:
  1. **Summarized Text Display**: The summary is presented in a user-friendly format, displayed within a container that adjusts to the size of the content. Proper formatting and paragraph structure enhance readability.
  2. **PDF Download Option**: Users can download the summarized content as a PDF. Using Apache PDFBox, the system generates a PDF file where the summary is formatted with appropriate headings, borders, and text wrapping, ensuring clarity and professional presentation.

This integrated design allows users to interact easily with the system, view the summary on the screen, and download it in a well-structured PDF format.

**2.1 TECHNOLOGY USED :**

* HTML
* CSS
* JSP
* Standford CoreNLP
* Apache PDFBox
* Database - Java Derby



**2.2 MODULES:**

The modules present in the project include the following,

1. Summary Page
2. Summarized Page

**SUMMARY PAGE**

1. Index file
2. SummarizeServlet file

**i) INDEX FILE:**

The index.jsp file is the frontend interface for the JSP Smart Summarizer application, designed to facilitate text summarization. It features an appealing layout with a gradient background and decorative shapes, creating a user-friendly experience. Users can choose their input type—manual text, PDF, or text file—through a dropdown menu. A textarea allows for direct text entry, while an optional field enables file path input for PDFs or text files. The interface includes a "Summarize" button to submit the input and a "Clear" button to reset all fields. The CSS styles enhance usability with hover effects and responsive design elements. Upon submission, the form data is processed by the SummarizeServlet, which handles the summarization logic. Overall, index.jsp seamlessly integrates visual design and functionality, offering users an interactive platform for text summarization.

**ii) SUMMARIZESERVLET PAGE:**

The SummarizeServlet is a Java servlet designed to handle text summarization requests from users, supporting multiple input types, including manual text input, PDF files, and text files. It utilizes Apache PDFBox for extracting text from PDF documents and Stanford NLP for text processing. Upon receiving a request, the servlet identifies the input type and extracts the relevant text accordingly. The extracted text is then cleaned to remove unnecessary characters and symbols before being processed for summarization. The servlet employs a TF-IDF (Term Frequency-Inverse Document Frequency) approach to rank sentences based on their importance, ultimately selecting the top sentences to generate a coherent summary. Additionally, it connects to a Derby database to store the summarization requests, including input type, input text, summarized text, and a timestamp, while also retrieving previous summaries to maintain a summary history for users. The summarized text and summary history are stored in the user session, facilitating easy access and display on the summary.jsp page. Overall, the SummarizeServlet effectively integrates text extraction, processing, and summarization capabilities, providing a user-friendly platform for summarizing text from various sources while maintaining a history of past requests. Future enhancements could include implementing advanced NLP techniques for improved summarization quality, enhancing the user interface for better user experience, and supporting additional file formats for text input.

**SUMMARIZED PAGE**

**i) SUMMARY FILE**

The summary.jsp page is designed to present the summarized text generated by the SummarizeServlet in a visually appealing and user-friendly manner. It features a soft gradient background that enhances the aesthetic appeal, while the main container displays the summary prominently, ensuring easy readability. The use of a clear and modern font, combined with appropriate padding and margin, creates a spacious layout that guides the user's focus toward the summarized content. The summary is displayed in a paragraph format, complete with a maximum height to prevent overflow, allowing users to scroll through longer texts if needed. Additionally, the page includes interactive elements, such as a button to download the summary as a PDF and a link to navigate back to the main interface, enhancing user experience and accessibility. Overall, this JSP page effectively combines functionality with design, ensuring that users can effortlessly view and download their summarized texts while maintaining an attractive and engaging interface.

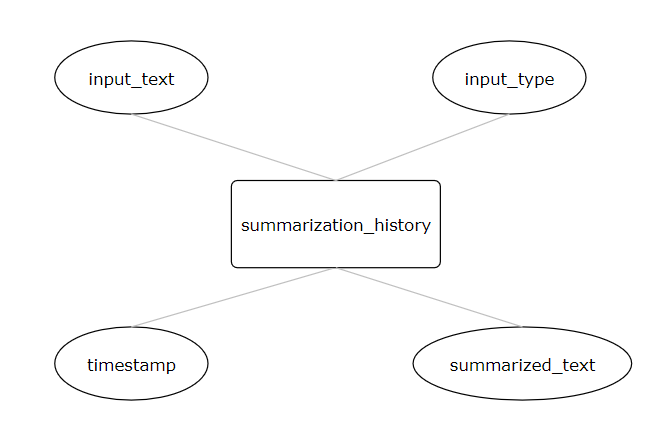
**ii) DOWNLOADSERVLET FILE:**

The DownloadServlet class generates a PDF containing a summary stored in the user's session. When a GET request is received, it retrieves and sanitizes the summary, then prepares the PDF for download with the correct content type and headers. Using Apache PDFBox, the servlet creates a document with a title, "Summary of the Provided Content," and formats the summary text to ensure readability. It also includes methods for sanitizing the text and wrapping it to fit the page width. If the summary is absent, the servlet returns a 404 error. This servlet effectively enables users to download their content summaries in a well-structured PDF format, enhancing the user experience. Future improvements may include customization options for the PDF layout.

**CHAPTER 3: DATABASE DESIGN**

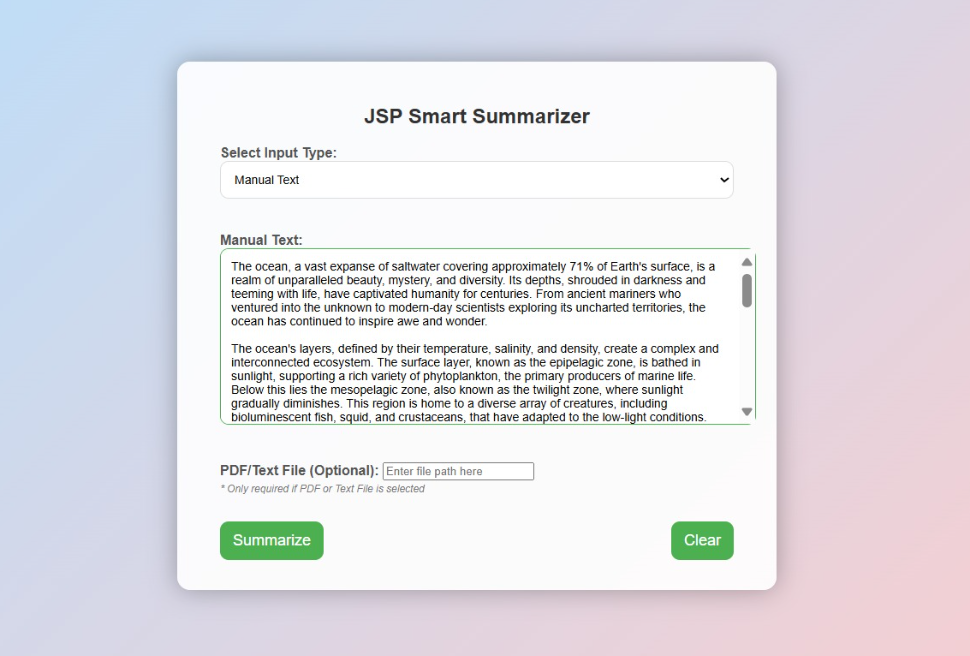
The system uses **Java DB (Apache Derby)** as the database, which is integrated within **NetBeans IDE 8.0**. The database consists of a single table called **summarization\_history** to efficiently store and manage the input and output of text summarization activities. This design simplifies the storage of user interactions while ensuring that all necessary data is recorded for future reference or processing.

* **Table Name:** summarization\_history
* **Attributes:**
  + input\_text: Stores the original text provided by the user, either entered directly or extracted from an uploaded document.
  + input\_type: Indicates the type of input, such as "text" for direct text input or "PDF" for file uploads.
  + summarized\_text: Stores the system-generated summary of the input text.
  + timestamp: Records the date and time when the summarization process was completed, allowing for tracking of summarization activities.

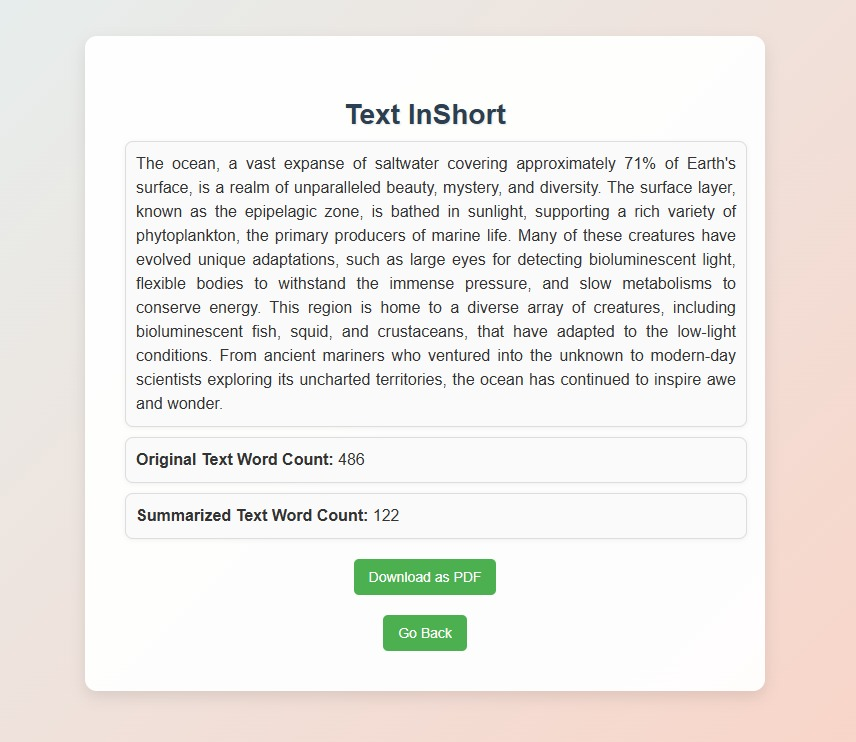
****

**Fig. 3.1** ER Diagram for Summarization\_history table

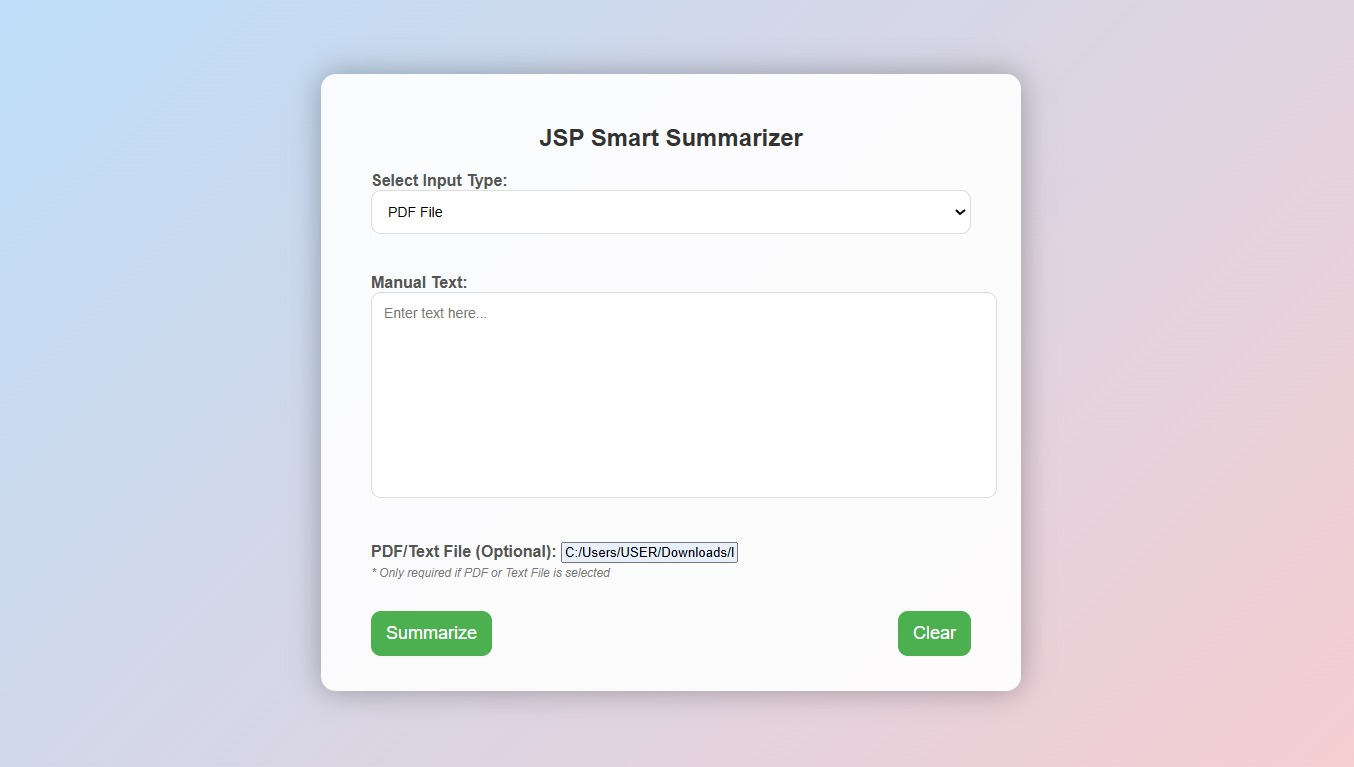
**CHAPTER 4: IMPLEMENTATION**



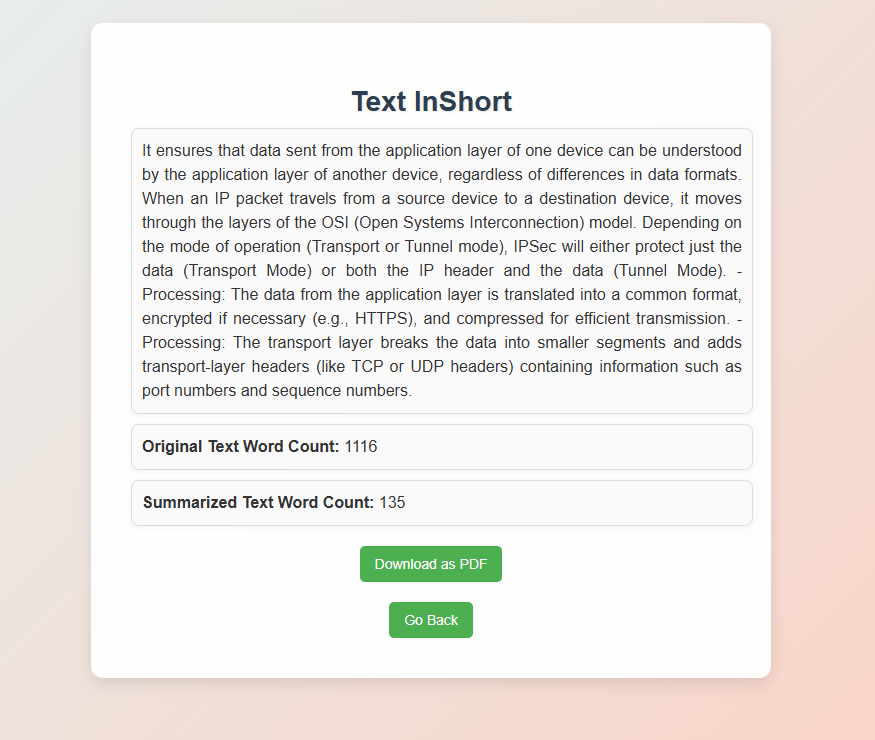
**Fig. 4.1** Manual Text Upload Section Page



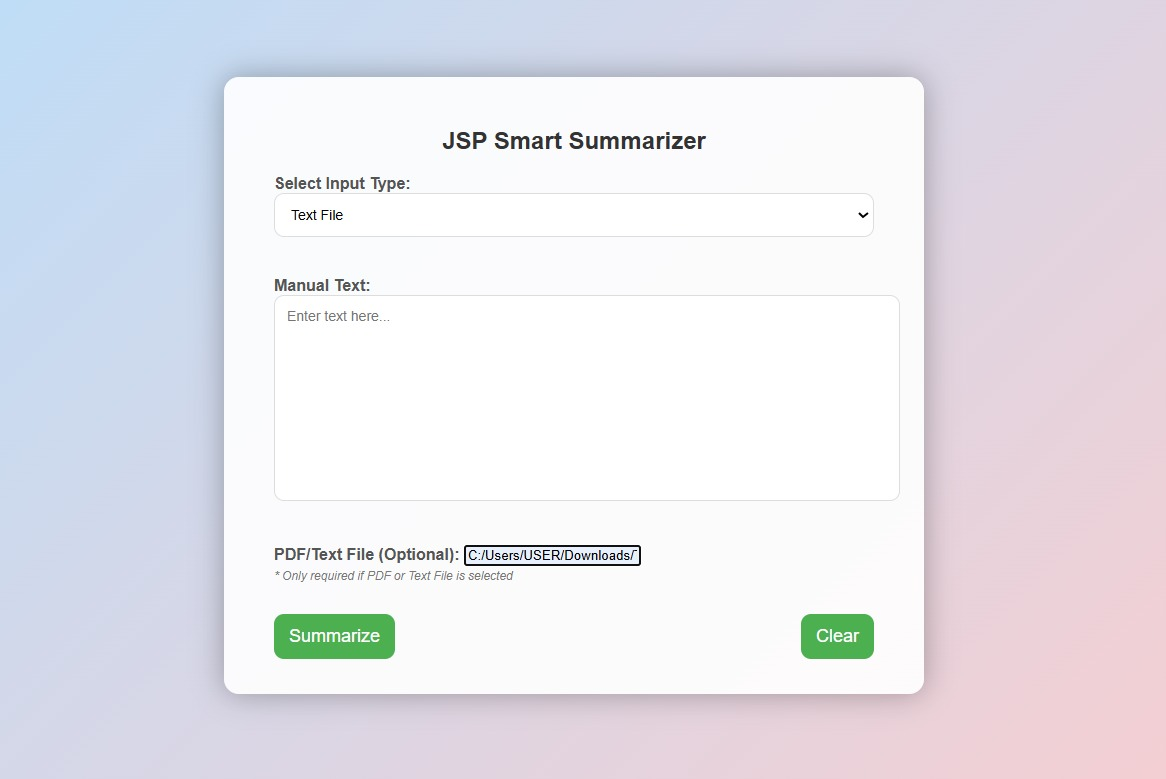
**Fig. 4.2** Summarized Page for Manual Text



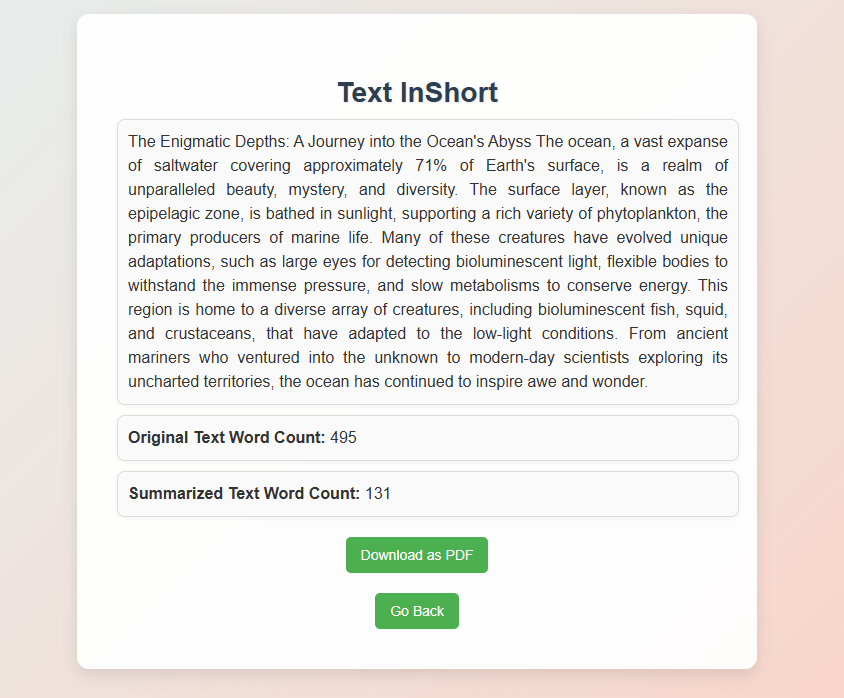
**Fig. 4.3** PDF File Upload Section Page



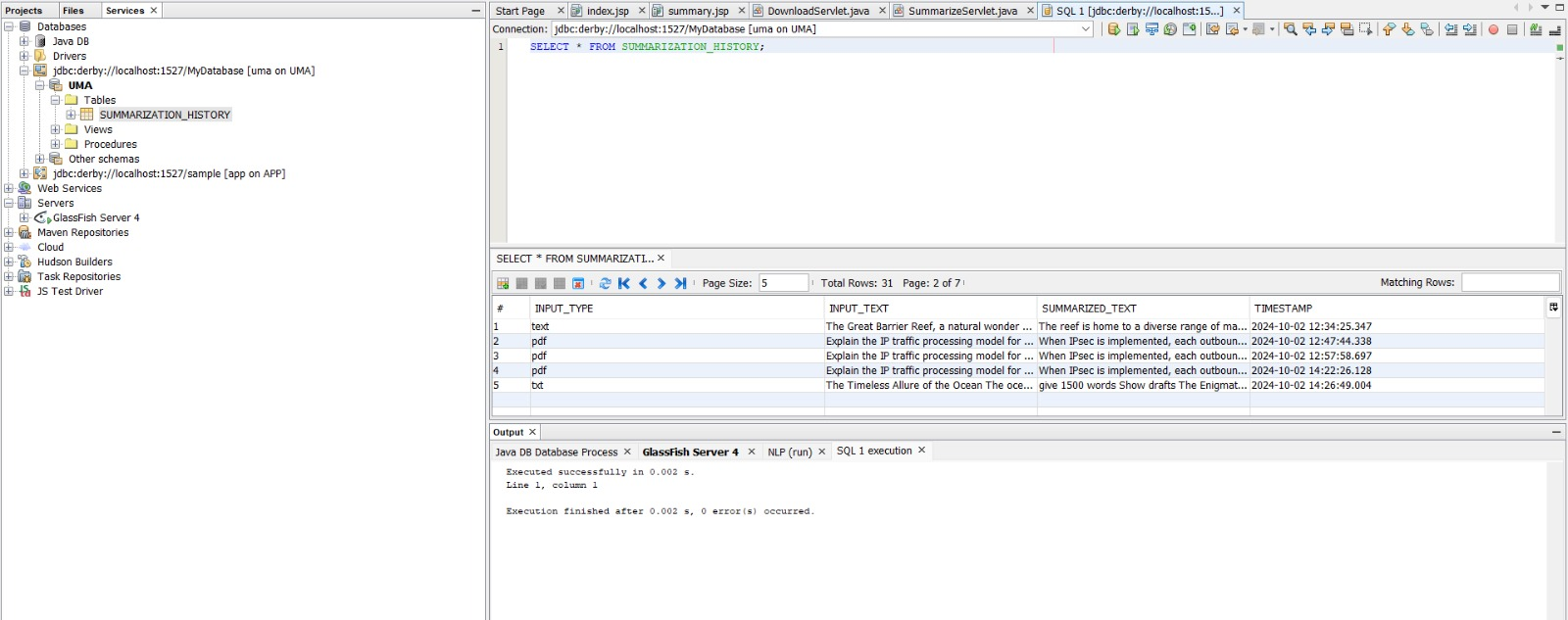
**Fig. 4.4** Summarized Page for PDF File



**Fig. 4.5** Text File Upload Section



**Fig. 4.6** Summarized Page for Text File



**Fig 4.7** Summarization History Record Table

**CHAPTER 5: SOURCE CODE**

1. **index.jsp**

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

    <title>Text Summarizer</title>

    <style>

        body {

            font-family: 'Arial', sans-serif;

            margin: 0;

            padding: 0;

            display: flex;

            justify-content: center; align-items: center;

            height: 100vh;

            background: linear-gradient(135deg, #b3e0ff, #ffcccb); /\* Soft gradient \*/

            position: relative;

            overflow: hidden; /\* Prevents overflow of shapes \*/

        }

        .container {

            background-color: rgba(255, 255, 255, 0.9); /\* Slight transparency \*/

            border-radius: 15px;

            box-shadow: 0 0 30px rgba(0, 0, 0, 0.3);

            padding: 30px 50px;

            width: 600px; /\* Increased width for better layout \*/

            position: relative;

            z-index: 1; /\* Bring the container above the background shapes \*/

        }

        /\* Adding decorative shapes \*/

        .shape {

            position: absolute;

            border-radius: 50%;

            opacity: 0.15; /\* Transparency for the shapes \*/

        }

        .shape1 {

            width: 300px;

            height: 300px;

            background: #f0f8ff;

            top: -50px;

            left: -50px;

        }

        .shape2 {

            width: 400px;

            height: 400px;

            background: #ffebcd;

            bottom: -100px;

            right: -100px;

        }

        h2 {

            text-align: center;

            color: #333;

        }

        form {

            display: flex;

            flex-direction: column;

        }

/\* Other existing styles... \*/

        label {

            margin: 10px 0 5px;

            font-weight: bold;

            color: #555;

        }

        select, textarea, input[type="file"] {

            width: 100%;

            padding: 12px;

            border-radius: 10px; /\* Increased border radius \*/

            border: 1px solid #ddd;

            font-size: 14px;

            margin-bottom: 15px;

            transition: border-color 0.3s;

        }

        select:focus, textarea:focus, input[type="file"]:focus {

            border-color: #4CAF50;

            outline: none;

        }

        textarea {

            resize: none;

            height: 180px; /\* Increased height for textarea \*/

            font-family: 'Arial', sans-serif;

        }

        input[type="submit"], .clear-btn {

            background-color: #4CAF50;

            color: white;

            padding: 12px 15px;

            border: none;

            border-radius: 10px;

            cursor: pointer;

            font-size: 18px; /\* Increased font size \*/

            transition: background-color 0.3s, transform 0.2s;

            margin: 5px 0; /\* Spacing between buttons \*/

        }

        input[type="submit"]:hover, .clear-btn:hover {

            background-color: #45a049;

           transform: translateY(-2px);

        }

        .input-group {

            margin-bottom: 25px; /\* Increased spacing \*/

        }

        .optional {

            color: #777;

            font-size: 12px;

            font-style: italic;

            margin-top: 5px; /\* Add some space above optional note \*/

        }

        .button-group {

            display: flex;

            justify-content: space-between; /\* Space between buttons \*/

        }

    </style>

</head>

<body>

    <div class="container">

        <h2>JSP Smart Summarizer</h2>

        <form action="SummarizeServlet" method="post">

            <div class="input-group">

                <label for="inputType">Select Input Type:</label>

                <select id="inputType" name="inputType" required>

                    <option value="text">Manual Text</option>

                    <option value="pdf">PDF File</option>

                    <option value="txt">Text File</option>

                </select>

            </div>

            <div class="input-group">

                <label for="inputText">Manual Text:</label>

                <textarea id="inputText" name="inputText" placeholder="Enter text here..."></textarea>

            </div>

            <div class="input-group">

                <label for="filePath">PDF/Text File (Optional):</label>

                <input id="filePath" type="text" name="filePath" placeholder="Enter file path here" /><br>

                <span class="optional">\* Only required if PDF or Text File is selected</span>

            </div>

            <div class="button-group">

                <input type="submit" value="Summarize">

                <input type="button" class="clear-btn" value="Clear" onclick="clearText()">

            </div>

        </form>

    </div>

    <script>

        function clearText() {

            document.getElementById("inputText").value = ""; // Clear the text area

            document.getElementById("filePath").value = "";  // Clear the file path

            document.getElementById("inputType").selectedIndex = 0; // Reset the dropdown

        }

    </script>

</body>

</html>

1. **summary.jsp**

        <%@ page session="true" %>

<!DOCTYPE html>

<html>

<head>

    <title>Text Summary</title>

    <style>

        body {

            font-family: 'Arial', sans-serif;

            background: linear-gradient(135deg, #e0f7fa, #ffccbc); /\* Soft gradient background \*/

            margin: 0;

            padding: 0;

            display: flex;

            justify-content: center;

            align-items: center;

            height: 100vh;

            overflow: hidden;

        }

        h2 {

            color: #2c3e50; /\* Darker text color for better readability \*/

            text-align: center;

            margin-top: 20px;

            font-size: 36px;

            text-shadow: 1px 1px 2px rgba(0, 0, 0, 0.1); /\* Subtle shadow for depth \*/

        }

        .container {

            width: 80%; /\* Increased width for better visibility \*/

            max-width: 600px; /\* Limit maximum width \*/

            background-color: rgba(255, 255, 255, 0.9); /\* Slightly transparent background \*/

            box-shadow: 0 8px 20px rgba(0, 0, 0, 0.1);

            padding: 40px;

            border-radius: 15px;

            margin-top: 40px;

            display: flex;

            flex-direction: column;

            align-items: center; /\* Center align content \*/

        }

        p {

            font-size: 18px;

            color: #333;

            line-height: 1.6;

            text-align: justify;

            padding: 15px;

            background-color: #fafafa;

            border-radius: 10px;

            border: 1px solid #ddd;

            max-height: 400px;

            overflow-y: auto;

            margin: 20px 0; /\* Added spacing between paragraphs \*/

            width: 100%; /\* Full width \*/

            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.05); /\* Soft shadow \*/

        }

        a {

            display: inline-block;

            text-decoration: none;

            background-color: #4CAF50;

            color: white;

            padding: 12px 20px; /\* Increased padding \*/

            border-radius: 5px;

            text-align: center;

            margin-top: 20px;

            transition: background-color 0.3s, transform 0.2s;

            font-size: 16px; /\* Increased font size \*/

        }

        a:hover {

            background-color: #45a049;

            transform: translateY(-2px); /\* Lift effect \*/

        }

        .back-link {

            display: block;

            text-align: center;

            margin-top: 20px;

            width: 100%; /\* Full width for back link \*/

        }

    </style>

</head>

<body>

    <div class="container">

        <h2>Text InShort</h2>

        <p>${sessionScope.summary}</p>

        <div class="back-link">

            <a href="DownloadServlet?summary=${sessionScope.summary}" target="\_blank">Download as PDF</a>

            <div class="back-link" style="margin-top: 10px;">

                <a href="index.jsp" class="back-button">Go Back</a>

            </div>

        </div>

    </div>

</body>

</html>

1. **SummarizeServlet.java**

import edu.stanford.nlp.pipeline.\*;

import edu.stanford.nlp.ling.CoreAnnotations;

import edu.stanford.nlp.util.CoreMap;

import org.apache.pdfbox.pdmodel.PDDocument;

import org.apache.pdfbox.text.PDFTextStripper;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.\*;

import java.io.File;

import java.io.IOException;

import java.sql.ResultSet;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import java.util.\*;

@WebServlet("/SummarizeServlet")

public class SummarizeServlet extends HttpServlet {

    private static final String DB\_URL = "jdbc:derby://localhost:1527/MyDatabase";

    private static final String DB\_USER = "uma";

    private static final String DB\_PASSWORD = "123";

    // Summarization entry-point for POST request

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        String inputType = request.getParameter("inputType");

        String inputText = "";

        String filePath = request.getParameter("filePath");

        // Handle the input type: either a PDF file, Text file, or manual input

        if ("text".equals(inputType)) {

            inputText = request.getParameter("inputText");

        } else if ("pdf".equals(inputType)) {

            inputText = extractTextFromPDF(filePath);

        } else if ("txt".equals(inputType)) {

            inputText = readFileText(filePath);

        }

        // Summarize the text

        String summarizedText = summarizeText(inputText);

        // Save the summarization request to the database

        saveSummaryToDatabase(inputType, inputText, summarizedText);

        // Retrieve summary history from the database and store it in the session

        List<String> summaryHistory = getSummaryHistory();

        HttpSession session = request.getSession();

        session.setAttribute("summaryHistory", summaryHistory);

        // Send summary to JSP

        session.setAttribute("summary", summarizedText);

        response.sendRedirect("summary.jsp");

    }

    // Extract text from a PDF file

    private String extractTextFromPDF(String pdfPath) throws IOException {

        File file = new File(pdfPath);

        try (PDDocument document = PDDocument.load(file)) {

            PDFTextStripper pdfStripper = new PDFTextStripper();

            return cleanText(pdfStripper.getText(document)); // Clean the extracted text

        }

    }

    // Read text from a file

    private String readFileText(String filePath) throws IOException {

        StringBuilder contentBuilder = new StringBuilder();

        try (Scanner scanner = new Scanner(new File(filePath))) {

            while (scanner.hasNextLine()) {

                contentBuilder.append(scanner.nextLine()).append("\n");

            }

        }

        return cleanText(contentBuilder.toString()); // Clean the file text

    }

    // Clean the text by removing unnecessary characters or symbols

    private String cleanText(String text) {

        return text.replaceAll("\\?", "").trim();

    }

    // Summarize text using advanced techniques like TF-IDF

    private String summarizeText(String text) {

        if (text == null || text.isEmpty()) return "No text provided.";

        // Setup Stanford NLP pipeline

        Properties props = new Properties();

        props.setProperty("annotators", "tokenize,ssplit,pos,lemma");

        StanfordCoreNLP pipeline = new StanfordCoreNLP(props);

        // Annotate the text

        Annotation document = new Annotation(text);

        pipeline.annotate(document);

        // Extract sentences from the text

        List<CoreMap> sentences = document.get(CoreAnnotations.SentencesAnnotation.class);

        // Calculate TF-IDF scores for words

        Map<String, Double> tfidfScores = calculateTFIDF(sentences);

        // Rank sentences based on TF-IDF and sentence position

        List<String> rankedSentences = rankSentences(sentences, tfidfScores, 5);

        // Join the ranked sentences into a final summary

        return String.join(" ", rankedSentences);

    }

    // Calculate TF-IDF scores for words

    private Map<String, Double> calculateTFIDF(List<CoreMap> sentences) {

       Map<String, Integer> wordFreq = new HashMap<>();

        Map<String, Double> tfidfScores = new HashMap<>();

        int totalWordCount = 0;

        // Calculate word frequency

        for (CoreMap sentence : sentences) {

            String[] words = sentence.toString().toLowerCase().split("\\W+");

            totalWordCount += words.length;

            for (String word : words) {

                if (!word.isEmpty()) {

                    wordFreq.put(word, wordFreq.getOrDefault(word, 0) + 1);

                }

            }

        }

        // Calculate TF (Term Frequency)

        for (Map.Entry<String, Integer> entry : wordFreq.entrySet()) {

            double tf = (double) entry.getValue() / totalWordCount;

            tfidfScores.put(entry.getKey(), tf);

        }

        // Inverse Document Frequency (IDF) based on word frequency

        for (Map.Entry<String, Double> entry : tfidfScores.entrySet()) {

            String word = entry.getKey();

            double idf = Math.log(1 + (1.0 / (wordFreq.get(word) + 1))); // Simple IDF assumption

            tfidfScores.put(word, entry.getValue() \* idf);

        }

        return tfidfScores;

    }

    // Rank sentences based on TF-IDF scores and sentence position

    private List<String> rankSentences(List<CoreMap> sentences, Map<String, Double> tfidfScores, int limit) {

        PriorityQueue<Map.Entry<String, Double>> rankedSentences = new PriorityQueue<>(

                (a, b) -> b.getValue().compareTo(a.getValue())

        );

        // Score each sentence based on TF-IDF and position (earlier sentences have slightly higher weight)

        for (int i = 0; i < sentences.size(); i++) {

            CoreMap sentence = sentences.get(i);

            String sentenceText = sentence.toString();

            double score = calculateSentenceScore(sentenceText, tfidfScores) \* (1 + 0.05 \* (sentences.size() - i));

            rankedSentences.add(new AbstractMap.SimpleEntry<>(sentenceText, score));

        }

        // Extract top N sentences

        List<String> topSentences = new ArrayList<>();

        for (int i = 0; i < limit && !rankedSentences.isEmpty(); i++) {

            topSentences.add(rankedSentences.poll().getKey());

        }

        return topSentences;

    }

    // Calculate the score of a sentence based on TF-IDF scores

    private double calculateSentenceScore(String sentence, Map<String, Double> tfidfScores) {

        String[] words = sentence.toLowerCase().split("\\W+");

        double score = 0.0;

        for (String word : words) {

            score += tfidfScores.getOrDefault(word, 0.0);

        }

        return score;

    }

    // Method to save the summary to the database

    private void saveSummaryToDatabase(String inputType, String inputText, String summarizedText) {

        try (Connection conn = DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD)) {

            String sql = "INSERT INTO SUMMARIZATION\_HISTORY (INPUT\_TYPE, INPUT\_TEXT, SUMMARIZED\_TEXT, TIMESTAMP) VALUES (?, ?, ?, ?)";

            try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

                pstmt.setString(1, inputType);

                pstmt.setString(2, inputText);

                pstmt.setString(3, summarizedText);

                pstmt.setTimestamp(4, new java.sql.Timestamp(new Date().getTime()));

                pstmt.executeUpdate();

                System.out.println("Summary successfully stored in the database!");

            }

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

    // Method to retrieve summary history from the database

    private List<String> getSummaryHistory() {

        List<String> summaries = new ArrayList<>();

        try (Connection conn = DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD)) {

            String sql = "SELECT SUMMARIZATION\_TEXT FROM SUMMARIZATION\_HISTORY ORDER BY TIMESTAMP DESC"; // Use correct table name

            try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

                ResultSet rs = pstmt.executeQuery();

                while (rs.next()) {

                    summaries.add(rs.getString("SUMMARIZED\_TEXT")); // Use the correct column name

                }

            }

        } catch (SQLException e) {

            e.printStackTrace();

        }

        return summaries;

    }

}

1. **DownloadServlet.java**

import org.apache.pdfbox.pdmodel.PDDocument;

import org.apache.pdfbox.pdmodel.PDPage;

import org.apache.pdfbox.pdmodel.PDPageContentStream;

import org.apache.pdfbox.pdmodel.font.PDType1Font;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import java.io.IOException;

import java.io.OutputStream;

import java.util.ArrayList;

import java.util.List;

public class DownloadServlet extends HttpServlet {

    @Override

    protected void doGet(HttpServletRequest request, HttpServletResponse response)

            throws ServletException, IOException {

        HttpSession session = request.getSession();

        String summary = (String) session.getAttribute("summary"); // Get the summary from session

        if (summary != null) {

            // Sanitize the summary text to remove unsupported characters

            summary = sanitizeText(summary);

            response.setContentType("application/pdf");

            response.setHeader("Content-Disposition", "attachment; filename=\"summary.pdf\"");

            try (PDDocument document = new PDDocument()) {

                PDPage page = new PDPage();

                document.addPage(page);

                try (PDPageContentStream contentStream = new PDPageContentStream(document, page)) {

                    drawBorder(contentStream, page, 30);

                    contentStream.setFont(PDType1Font.HELVETICA\_BOLD, 16);

                    contentStream.beginText();

                    contentStream.newLineAtOffset((page.getMediaBox().getWidth() - 200) / 2, 700);

                    contentStream.showText("Summary of the Provided Content");

                    contentStream.endText();

                    contentStream.beginText();

                    contentStream.newLineAtOffset(0, -20);

                    contentStream.showText("");

                    contentStream.endText();

                    contentStream.setFont(PDType1Font.HELVETICA, 12);

                    contentStream.beginText();

                    contentStream.newLineAtOffset(50, 600);

                    List<String> lines = wrapText(summary, PDType1Font.HELVETICA, 12, 500);

                    for (int i = 0; i < lines.size(); i++) {

                        String line = lines.get(i);

                        if (i == 0) {

                            contentStream.newLineAtOffset(15, 0);

                        }

                        contentStream.showText(line);

                        contentStream.newLineAtOffset(0, -15);

                    }

                    contentStream.endText();

                }

                try (OutputStream out = response.getOutputStream()) {

                    document.save(out);

                }

            }

        } else {

            response.sendError(HttpServletResponse.SC\_NOT\_FOUND, "Summary not found.");

        }

    }

    private String sanitizeText(String text) {

        return text.replaceAll("[^\\x20-\\x7E]", " "); // Keep only printable ASCII characters

    }

    private List<String> wrapText(String text, PDType1Font font, float fontSize, float width) throws IOException {

        List<String> wrappedLines = new ArrayList<>();

        String[] words = text.split(" ");

        StringBuilder line = new StringBuilder();

        for (String word : words) {

            float lineWidth = font.getStringWidth(line.toString() + word) / 1000 \* fontSize;

            if (lineWidth > width) {

                wrappedLines.add(line.toString().trim());

                line = new StringBuilder(word + " ");

            } else {

                line.append(word).append(" ");

            }

        }

        wrappedLines.add(line.toString().trim());

        return wrappedLines;

    }

    private void drawBorder(PDPageContentStream contentStream, PDPage page, float borderSize) throws IOException {

        float margin = borderSize;

        float width = page.getMediaBox().getWidth();

        float height = page.getMediaBox().getHeight();

        contentStream.setLineWidth(1f);

        contentStream.moveTo(margin, margin);

        contentStream.lineTo(width - margin, margin);

        contentStream.lineTo(width - margin, height - margin);

        contentStream.lineTo(margin, height - margin);

        contentStream.closePath();

        contentStream.stroke();

    }

}

**CHAPTER 6: CONCLUSION**

The **JSP Smart Summarizer** project provides an easy-to-use platform for creating short summaries from both manually entered text and uploaded PDF documents. It uses Java Servlets, Apache PDFBox, and a Java Derby database to manage text summarization, keep track of user input history, and generate downloadable summaries in PDF format. The design focuses on providing a smooth user experience, combining a clean front-end layout with strong back-end features. This project highlights how Java-based web technologies can create helpful tools for real-life use and opens up possibilities for future improvements in natural language processing and web development.

**FUTURE ENHANCEMENTS**

* **Abstractive Summarization**: Implementing abstractive summarization for more natural, human-like summaries.
* **Deep Learning Integration**: Utilizing deep learning models like BERT or GPT to improve summary accuracy and context.
* **Multiple Export Formats:** Providing options to export summaries as PDF, Word, or Excel files.

**REFERENCES**

<https://github.com/LunaticPrakash/Text-Summarization>