DESIGN OF A WEB BASED E-LIBRARY WITH A SEARCHABLE PDF VIEWER

BY

OKI RAPHAEL EBIWENI

PSC1805323

DEPARTMENT OF COMPUTER SCIENCE,

FACULTY OF PHYSICAL SCIENCES,

UNIVERSITY OF BENIN,

BENIN CITY,

EDOSTATE, NIGERIA.

NOVEMBER 2025

DESIGN OF A WEB BASED E-LIBRARY WITH A SEARCHABLE PDF VIEWER

BY

OKI RAPHAEL EBIWENI

PSC1805323

A PROJECT REPORT SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE, FACULTY OF PHYSICAL SCIENCES, UNIVERSITY OF BENIN, BENIN CITY

IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE AWARD OF A

BACHELOR OF SCIENCE (B.Sc.) DEGREE IN COMPUTER SCIENCE

NOVEMBER 2025

CERTIFICATION

This is to certify that this project work was carried out by **OKI RAPHAEL EBIWENI** with Matriculation Number **PSC1805323** under my supervision. It is adequate and satisfactory, both in scope and content, for the award of Bachelor of Science (B.sc) Degree in Computer Science of the University of Benin.

---------- ----------

DR. EMMANUEL NWELIH **DATE**

Project Supervisor

APPROVAL

This project work is hereby approved in partial fulfilment of the requirements for the award of Bachelor of Science (B.Sc.) Degree in Computer Science from the University of Benin.

---------- ----------

DR. EMMANUEL NWELIH **DATE**

Project Supervisor

---------- ----------

DR. (MRS.) ROSEMARY USIOBAIFO **DATE**

Head of Department

DEDICATION

This seminar is dedicated to God for giving me the life, wisdom and grace to carry out the research and project work successfully, as well as to everyone who made efforts in assisting to ensure that this project was carried out without much issues.

ACKNOWLEDGEMENT

I would like to give my thanks to my project supervisor, Dr. Emmanuel Nwelih for providing me with his constant guidance on how to go about my project research, implementation and report writing.

I would also like to thank my project coordinator, Dr. Maxwell S.U. Osagie for always ensuring to pass along all necessary information, answering questions, as well as making sure all our 400level project activities are put in order without much confusion.

I would also like to thank my parents, Mr. and Mrs. Oki, and my siblings for their support, and encouragement over the course of my studies. Without them, would not be in this school to carry out this project. I would also like to express my gratitude to all those who supported me in one way or the other over the course of my program here

Finally, I want to acknowledge God Almighty for giving me the strength and wisdom to complete this study. His grace has been sufficient for me, and I am forever grateful for His blessings.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

While traditional libraries face limitations in accessibility and scale, web-based e-libraries have emerged as indispensable platforms for delivering vast digital content. They come with benefits such as the prevalent one of offering 24/7 access to digital resources from any device with internet accessibility. This project proposes the design of a web-based e-library system using HTML, CSS and Javascript programming, specifically engineered to address the real-world problems of fragmented user experiences; which require the use of multiple tools for interacting with different file types, limited search functionalities within digital documents, especially Portable Document Format (PDF) files, inconsistent search capabilities, and over-reliance on external viewing applications. The core innovation lies in the integration of a robust, searchable PDF viewer directly within the platform, enabling users to perform full-text searches not only across document metadata but also within the content of uploaded PDF files. The system will feature a centralized, intuitive interface for managing and accessing a wide range of digital materials, including books, journals, and research papers, regardless of their original format. Furthermore, the proposed e-library will prioritize user accessibility through features such as responsive design and clear navigation. By providing an integrated, highly searchable, and user-friendly platform, the success of this project aims to significantly improve the efficiency and effectiveness of digital information access, thereby fostering a more productive learning and research environment.

CHAPTER ONE

INTRODUCTION

1.0 Background Study

Libraries are structures created for the purpose of storing, organizing and accessing information and any materials relevant to the propagation of said information, in most cases for the purpose of study. Libraries can be either traditional; where physical materials of information are stored in a physical space, or digital; such as e-libraries where information is stored, organized and accessed through the use of a computer; that is, in digital format without respect to the physical location of said information (Pacific, 1977). The evolution of information access across the ages has contributed to the rise of e-libraries, transforming how we discover and consume knowledge.

The growth and evolution of digital libraries, ranging from early digitization efforts to complex information management systems, have been driven by the increasing volume of digital content and advancements in network infrastructure over the years. Universities, in particular, have been at the forefront of adopting e-libraries to support diverse academic activities, ranging from remote learning to cutting-edge research. Despite this progress, a critical examination of existing e-library architectures reveals persistent challenges. These often stem from a legacy of integrating disparate systems, leading to non-uniform interfaces, complex content ingestion processes, and most notably, an inability to provide seamless and deep interaction with widely used document formats like PDFs.

Cases such as ‘Open Library’, which launched as a project of Internet Archive in 2006, is an important digital library hosting digital publications. The archive created from scanned books causes large file sizes as a result of this the slowness in the document view process is felt by the user. Extracting text information from scanned books is achieved by optical character recognition (Hasbrouck, 2020). There is also the problem where located books could not be read directly on the site, and would lead to the user being redirected to another website to access the content. The project is open source code and has audiobook reading feature in text information. Project Gutenberg, founded in 1971 by Michael S. Hart is another example of a digital library in which you can read a document online, but you cannot search within the document; as you would have to download the document before being able to do so.

Current solutions for such problems would frequently externalize PDF viewing, forcing users to download files or depend on sometimes problematic browser plugins, resulting in the disruption of the workflow and diminishing the overall utility of the e-library.

1.1 Motivation

The motivation behind this project stems from the recognized need for more efficient and user-friendly digital resource access within academic environments, as well as to enhance the accessibility and usability of digital libraries. While universities increasingly provide digital content, the user experience can often be cumbersome. Many existing solutions either rely on external applications for PDF viewing, which disrupts the workflow, or offer basic viewers with limited search capabilities. Students and researchers spend considerable time sifting through documents to find specific information. A dedicated web-based e-library with an

integrated, searchable PDF viewer would streamline this process, enhancing productivity and promoting deeper engagement with academic materials. Furthermore, developing such a system using HTML, CSS, and JavaScript provides a valuable opportunity to apply core web development skills to a practical and impactful application.

1.2 Statement of Problems

Based on the findings gotten from research, the core problems this project would address are the inefficient and fragmented user experience within existing web-based e-libraries, primarily characterized by limited in-document search capabilities for PDF content and an over-reliance on external viewing solutions, as well as sub-optimal user interface design and non-responsive layouts in some e-libraries that hinder navigation and make content difficult to consume across various devices, further compounded by inaccessible PDF viewing.

1.3 Goal and Objectives

Goal: To design a user-friendly, web-based e-library system, with a primary focus on offering an integrated and highly effective searchable PDF viewer, thereby mitigating the architectural limitations of certain current systems.

Objectives:

-To develop a responsive and intuitive user interface for the e-library using HTML and CSS.

-To implement core e-library functionalities (Browse, categorization, global search) using JavaScript.

-To integrate a robust web-based PDF viewer capable of rendering PDF documents directly within the browser, taking advantage of JavaScript libraries.

-Implementing full-text search functionality directly within the integrated PDF viewer.

-To enable essential PDF viewer controls (text selection/copying, page navigation, zooming) with high level of ease and efficiency.

-To ensure efficient loading and rendering of PDF documents to provide a smooth user experience, even accommodating large documents.

1.4 Scope of Research

This project will focus more on the client-side (front-end) development of the e-library and its integrated searchable PDF viewer, demonstrating solutions to the identified architectural problems. The scope includes:

-Front-end Technologies: HTML for base structure, CSS for responsive design and visual presentation, and JavaScript for dynamic functionality, interactive elements, and API interactions.

-PDF Viewing Technology: Comprehensive utilization of an existing open-source JavaScript library (e.g., PDF.js) for robust PDF rendering and enabling the core in-document search functionality. This involves understanding the library's capabilities for text extraction and rendering.

-Core E-Library Features: Implementation of essential functionalities such as resource Browse, categorization (e.g., by subject, author), and a global search mechanism for library content (metadata-based).

-In-Document Search Implementation: The primary focus will be on the technical aspects of achieving efficient and accurate full-text search within the displayed PDF, including highlighting results.

-User Interface Design: Crafting a clean, intuitive, and responsive UI that prioritizes ease of navigation and interaction with both the library and the integrated viewer.

1.5 Research Methodology

The project will adopt an "Agile development methodology", involving the emphasis on iterative cycles, continuous feedback, and adaptive planning to address the complexities of integrating diverse functionalities. The key stages are:

-Literature Review: A comprehensive study of existing web-based e-library architectures, prevalent JavaScript PDF viewer libraries (e.g., PDF.js, react-pdf), and best practices for implementing efficient in-document search. This phase will specifically focus on identifying the architectural bottlenecks and limitations documented in current research.

-Requirements Analysis: Detailed elicitation and definition of functional and non-functional requirements, with a strong emphasis on user experience, search performance, and browser compatibility, directly derived from the identified problems.

-Architectural Design: Designing the client-side system architecture, including the modular breakdown of HTML, CSS, and JavaScript components, the data flow for content loading, and the integration strategy for the chosen PDF viewing library to ensure a seamless experience.

-Prototyping and Iteration: Rapid development of initial prototypes for both the e-library interface and the integrated PDF viewer, followed by iterative refinements based on usability testing and functional validation, directly responding to the goal of improving user experience.

-Implementation: Coding the e-library components using HTML5, CSS3, and modern JavaScript, and meticulously integrating and customizing the chosen JavaScript PDF viewer library to enable the core searchable functionality.

-Testing and Debugging: Rigorous testing of all functionalities, including global library search, PDF document rendering, in-document full-text search accuracy and performance, cross-browser compatibility, and responsiveness across various devices.

-Documentation: Comprehensive documentation of the project's code, architectural decisions, and a user guide, facilitating future understanding, maintenance, and potential expansion.

1.6 Research Significance

The development of this web-based e-library with a searchable PDF viewer holds significant academic and practical value by directly addressing critical limitations in existing e-library architectures:

-Enhancing User Experience and Productivity: By providing a unified interface and seamless in-document search, the project directly resolves the problem of fragmented experiences and inefficient information retrieval, significantly boosting user productivity for students and researchers.

-Reducing Dependence on External Software: The browser-native PDF viewing eliminates the need for downloads and external plugins, mitigating compatibility issues and improving accessibility for a wider range of users across devices.

-Demonstrating Practical Web Development Mastery: This project serves as a robust application of fundamental HTML, CSS, and JavaScript skills to solve real-world architectural problems, showcasing the ability to build a complex, interactive, and performant web application.

-Advancing Client-Side Document Interaction: It delves into the technical intricacies of client-side PDF rendering and text extraction for search, contributing practical insights into the capabilities and limitations of JavaScript libraries in this domain.

-Cost-Effectiveness and Open-Source Contribution: By leveraging open-source technologies, the project demonstrates a potentially more cost-effective approach to developing feature-rich e-library components, which could benefit institutions with limited resources.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

This chapter provides a comprehensive review of existing literature relevant to digital libraries, focusing on key areas such as their evolution and components, the challenges they face, and the advancements in in-document search and retrieval for PDF-based content.

2.2 Evolution and Components of Digital Libraries

The concept of digital libraries originated in the late 20th century, brought about by the emergence of networked computing and the increasing volume of information available in digital formats. Early definitions, such as those by Arms (1995), emphasized collections of information stored electronically and accessible over networks. Subsequent conceptualizations have broadened, not just focusing on collections but also the services, infrastructure, and user communities that interact with them (Borgman, 1999).

The basic components of a digital library as explained byRicha Pandey (2003), include the following;

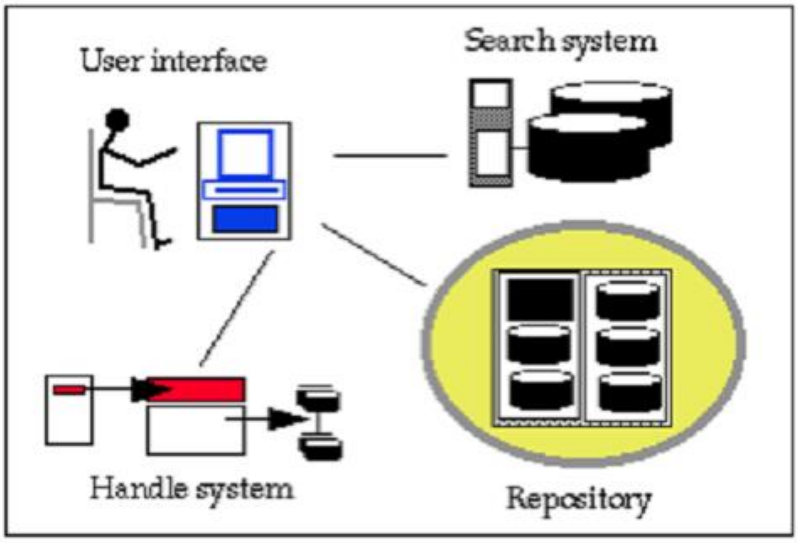

Figure 2.1 Basic components of a digital library.(Richa Pandey, 2003)

2.2.1. User Interfaces

As seen in figure 2.1, there are two user interfaces used here: the first is meant for the end-users of the digital library, while the other is for digital librarians and system administrators who manage the collections. A standard Internet browser would be used for the actual interactions between the user and the library. Examples of such could be Netscape Navigator, Microsoft's Internet Explorer, or Microsoft Edge. The browser connects to client services, which provide intermediary functions between the browser and the other partsof the system. The client services allow the user to decide where to search and what to retrieve; they interpret information structured as digital objects; they negotiate terms andconditions, manage relationships between digital objects, remember the state of theinteraction, and convert among the protocols used by the various parts of the system.

2.2.2. Repository

Also as seen in figure 2.1, repositories are meant to store and manage digital objects and other information. A large digital library may have different types of repositories, including modern repositories, legacy databases, and Web servers. The interface to these repositories is called the repository accessprotocol (RAP). Main features of RAP are explicit recognition of rights and permissions that needto be satisfied before a client can access digital objects, support for a very general range of dissemination of digital objects, and an open architecture with properly defined interfaces.

2.2.3. Handle System

Another basic component of libraries shown in figure 2.1 would be the ‘handles’. Handles are general-purpose identifiers that can be used to identify certain internet resources, such as digital objects over long periods of time and to manage materials stored in any databaseor repository. When these are used with the repository, the handle system receives as input a handle for a digital object and returns the identifier of the repository where the object is stored.

2.2.4. Search System

The last component also as shown in the figure above is the ‘search system’. The basic design of the digital library system assumes that there will always be many indexes and catalogsthat can be searched to discover data or information before retrieving it from a repository. These indexes may bemanaged independentlyand support a wide range of protocols.

2.3 Key Features of a Modern Digital Library

2.3.1 Content Acquisition and Organization: This involves digitizing physical materials, acquiring born-digital resources, and organizing them using metadata standards (e.g., Dublin Core, MARC21) to ensure discoverability (Salokhe et al., 2011).

2.3.2 Storage and Preservation: Providing secure and reliable storage solutions (e.g., cloud storage, distributed systems) are vital, along with strategies for long-term digital preservation to counteract technological obsolescence (Lynch, 1998).

2.3.3 Information Retrieval Systems: Robust search engines and browsing interfaces enable users to locate desired information efficiently. This often involves full-text indexing, faceted search, and relevance ranking algorithms.

2.3.4 User Interface and Access: Modern web-based interfaces are intuitive, and as such provide access to the digital collection, often incorporating features like user accounts, personalization, and interactive viewers.

2.3.5 Interoperability: The ability of different digital libraries or information systems to exchange data and services, often facilitated by protocols like OAI-PMH (Open Archives Initiative Protocol for Metadata Harvesting) (Lagoze& Van de Sompel, 2001).

The shift from physical to digital has introduced new challenges, including copyright management, ensuring integrity of digital objects, and bridging the digital divide to ensure equitable access (Gaur & Rani, 2012). Despite these challenges, digital libraries offer undeniable advantages in terms of accessibility, scalability, and the potential for enhanced discovery.

2.4 Challenges and Opportunities in PDF Document Management

PDF (Portable Document Format) has become a well known standard for document archiving and exchange due to its ability to preserve document formatting across various platforms. Its widespread adoption in academic, legal, and governmental sectors makes it a critical format for digital libraries (Adobe, 2008). However, managing and searching within PDF documents presents unique challenges.

Traditional search engines often rely on extracting plain text from PDFs, which can sometimes lead to loss of formatting context, misinterpretation of tables, or difficulties with scanned documents (image-based PDFs) that require Optical Character Recognition (OCR) (Choudhary et al., 2017). Furthermore, simply finding a document is often not enough; users frequently need to pinpoint specific information within the document itself. This necessitates effective in-document search capabilities.

Opportunities within PDF document management lie in leveraging advanced techniques for text extraction, indexing, and visualization. Modern libraries like PDF.js (Mozilla Foundation, 2024), for instance, enable client-side rendering of PDFs in web browsers, opening doors for rich interactive experiences and direct manipulation of document content, including text selection and highlighting.

2.5 Existing Search and Retrieval Systems in Digital Libraries

Existing digital libraries and document management systems today make use of various strategies for search and retrieval. These can broadly be categorized by their approach to content and their level of granularity in search.

2.5.1 Metadata-Based Search

Many early and even contemporary systems primarily rely on metadata for searching. Users search using fields like "title," "subject," "author," or "publication date" (Tenopir et al., 2014). While effective for known-item searches and broad topic exploration, metadata search is limited by the quality and granularity of the metadata itself. It cannot pinpoint information within a document if that information is not explicitly captured in its descriptive metadata.

2.5.2 Full-Text Search

The advent of full-text indexing allowed for keyword searching across the entire textual content of documents. Systems like Google Scholar, academic databases (e.g., JSTOR, ScienceDirect), and institutional repositories utilize this approach extensively. When applied to PDFs, this typically involves server-side text extraction (often using tools like Apache Tika or Poppler) during the indexing process, converting the PDF into searchable plain text (Shaker et al., 2013). While powerful for document discovery, the results of such searches usually return the entire document, leaving the user to manually find the relevant section within the retrieved PDF.

2.5.3 In-Document Search and Highlighting

A more advanced feature, crucial for enhanced user experience, is the ability to search within a document once it is opened in a viewer and to highlight the search terms. Desktop PDF readers (e.g., Adobe Acrobat Reader, Foxit Reader) have long offered this functionality. However, replicating this experience seamlessly in a web browser without requiring plugin installations has been a more recent development.

Early web-based solutions often relied on proprietary plugins or server-side rendering of individual PDF pages as images, which made text selection and highlighting difficult or impossible (e.g., embedding Google Docs Viewer). Modern approaches leverage client-side JavaScript libraries. Technologies like Mozilla's PDF.js have revolutionized this by parsing and rendering PDFs directly in the browser's HTML5 canvas (Mozilla Foundation, 2024). This client-side rendering capability allows for programmatic access to the PDF's text layer, enabling in-document text search and dynamic highlighting of keywords without server interaction for each search query. This significantly reduces server load and improves responsiveness for the user.

Research by Miron et al. (2014) on the implementation of a web-based PDF viewer demonstrates the technical feasibility and user benefits of integrating such client-side capabilities, highlighting the importance of efficient text extraction and rendering performance for a smooth user experience.

2.6 Text Extraction and Indexing for PDF Documents

Accurate text extraction is fundamental for any full-text search capability. For PDF documents, this is not always straightforward due to their complex internal structure. PDFs can contain text, images, vectors, and fonts, and the textual content might not always be stored in a linear, easily readable order (Adobe, 2008).

Techniques for text extraction from PDFs include:

Rule-based Parsing: Directly interpreting the PDF's internal syntax to extract text strings and their positions. Libraries like PDF.js perform this client-side.

Optical Character Recognition (OCR): For scanned PDFs or image-based documents, OCR software is used to convert images of text into machine-readable text. While powerful, OCR introduces potential for errors depending on image quality (Smith, 2007).

Commercial/Proprietary Tools: Many commercial PDF processors offer highly optimized text extraction with varying degrees of accuracy and feature sets.

Once text is extracted, it needs to be indexed to facilitate quick searching. Indexing involves creating data structures (e.g., inverted indexes) that map keywords to their locations within documents. For in-document search, the index not only needs to store the page number but also the precise character offset or bounding box coordinates for accurate highlighting (Baeza-Yates &Ribeiro-Neto, 2011). Client-side indexing for individual PDFs as seen with PDF.js can leverage the extracted text content in memory to perform rapid searches.

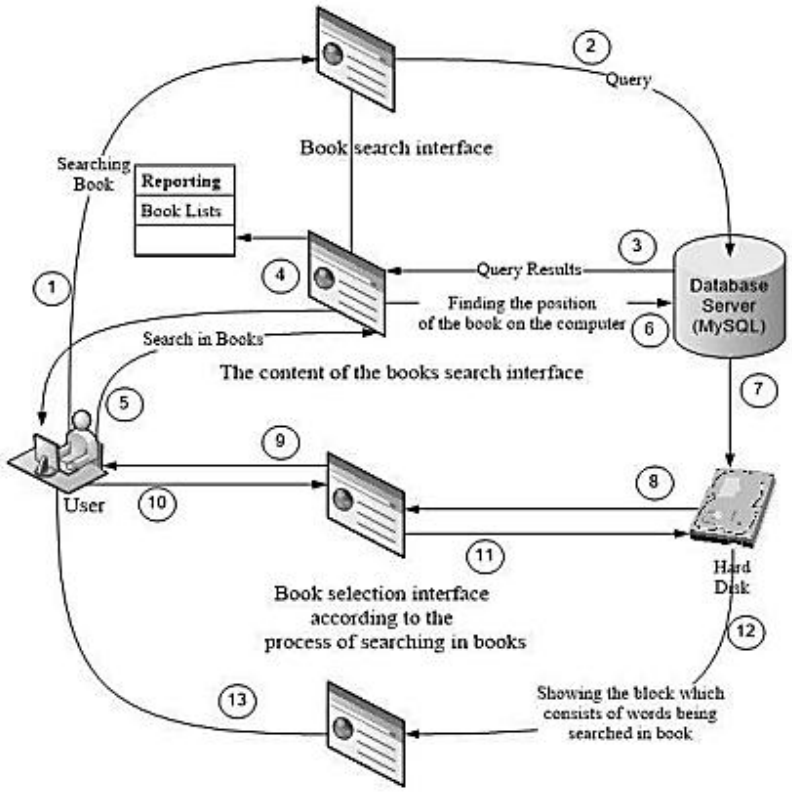

Figure 2.2 Example diagram for showing a part of a PDF file within a web browser (Atalar,2021).

2.7 Agent Architecture for a Virtual Digital Library

As described by Birmingham, (1995), this is an example of a real life E-library system which makes use of different software agents to bridge the gap between users, authors and publishers. This architecture is agent based, meaning its functions, such as search, retrieval, publishing and coordination; are handled by specialized software agents that interact autonomously.

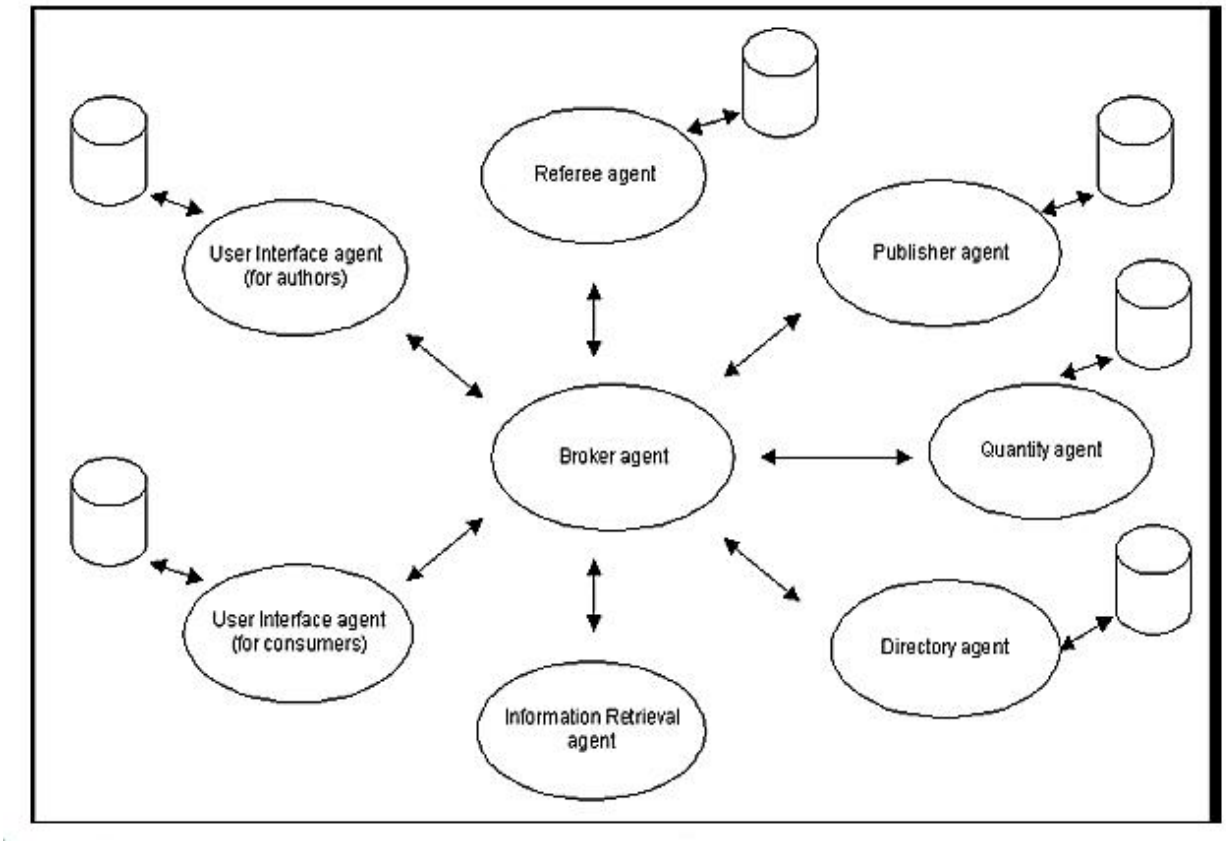

Figure 2.3 Diagram of the Agent Architecture for a Virtual Digital Library (Birmingham, 1995).

This type of architecture seen in figure 2.3 was heavily influenced by:

-Early Distributed Digital Library research projects.

-The National Science Digital Library (NSDL) in the USA.

-Work by Fox, Suleman, Kahn, Cerf, and others at Virginia Tech Digital Library Research Laboratory (DLRL).

-European projects like ERCIM (European Research Consortium for Informatics and Mathematics).

-Many papers in D-Lib Magazine (late 1990s–early 2000s).

2.7.1 Types of Agents

I.User Interface Agents (Consumers)

-Help end-users formulate queries.

-Manage interaction with the system.

-Personalize services.

II.User Interface Agents (Authors)

-Help authors submit and manage digital content.

III. Publisher Agents

-Manage publishing workflows, copyright, licensing.

IV. Information Retrieval Agents

-Search distributed indexes and repositories.

-Aggregate and filter results.

V. Broker Agents

-Mediate between information providers and consumers.

-Coordinate multiple retrieval agents.

-Optimize query routing.

VI. Directory Agents

-Maintain catalogs of available resources and their metadata.

VII. Quantity Agents

-Track usage metrics.

-Handle billing or quotas.

VIII. Referee Agents

-Enforce policies, permissions.

-Resolve conflicts about access or duplication.

2.7.2 How it Works

This architecture operates with the following steps,

-When users submit a query, the User Interface Agent helps format it.

-The Broker Agent sends it to one or more Information Retrieval Agents.

-The Directory Agent helps find where relevant resources are.

-Publisher Agents ensure rights are respected if content is published or exported.

-Quantity and Referee Agents check usage, apply any restrictions, and log transactions.

2.7.3 Examples in Practice

-Analytics trackers (Google Analytics) play a quantity agent role.

-Search crawlers & bots (Googlebot, Bingbot) are retrieval agents.

-Recommendation engines (Netflix, YouTube) act like user interface and/or broker agents.

-DRM & rights-checking layers are like publisher & referee agents.

2.8 Conclusion

The literature review highlights impact of digital libraries and the evolving landscape of information retrieval. While metadata and full-text search at the document level are well-established, the demand for seamless, in-document search and highlighting within web-based PDF viewers represents a critical enhancement for user experience. The insights gained from this review will directly inform the system analysis and design of the proposed web-based e-library, ensuring that it leverages sufficient modern capabilities to address the practical needs of its users.

CHAPTER THREE

SYSTEM ANALYSIS AND DESIGN

3.1 Introduction

This chapter discusses the analysis and the subsequent design of the proposed web-based e-library system. Building upon the insights from the literature review regarding the evolution of digital libraries, the challenges of PDF document management, and the capabilities of modern in-document search technologies, this section will outline the functionality, architecture, and user interface of the proposed solution. It will also discuss the tools and technologies required for its implementation, ensuring alignment with the project's objectives of providing an efficient and user-friendly platform for accessing and searching digital documents.

3.2 Existing Systems Analysis

As highlighted in Chapter 2, various digital library systems and document viewers currently exist. These range from large-scale academic databases and repositories to smaller scale desktop PDF readers.A major example of this would be the “Agent Based Architecture for a Virtual Digital Library”. While these systems offer valuable functionalities, some of them often present certain limitations that the proposed system aims to address:

3.2.1 Inter-Agent Communication Bottlenecks: In a highly interactive e-library, frequent communication between separate services (like document retrieval, metadata extraction, search query processing) can indeed create bottlenecks.

3.2.2 Lack of Central Control: While agents are autonomous, a complete lack of central management or oversight can lead to inconsistencies, unmanageable resource consumption, or rogue behaviors.

3.2.3 User Experience Gaps: Some existing platforms may suffer from cluttered interfaces, inconsistent navigation, or lack of responsiveness, which hinders efficient information retrieval.

3.2.4 Complexity and Overhead: Lack of interoperability or unified viewing experiences means users might navigate disparate systems to find and then lead to systems getting too complex and hard to maintain.

The analysis of these existing systems reveals the need for a lightweight, entirely browser-based e-library that not only efficiently manages a collection of PDF documents but, more importantly, provides a highly interactive and responsive in-document search and highlighting feature without relying on external software installations.

3.3 Overview of the Proposed System

The proposed system is a Web-Based E-Library with Searchable PDF Viewer. It aims to provide an intuitive platform for users to browse, search, and view digital documents, primarily in PDF format, all while being centralized. The core innovation lies in its seamless, client-side in-document search capability, offering a user experience similar to desktop PDF readers directly within a web browser.

This system aims to;

-To provide an intuitive web interface for browsing and searching digital documents.

-To enable efficient client-side rendering of PDF documents directly within the browser using HTML5 Canvas.

-To implement robust in-document full-text search with dynamic highlighting and navigation of search results.

-To ensure a responsive and user-friendly design across various devices.

3.4 System Architecture

The proposed system adopts a client-side centric architecture leveraging modern web technologies. This architecture minimizes server load for document viewing and in-document search operations, making the application highly responsive and scalable in terms of user concurrent access to viewer functionalities.

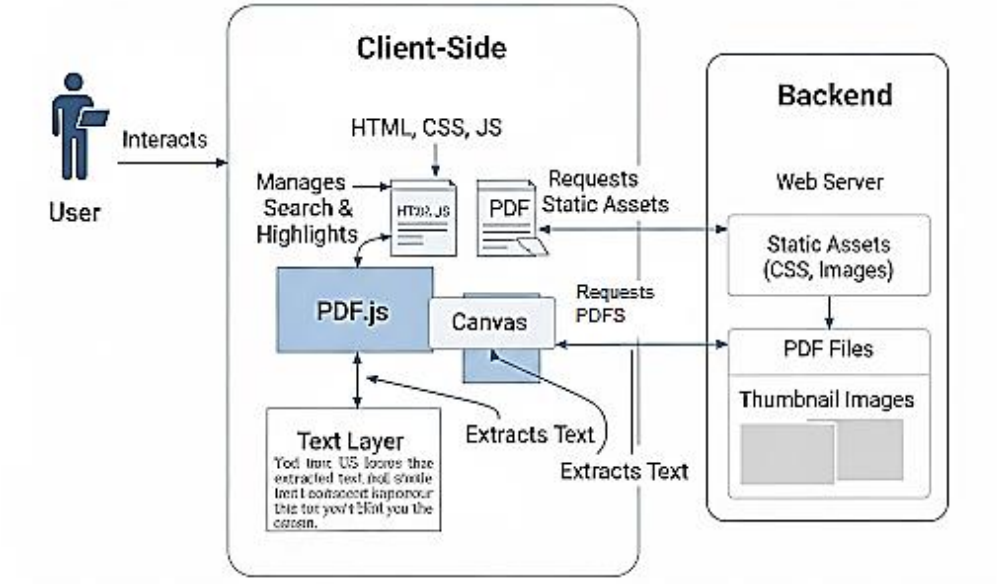

Figure 3.1 System Architecture Diagram

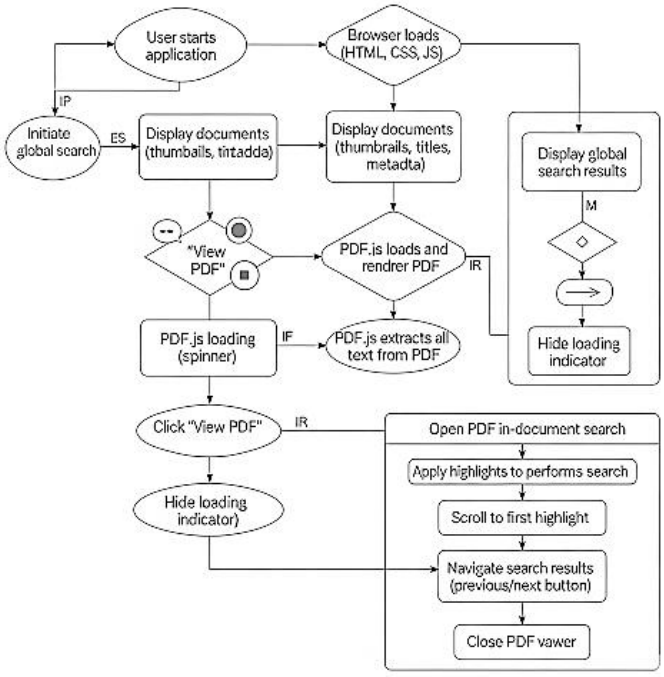

Figure 3.2: Flowchart diagram for the workflow of the proposed system.

3.4.1 Architectural Components

I.Client-Side (User's Browser): This is where the core application logic resides and executes.

II. HTML & CSS: Define the structure and visual presentation of the e-library interface, including the document cards, search bars, and the PDF viewer modal.

III. JavaScript Application Logic:

-Manages the dynamic behavior of the UI (e.g., showing/hiding modals, updating page numbers).

-Handles global library search (filtering document metadata).

-Coordinates with PDF.js for PDF rendering.

-Implements the in-document search functionality, operating directly on text extracted by PDF.js.

-Manages application state (e.g., current document, current page, zoom level, search results).

-Provides user feedback (loading, errors).

IV. PDF.js Library: Useful for,

-Parsing PDF binary data.

-Renders PDF pages to an HTML5 <canvas> element.

-Provides an API to extract text content and layout information from PDF pages. Internally uses Web Workers for performance.

V. HTML5 Canvas: The drawing surface where PDF.js renders the visual representation of PDF pages.

VI. TextLayer Overlay:A transparent HTML div element positioned precisely over the canvas. PDF.js helps populate this layer with selectable text elements, and our JavaScript inserts highlight s into these elements during in-document search.

VII. Backend (Minimal Static File Server): A very simple server that only needs to serve static files. It does not contain any complex application logic, databases, or runtime processing related to PDFs or search.

-Web Server: Any basic web server (e.g., Nginx, Apache, Node.js http-server, Python http.server) is sufficient.

-Static Assets: All HTML, CSS, and JavaScript files that constitute the client-side application.

-PDF Document Files: The actual PDF documents of the e-library collection.

-Thumbnail Images: Images for document cards.

3.4.2 Data Flow:

I. The user's browser requests index.html from the Web Server.

II. The Web Server responds with index.html, style.css, and script.js (which also loads PDF.js from a CDN).

III. The JavaScript Application Logic initializes, reads libraryDocuments (either embedded in JS or fetched from a simple JSON file also served by the Web Server), and displays document cards.

IV. When a user clicks "View PDF":

-The JavaScript Application Logic requests the specified PDF file (URL) from the Web Server.

-The Web Server serves the PDF file.

-PDF.js loads the PDF, renders the current page to the HTML5 Canvas, and extracts all page text into memory for the JavaScript Application Logic

V. When a user performs an in-document search:

-The JavaScript Application Logic performs the search directly on the in-memory text extracted by PDF.js.

-The Text Layer Overlay is manipulated by the JavaScript Application Logic to show highlights. No communication with the backend is needed for this operation.

3.5 System Interface Design

The user interface (UI) is designed to be intuitive and highly responsive, leveraging the client-side processing power. The design principles remain focused on user-friendliness and clear feedback, now with the understanding that responsiveness is directly tied to efficient client-side JavaScript execution.

3.6 Tools and Technologies

The development of this web-based e-library system requires the following tools and technologies:

-HTML5: For structuring the content of the web pages.

-CSS3: For styling the user interface, ensuring a modern and responsive design.

-JavaScript (ES6+): The primary language for implementing all interactive functionalities, including DOM manipulation, event handling, and custom logic.

-PDF.js Library (Mozilla Foundation): The core JavaScript library for rendering PDF documents within the browser. We leverage its CDN for easy inclusion.

-Web Browser: Any modern web browser (e.g., Chrome, Firefox, Edge,) that supports HTML5, CSS3, ES6+, and Web Workers.

-Text Editor / IDE: A code editor like Visual Studio Code for writing and managing the source code.

3.7 Use Case Diagram

This is a diagram used to highlight how the system functions and how it interacts with the users of said system. This is done using “Actors” and “use cases”.

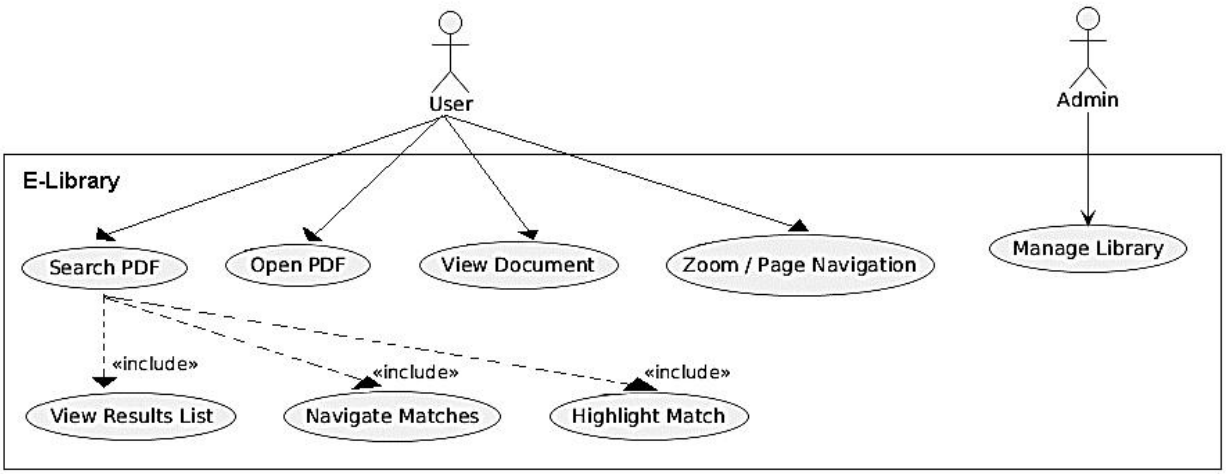

Figure 3.3 Use case diagram for the proposed system.

Actors 	Use Cases 
User Represents the end-user (student, researcher, reader). Primary goal is to open, read, and search within PDF documents.	**(User interactions)** **Open PDF** Load a document from the user's device into the system. **View Document** Render the PDF file in the browser for reading. **Search PDF** Enter keywords/phrases to search across all pages. This includes: -Highlight Match (visually mark results inside the PDF). -Navigate Matches (move to next/previous match). -View Results List (optional search result panel with page references). **Zoom / Page Navigation** Adjust view scale, move between pages.
Admin Responsible for managing the overall e-library (adding or removing documents). Less frequent interaction compared to normal users.	**Use Cases (Admin interactions)** Manage Library Upload, organize, or remove documents. Keep the digital library clean and accessible.

Table 3.1 Table describing the contents of the use case diagram in fig.3.3.

3.8 Class Diagram

This provides a visual representation of the architecture of a system, what it contains, and the relationships between said contents of the system.

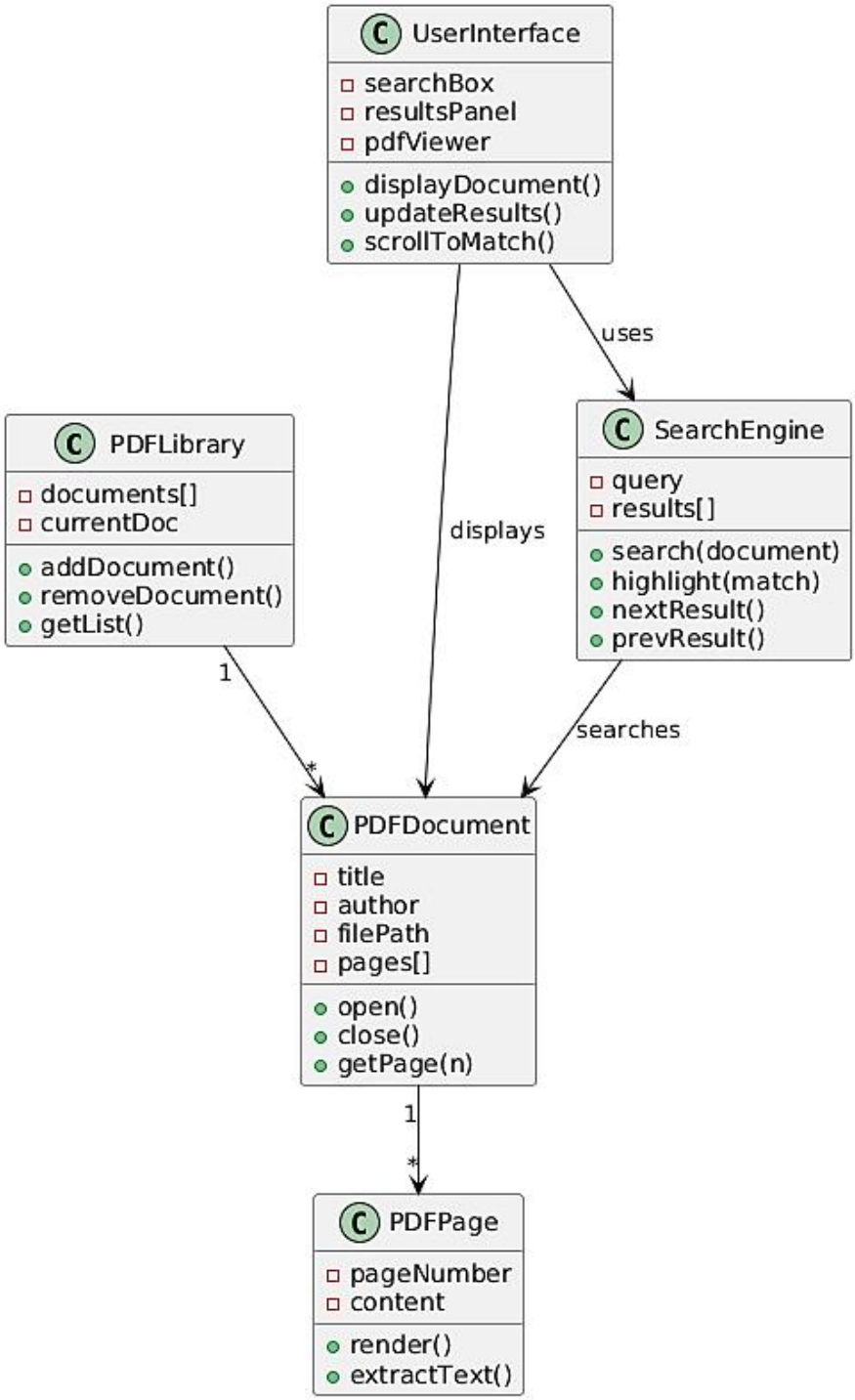

Figure 3.4 Class diagram for the proposed system.

Class	Type	Name	Description
PDF Library	Attribute	documents[]	Collection of PDFDocument objects stored in the library.
	Attribute	currentDoc	The currently opened PDFDocument.
	Method	addDocument()	Adds a new document to the library.
	Method	removeDocument()	Removes a document from the library.
	Method	getList()	Returns the list of all documents in the library.
PDFDocument	Attribute	title	Title of the document (metadata).
	Attribute	author	Author of the document (metadata).
	Attribute	filePath	Location/path/URL of the document.
	Attribute	pages[]	Collection of PDFPage objects representing each page.
	Method	open()	Loads the document into the viewer.
	Method	close()	Closes the document and releases resources.
	Method	getPage(n)	Retrieves a specific page by number.
PDFPage	Attribute	pageNumber	The index/number of the page in the document.
	Attribute	content	The text or graphical content of the page.
	Method	render()	Renders the page visually in the viewer.
	Method	extractText()	Extracts the text layer from the page for searching.
SearchEngine	Attribute	query	The current search keyword or phrase.
	Attribute	results[]	List of matches found (with page numbers and positions).
	Method	search(document)	Executes a keyword search across the document.
	Method	highlight(match)	Highlights a specific match in the viewer.
	Method	nextResult()	Moves to the next search result.
	Method	prevResult()	Moves to the previous search result.
UserInterface	Attribute	searchBox	Input field where the user enters a search query.
	Attribute	resultsPanel	Area displaying search results and their page numbers.
	Attribute	pdfViewer	The visual component where the PDF is displayed.
	Method	displayDocument()	Shows the currently opened document.
	Method	updateResults()	Updates and displays the list of search results.
	Method	scrollToMatch()	Navigates and scrolls directly to a search match inside the PDF.

Table 3.2 Table showing the contents of the class diagram in fig.3.4.

3.8.1 Relationships

Based on the diagram in figure 3.4 the relationships between the different classes are as follows;

-PDFLibrary → PDFDocument (1..*)

A library contains multiple documents.

-PDFDocument → PDFPage (1..*)

Each PDF is composed of many pages.

-SearchEngine → PDFDocument

The search engine operates on loaded documents.

-UserInterface → PDFDocument

Displays the document for user reading.

-UserInterface → SearchEngine

Uses search engine services to display matches and navigate.

CHAPTER FOUR

SYSTEM IMPLEMENTATION

4.1 Implementation Tools

This chapter focuses on a review of the tools and technologies used in the design and implementation of the searchable pdf viewer e-library.

4.1.1 Implementation Languages

The project was created using the following fundamental languages:

-HTML (HyperText Markup Language): This provides the structural foundation of the web page. It defines all the elements a user sees, such as the library sidebar, the main viewer, and the toolbar.

-CSS (Cascading Style Sheets): This language is used for the visual presentation and layout. The CSS functions to style the entire user interface, including the colors, fonts, spacing, and the responsive layout (e.g., the flexbox model).

-JavaScript: This is the main scripting language that provides all the dynamic functionality. It handles file I/O (reading local PDFs), manages the PDF rendering process, implements the search functionality, and controls user interactions such as button clicks and drag-and-drop events.

4.1.2 Implementation Framework

The system directly works with the Document Object Model (DOM) to update the user interface, and does not overly rely on a complex, full-fledged JavaScript framework like Angular, or Vue.js. It does however make some use of React.js for certain features.This choice makes the code lightweight and self-contained, but it could become more difficult to manage for larger, more complex applications.

-PDF.js: The core functionality of this project depends on this framework. PDF.js is the essential open-source library from Mozilla that enables the rendering of PDF files within a web browser. The code utilizes two main files from this library: pdf.js (the main library) and pdf.worker.js (a web worker that performs heavy-duty PDF parsing in a separate thread to prevent the UI from freezing). Without PDF.js, the core purpose of the application would’nt be achieved.

4.1.3 Implementation Platforms

The project is built to work as a web application. Its platform is the web browser itself. This means it is designed to run on any modern web browser that supports HTML5, CSS3, and JavaScript, including:

-Google Chrome

-Mozilla Firefox

-Microsoft Edge

-Safari

This platform is essential in making the application highly portable and universally accessible without the need for a separate installation process, or use of memory space.

4.1.4 Deployment Platforms

The deployment for this project is entirely client-side. This application does not require a server. It can be deployed in a few simple ways:

-Local File System: A user can simply open the index.html file directly from their computer, and the application will run. This is the simplest form of deployment.

-Static Web Server: The project files (HTML, CSS, JavaScript, and the pdfjs folder) can be hosted on any simple web server (e.g., Apache, Nginx, or a cloud service like GitHub Pages). The user accesses the application via a URL, but all processing still happens on their local machine.

The offline nature of the application means that even after the initial loading from a server, it can function without an internet connection, provided the user has previously loaded the necessary files.

4.1.5 Operating System

Since the application runs within a web browser, it is platform-independent at the operating system level. It will work virtually identically on:

-Windows

-MacOS

-Linux

-Mobile operating systems like Android and iOS

The only major requirement is a compatible web browser.

4.2 User Documentation & System Testing

Certain elements were looked into during the process of creating the project. To ensure functionality, these elements had to be tested. They include:

-Login Page: This was created as a way to make students have safe authorized access to the e-library for use. It allows for creating an account or logging in to an already created account using a student's email address. This can be seen in Fig. 4.1.

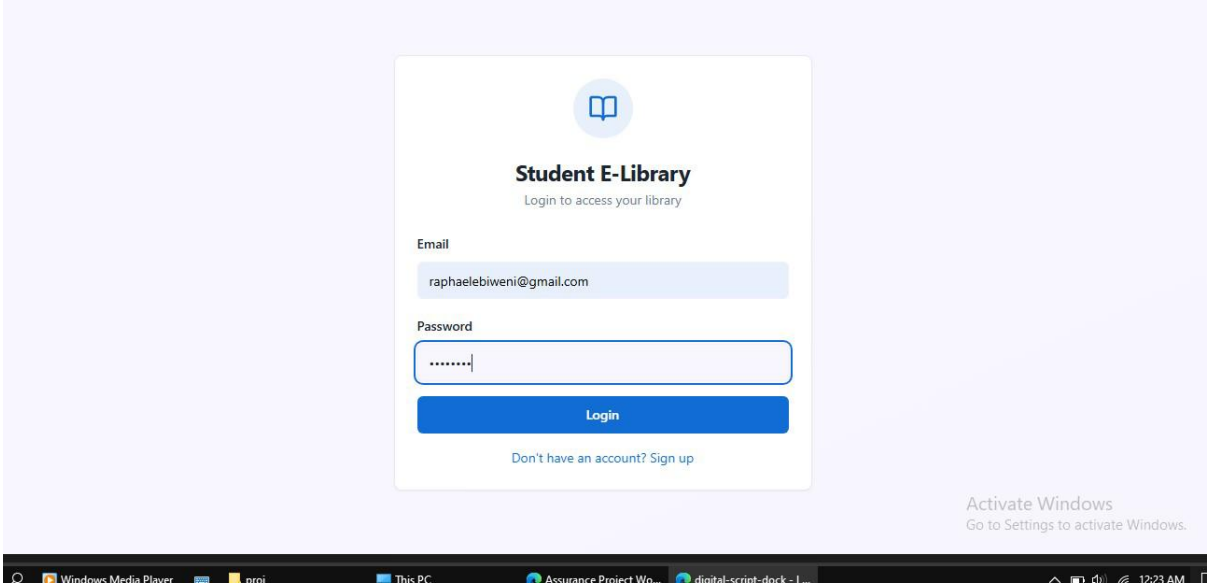

Figure 4.1 Screenshot showing the login page

-Main Menu: This would be what the user sees after logging into the e-library using their student e-mail address. It is here students can choose to use the library or check their login data. This can be seen in Fig. 4.2.

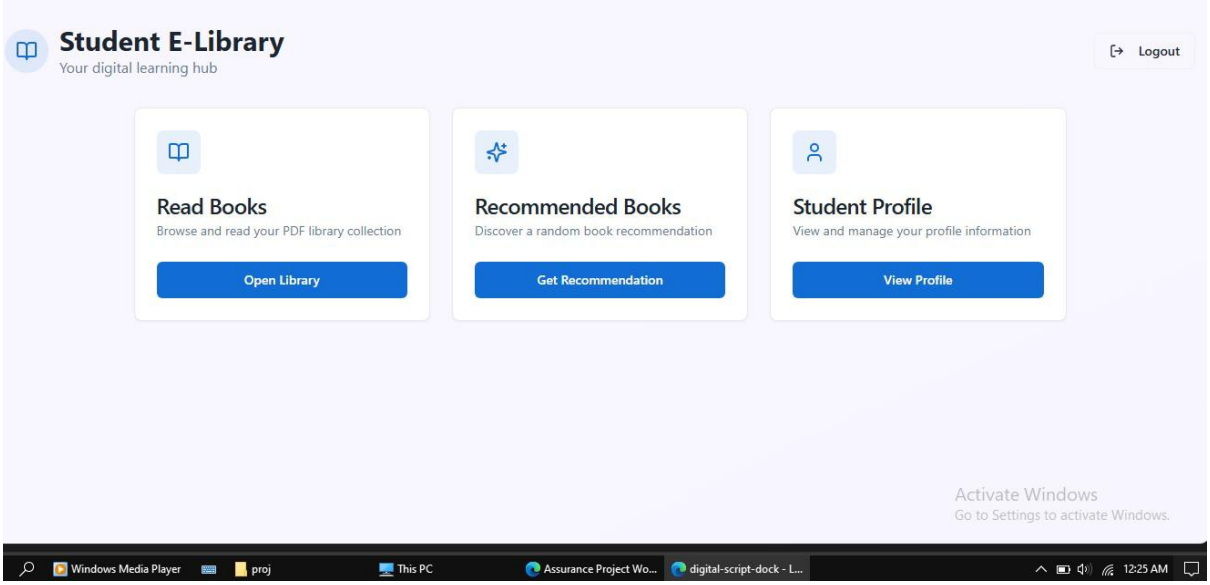

Figure 4.2 Screenshot showing the main start menu

-Adding PDFs: This was implemented by javascript to implement drag and drop, and was checked to see how well articles could be added to the library. This can be seen in Fig. 4.3.

Figure 4.3 Screenshot for the viewing of multiple PDFs section.

-Selecting Through Multiple Documents: As also seen in fig.4.3, the ability to look through, select and search from multiple pdf titles was checked and implemented.

-Search Functionality: The core aspect of this project, the ability to search through PDFs was also implemented using javascript and tested as well to see if it was working properly. It works visually by taking snippets of different areas where the selected text can be found, and then taking the viewer to said page once clicked. This can be seen in Fig. 4.4.

Figure 4.4 Screenshot for displaying the search functionality.

-Page Viewing: This entailed checking if the pdf viewer would display and order the pages of a pdf would be shown in the proper order.

4.3 System Usability Evaluation

The current system's usability can be evaluated against a few key criteria:

Efficiency: The system is highly efficient for its intended purpose. All processing is local, eliminating server latency. The use of a web worker for PDF rendering prevents the browser from becoming unresponsive. The search function, while potentially slow on very large files, provides quick access to relevant snippets.

Learnability: The user interface is straightforward. Drag-and-drop functionality is a familiar pattern, and the toolbar buttons are standard. A new user could likely figure out how to use the core features without instruction.

Memorability: Since the design is simple and uses standard web UI elements, a user who has used the application once would easily remember how to use it again.

Error Prevention and Handling: The system provides basic error messages (e.g., "Failed to open PDF"). It also prevents users from navigating to invalid pages. A key area for improvement would be providing better feedback during the search process, perhaps a progress bar for very long documents.

Satisfaction: The minimal, clean design and the ability to work completely offline are strong points that contribute to user satisfaction, especially for those who need a portable, self-contained e-library solution.

CHAPTER FIVE

SUMMARY AND CONCLUSION

5.1 Summary

This project set out to design a simple web-based e-library with an integrated searchable PDF viewer to address the limitations of certain existing digital library systems. The motivation for this project was driven by the need to enhance accessibility, usability, and efficiency in retrieving information within digital academic environments.

Looking at the insights seen in the literature review, the system analysis and design phase defined the shortcomings of certain current systems and proposed a lightweight, browser-based architecture. This architecture emphasized client-side functionality, responsive interface design, and in-document search capabilities. The design incorporated essential diagrams such as use case, class, and architecture diagrams to ensure clarity in functionality and system relationships.

The implementation stage made use of fundamental web technologies (HTML, CSS, JavaScript) along with PDF.js as the core library for rendering and searching within PDF documents. Repeated testing made sure that key functionalities such as drag-and-drop PDF upload, document viewing, in-document search, and efficient navigation were successfully achieved. The usability evaluation further revealed the system's efficiency, learnability, memorability, and user satisfaction, while also identifying potential improvements such as enhanced search feedback for larger files.

5.2 Conclusion

The successful development of the proposed web-based e-library with a searchable PDF viewer demonstrates that it is possible to overcome the limitations of traditional and existing digital libraries by making use of lightweight, client-side technologies. By eliminating reliance on external viewing applications, the project provides a unified, browser-centric solution that significantly improves the user experience for students, researchers, and academic institutions.

The project's contribution lies not only in its functional implementation but also in showcasing how fundamental web technologies can be combined with open-source libraries to create efficient, user-friendly, and scalable digital systems. Its platform independence further enhances its accessibility, thereby making it a versatile solution across diverse learning contexts.

In conclusion, the project successfully fulfills its goal of providing a functional, accessible, and practical e-library platform, thereby contributing to improved information retrieval, learning, and research efficiency in this current digital era.

References

Adobe Systems Incorporated. (2008). PDF Reference, Sixth Edition: Adobe Portable Document Format Version 1.7. Addison-Wesley.

Arms, W. Y. (1995). Key concepts in the architecture of the digital library. D-Lib Magazine.

Atalar, A. (2021). Example diagram for showing a part of a PDF file within a web browser.

Baeza-Yates, R., &Ribeiro-Neto, B. (2011). Modern information retrieval: The concepts and technology behind search (2nd ed.). Addison-Wesley.

Birmingham, W. P. (1995). Agent architecture for a virtual digital library.

Borgman, C. L. (1999). What are digital libraries? Competing visions. Information Processing & Management, 35(3), 227–243.

Choudhary, D., Vidhani, M., & Patel, A. (2017). Text extraction from portable document format (PDF) files. International Journal of Computer Applications, 170(3), 32–37.

Fox, E., Suleman, H., Kahn, R., Cerf, V., & others. (1990s–2000s). Digital Library Research Laboratory (DLRL) contributions. Virginia Tech.

Gaur, R. R., & Rani, R. (2012). Digital libraries in knowledge management: Opportunities and challenges. Library Philosophy and Practice.

Hasbrouck, J. (2020). Optical character recognition in digital libraries.

Lagoze, C., & Van de Sompel, H. (2001). The Open Archives Initiative: Building a low-barrier interoperability framework. Proceedings of the First ACM/IEEE-CS Joint Conference on Digital Libraries, 54–62.

Lynch, C. A. (1998). The integrity of digital information: Mechanics and definitions. Journal of the American Society for Information Science, 49(7), 580–589.

Miron, M., Rusu, D., &Rus, L. (2014). Implementation of a web-based PDF viewer using HTML5 and JavaScript. International Journal of Computer Applications, 94(2), 1–7.

Mozilla Foundation. (2024). PDF.js: Portable Document Format rendering in JavaScript. https://mozilla.github.io/pdf.js

Pacific, J. (1977). Background study of libraries and information access.

Pandey, R. (2003). Components of a digital library.

Salokhe, G., Keizer, J., Katz, S., &Bekiari, C. (2011). Metadata and interoperability in agricultural information systems. Agricultural Information Worldwide, 4(1), 14–19.

Shaker, K., Tunkelang, D., & others. (2013). Server-side text extraction for digital repositories. Information Retrieval Journal, 16(3), 245–263.

Smith, R. (2007). An overview of the Tesseract OCR engine. Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR), 629–633.

Tenopir, C., King, D. W., Edwards, S., & Wu, L. (2014). Electronic journals and changes in scholarly article seeking and reading patterns. Aslib Journal of Information Management, 66(1), 5–24.

APPENDIX

FEATURE SOURCE CODES

1. **File upload and document ingestion**

-Library Header (file input and handler)

```
// src/components/PDFLibrary/LibraryHeader.tsx
// handleFileChange calls the parent onFileUpload(files)
consthandleFileChange = (e: React.ChangeEvent<HTMLInputElement>) => {
if (e.target.files&&e.target.files.length> 0) {
onFileUpload(e.target.files);
  }
};
...
<label htmlFor="pdf-upload">
<Button variant="outline" className="cursor-pointer" asChild>
<span>
<Upload className="h-4 w-4 mr-2" />
     Add PDFs
</span>
</Button>
</label>
<input
id="pdf-upload"
type="file"
accept=".pdf"
multiple
onChange={handleFileChange}
className="hidden"
/>
```

-Handling file upload

```
// src/pages/Index.tsx
consthandleFileUpload = async (files: FileList) => {
setIsLoading(true);
constnewDocuments: PDFDocument[] = [];

try {
for (let i = 0; i<files.length; i++) {
const file = files[i];
if (file.type === 'application/pdf') {
try {
constpdfDoc = await PDFUtils.loadPDF(file);
newDocuments.push(pdfDoc);
       } catch (error) {
console.error(`Error processing ${file.name}:`, error);
toast({ title: "Error", description: `Failed to process ${file.name}` });
```

```
      }
    }
  }

if (newDocuments.length> 0) {
setDocuments(prev => [...prev, ...newDocuments]);
toast({
title: "Success",
description: `Added ${newDocuments.length} document${newDocuments.length !==
1 ? 's' : ''} to your library`,
    });
  }
 } catch (error) {
console.error('Error uploading files:', error);
toast({ title: "Error", description: "Failed to upload files", variant: "destructive" });
 } finally {
setIsLoading(false);
 }
};
```

-PDF.js text extraction and thumbnail extraction

```
// src/utils/pdfUtils.ts
staticasyncloadPDF(file: File): Promise<PDFDocument> {
constarrayBuffer = await file.arrayBuffer();
const pdf = await pdfjsLib.getDocument({ data: arrayBuffer }).promise;

 // Extract metadata
const metadata = await pdf.getMetadata();
const info = metadata.info as any;

 // Extract text from all pages
letextractedText = '';
for (let i = 1; i<= pdf.numPages; i++) {
const page = await pdf.getPage(i);
consttextContent = await page.getTextContent();
constpageText = textContent.items
    .map((item: any) =>item.str)
    .join(' ');
extractedText += pageText + '\n';
 }

 // Generate thumbnail (first page)
constfirstPage = await pdf.getPage(1);
const viewport = firstPage.getViewport({ scale: 0.2 });
const canvas = document.createElement('canvas');
const context = canvas.getContext('2d')!;
canvas.height = viewport.height;
```

```
canvas.width = viewport.width;

awaitfirstPage.render({
canvasContext: context,
viewport: viewport,
  }).promise;

conststhumbnailUrl = canvas.toDataURL();

return {
id: crypto.randomUUID(),
title: info?.Title || file.name.replace('.pdf',''),
author: info?.Author,
pages: pdf.numPages,
fileSize: file.size,
filePath: URL.createObjectURL(file),
thumbnailUrl,
extractedText,
    // other metadata...
  };
}
```

2. **Library state and global search across documents**

```
// src/pages/Index.tsx (load on mount)
useEffect(() => {
constsavedDocs = localStorage.getItem('pdf-library-documents');
if (savedDocs) {
try {
setDocuments(JSON.parse(savedDocs));
    } catch (error) {
console.error('Error loading saved documents:', error);
    }
  }
}, []);

// save when changed
useEffect(() => {
localStorage.setItem('pdf-library-documents', JSON.stringify(documents));
}, [documents]);

// compute search results when searchTerm/documents change
useEffect(() => {
if (!searchTerm.trim()) {
setSearchResults([]);
return;
  }
```

```
const results: SearchResult[] = documents
   .map(doc => {
const matches = doc.extractedText
       ? PDFUtils.searchInText(doc.extractedText, searchTerm)
       : [];

return {
documentId: doc.id,
matches,
totalMatches: matches.length,
     };
   })
   .filter(result =>result.totalMatches> 0);

setSearchResults(results);
}, [searchTerm, documents]);
```

3. **PDF rendering with PDF.js**

-Render page utility

```
// src/utils/pdfUtils.ts
staticasyncrenderPage(
pdfDoc: any,
pageNumber: number,
canvas: HTMLCanvasElement,
scale: number = 1.5
): Promise<void> {
const page = await pdfDoc.getPage(pageNumber);
const viewport = page.getViewport({ scale });
const context = canvas.getContext('2d')!;

canvas.height = viewport.height;
canvas.width = viewport.width;

awaitpage.render({
canvasContext: context,
viewport: viewport,
  }).promise;
}
```

-Viewer uses canvasRef and calls render

```
// src/components/PDFLibrary/PDFViewer.tsx (excerpt)
constcanvasRef = useRef<HTMLCanvasElement>(null);
...
useEffect(() => {
if (pdfDoc&&canvasRef.current) {
renderPage();
  }
}, [pdfDoc, currentPage, scale]);
```

```
constrenderPage = async () => {
if (!pdfDoc || !canvasRef.current) return;
setIsLoading(true);
try {
awaitPDFUtils.renderPage(pdfDoc, currentPage, canvasRef.current, scale);
  } catch (error) {
console.error('Error rendering page:', error);
  } finally {
setIsLoading(false);
  }
};
...
// in JSX:
<canvas ref={canvasRef} className="max-w-full h-auto" />
```

4. **In-document search and search results panel**
   -searchInTextImplementation

```
// src/utils/pdfUtils.ts
staticsearchInText(text: string, searchTerm: string, pageTexts?: string[]): SearchMatch[] {
if (!searchTerm.trim()) return [];

const matches: SearchMatch[] = [];
const regex = new RegExp(searchTerm.replace(/[.*+?^${}()|[\]\\]/g, '\\$&'), 'gi');

if (pageTexts&&pageTexts.length> 0) {
   // Search within individual pages
pageTexts.forEach((pageText, pageIndex) => {
let match;
constpageRegex = new RegExp(searchTerm.replace(/[.*+?^${}()|[\]\\]/g, '\\$&'), 'gi');

while ((match = pageRegex.exec(pageText)) !== null) {
conststartIndex = match.index;
constendIndex = startIndex + match[0].length;
constbeforeContext = pageText.substring(Math.max(0, startIndex - 30), startIndex);
constafterContext = pageText.substring(endIndex, Math.min(pageText.length, endIndex + 30));

matches.push({
pageNumber: pageIndex + 1,
context: match[0],
startIndex,
endIndex,
beforeContext: beforeContext.trim(),
afterContext: afterContext.trim()
      });
```

```
    }
   });
  } else {
   // fallback: search in whole text (single "document" text)
let match;
while ((match = regex.exec(text)) !== null) {
conststartIndex = match.index;
constendIndex = startIndex + match[0].length;
constbeforeContext = text.substring(Math.max(0, startIndex - 30), startIndex);
constafterContext = text.substring(endIndex, Math.min(text.length, endIndex + 30));

matches.push({
pageNumber: 1,
context: match[0],
startIndex,
endIndex,
beforeContext: beforeContext.trim(),
afterContext: afterContext.trim()
    });
   }
  }

return matches;
}
```

-Search results panel

```
// src/components/PDFLibrary/SearchResultsPanel.tsx (core rendering)
{searchResults.map((result, index) => (
<Button key={index} variant="ghost" onClick={() =>onResultClick(index)}>
<div className="w-full text-left">
<div className="flex items-center justify-between">
<div className="text-sm font-medium">
        Page {result.pageNumber}
</div>
<Badge>{/* maybe highlight if current */}</Badge>
</div>
<div className="text-xs text-muted-foreground line-clamp-2 break-words">
      {result.beforeContext}
<span className="bg-primary/20 text-primary font-medium px-1 rounded">
        {searchTerm}
</span>
      {result.afterContext}
</div>
</div>
</Button>
))}
```

5. **Viewer controls: zoom, page navigation**

```
// PDFViewer.tsx
constzoomIn = () =>setScale(prev =>Math.min(prev + 0.25, 3));
constzoomOut = () =>setScale(prev =>Math.max(prev - 0.25, 0.5));

constnextPage = () =>setCurrentPage(prev =>Math.min(prev + 1, document.pages));
constprevPage = () =>setCurrentPage(prev =>Math.max(prev - 1, 1));
```

6. **Document and its metadata display**

```
// src/components/PDFLibrary/DocumentCard.tsx
consthighlightedTitle = searchTerm
  ? PDFUtils.highlightText(document.title, searchTerm) // (example helper)
  : document.title;

<div                    className="font-semibold                    text-foreground"
dangerouslySetInnerHTML={{ __html: highlightedTitle }} />
...
<Button variant="ghost" size="icon" onClick={(e) => { e.stopPropagation();
onDelete(); }}>
<Trash2 className="h-3 w-3" />
</Button>
```