

SAINT MICHAEL COLLEGE OF CARAGA

WEB-BASED ARCHIVAL AND DOCUMENT MAPPING FOR BUENAVISTA'S MUNICIPAL SOCIAL WELFARE SERVICES

A Thesis Presented To
The Faculty of
College of Computing and Information Sciences

In Partial Requirements for the Degree
Bachelor of Science in Information Technology



By

WINSTON LLOYD B. TIMOGAN
JULI-ANN S. ECHALICO
JOHN CARLO M. BONOTAN
JAMES NOLIE S. PIEL
ELMERTO G. OSIGAN JR.

ADVISER: MARLON JUHN M. TIMOGAN

APRIL 2025

SAINT MICHAEL COLLEGE OF CARAGA

SAINT MICHAEL COLLEGE OF CARAGA
College Of Computing and Information Sciences

CERTIFICATE OF RESEARCH APPROVAL

This thesis entitled "**WEB-BASED ARCHIVAL AND DOCUMENT MAPPING FOR BUENAVISTA'S MUNICIPAL SOCIAL WELFARE SERVICES**" presented and submitted by **WINSTON LLOYD B. TIMOGAN, JULI-ANN S. ECHALICO, JOHN CARLO M. BONOTAN, JAMES NOLIE S. PIEL, and ELMERTO G. OSIGAN JR.** in partial fulfillment of the requirements for the degree of **BACHELOR OF SCIENCE AND INFORMATION TECHNOLOGY** is hereby accepted and recommended for Oral Examination.

MARLON JUHN M. TIMOGAN, MIT
Adviser

LEALIL I. PALACIO, MSIT (CAR)
Member

MARISOL S. ROSARIO
Member

MICHELLE ANN G. LUCINO
Member

DAISA O. GUPIT, MIT
Member

KENNETH IAN B. BARRERA, MA
Chairperson

ACCEPTED as partial fulfillment of the requirements for the degree of Bachelor of Science and Information Technology.

APPROVED by the tribunal at the Oral Examination with the grade of _____.

MARLON JUHN M. TIMOGAN, MIT
Adviser

LEALIL I. PALACIO, MSIT (CAR)
Member

MARISOL S. ROSARIO
Member

MICHELLE ANN G. LUCINO
Member

DAISA O. GUPIT, MIT
Member

KENNETH IAN B. BARRERA, MA
Chairperson



ABSTRACT

TITLE:	Web-Based Archival and Document Mapping for Buenavista's Municipal Social Welfare Services
AUTHOR:	Winston Lloyd B. Timogan, Juli-Ann S. Echalico, John Carlo M. Bonotan, James Nolie S. Piel, Elmerto G. Osigan Jr.
DEGREE:	Bachelor of Science in Information Technology
ADVISER:	Marlon Juhn M. Timogan, MIT
PLACE OF	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX
PUBLICATION:	
DATE	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX
PAGES	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX

I. OBJECTIVES

This study aims to design, develop, and implement a web-based system for Buenavista's Municipal Social Welfare Services, titled Web-Based Archival and Document Mapping System. The primary objectives are to develop a storage system that efficiently organizes files to ensure easy access to all records and documents; to establish Role-Based Access Control through user authentication and permission settings, safeguarding the confidentiality of private data by limiting access to authorized users; and to implement a document mapping feature that enables quick and easy retrieval of needed files.

II. METHODOLOGY

The system was developed using the Systems Development Life Cycle (SDLC) approach with a structured three-tier architecture to ensure scalability, maintainability, and efficient data management. Design tools such as conceptual, use case, activity, and sequence diagrams were used to model system behavior, while an Entity-Relationship Diagram (ERD) structured a secure and organized database. Development utilized PHP, MySQL, and Visual Studio Code, resulting in a responsive, user-friendly interface optimized for performance and usability.



III. FINDINGS

The system received high ratings in functionality, performance efficiency, and usability based on ISO 25010 standards—reflected in a high total weighted mean of 3.65—confirming its effectiveness, reliability, and user-friendliness, with recommendations for ongoing optimization and feature customization to support long-term adaptability.

IV. RECOMMENDATIONS

To enhance the system's effectiveness, future efforts should focus on improving storage capacity, optimizing search and retrieval, enhancing user training and interface design, integrating real-time tracking and automated classification, expanding access to key stakeholders, and refining document mapping and reporting features for greater efficiency and user satisfaction.

KEYWORDS: *Document Archival, Document Mapping or Tracking, Rôle-Based Access Control*



DEDICATION

This capstone research is dedicated with heartfelt appreciation to the people who have been part of our journey and success.

To our families, thank you for your unwavering love, patience, and support. Your encouragement gave us the strength to persevere through every challenge and milestone of this research endeavor.

To our adviser, Mr. Marlon Juhn Timogan, we are deeply grateful for your invaluable guidance, support, and dedication. Your insightful advice, constructive feedback, and encouragement played a vital role in shaping this study and bringing it to completion. Your mentorship has left a lasting mark on both our academic and personal growth.

This work is also a reflection of the collaboration, hard work, and shared commitment among us. Each contribution, no matter how big or small, helped transform our ideas into a meaningful system that aims to benefit the community.

This capstone research would not have been possible without the people who stood beside us from beginning to end. Thank you.



ACKNOWLEDGEMENTS

First and foremost, we give all glory and thanks to God for His guidance, wisdom, and strength throughout this journey. Without His grace, this capstone thesis would not have been possible.

We are forever grateful for His unwavering support in helping us overcome every challenge. We sincerely thank our thesis advisor, Mr. Marlon Juhn Timogan, for his invaluable guidance, expertise, and patience. Your constructive advice helped shape this thesis, and your mentorship greatly contributed to our growth.

We also extend our appreciation to our professors and committee members for their helpful insights and support, which were essential in refining our study.

To our families, thank you for your endless love, understanding, and sacrifices. Your emotional and financial support gave us strength and motivation throughout this endeavor.

We are especially grateful to the Tertiary Education Subsidy for providing the financial assistance that allowed us to focus on our academic goals without additional burden.

Special thanks to our friends, peers, and mentors who supported us through teamwork, advice, and encouragement. Your presence made a meaningful difference.

Finally, to everyone who helped in any way—thank you. Your contributions made this journey smoother and more fulfilling. With heartfelt gratitude, we dedicate this thesis to all of you.



TABLE OF CONTENTS

	Pages
TITLE PAGE	i
CERTIFICATE OF RESEARCH APPROVAL	ii
ABSTRACT	iii
DEDICATION	v
ACKNOWLEDGEMENT	vi
TABLE OF CONTENTS	vii
LIST OF FIGURES	viii
LIST OF TABLES	xii
 CHAPTER	
1 INTRODUCTION	1
1.1 Project Context	1
1.2 Objective of the Study	2
1.3 Scope and Limitations	3
1.4 Definition of Terms	3
 2 REVIEW OF RELATED LITERATURE	6
 3 SOFTWARE REQUIREMENTS AND DESIGN SPECIFICATION	12
3.1 System Architecture	12
3.2 Conceptual Diagram	12
3.3 Use Case Diagram	13
3.4 Activity Diagram	14
3.5 Sequence Diagram	16
3.6 ERD (Database Design)	18
3.7 User Interface Design (Prototype)	19
3.8 Software Platforms, Development Environments and Tools	38
3.9 Hardware Requirements	39
 4 SOFTWARE DEVELOPMENT AND TESTING	40
4.1 Development Process	40
4.2 Testing Process	110
- Functionality Test	112
- Performance Test	114
- Usability Test	115
 5 SUMMARY, CONCLUSIONS AND RECOMMENDATIONS	118
5.1 Summary of Findings	118
5.2 Conclusion	119
5.3 Recommendations	119
5.4 Ethical Standard	121
 REFERENCES	126
APPENDICES	131
CERTIFICATE OF TECHNOLOGY BASED ASSESSMENT	
PUBLISHABLE JOURNAL	



LIST OF FIGURES

Figure No.	Title	Page
1	System Architecture	12
2	Conceptual Diagram	12
3	Use Case Diagram	13
4	Activity Diagram – Personnel	14
5	Activity Diagram – Admin	15
6	Sequence Diagram – Personnel	16
7	Sequence Diagram – Admin	17
8	Entity Relationship Diagram	18
9	Personnel / Admin – Login Page	19
10	Personnel – Home Page	19
11	Personnel – Folder Details	20
12	Personnel – Upload File	20
13	Personnel – Display File Card Style	21
14	Personnel – Display File Table Style	21
15	Personnel – View PDF File	22
16	Personnel – Generate PDF and Print	23
17	Personnel – Profile	23
18	Personnel – Reports	24
19	Personnel – Add Parent Folder	24
20	Personnel – List of Pending Files	25
21	Personnel – List of Approved Files	26
22	Personnel – List of Declined Files	26
23	Admin – Dashboard	27
24	Admin – Reports	28
25	Admin – List of Pending Files	28
26	Admin – List of Approved Files	29
27	Admin – List of Declined Files	30
28	Admin – List Departments	30
29	Admin – Add Department	31



30	Admin – Display Folders	31
31	Admin – Folder Details	32
32	Admin – Display Files Card Style	33
33	Admin – Display Files Table Style	33
34	Admin – View PDF File	34
35	Admin – Create Parent Folder	34
36	Admin – Create Subfolder and Set Metadata Title	35
37	Admin – Display All Accounts	36
38	Admin – Register Personnel Account	37
39	Input-Process-Output Diagram	40
40	Admin/Personnel Login Page	42
41	Admin/Personnel – Login Error Notification	43
42	Admin/Personnel – Login Successful Notification	43
43	Code snippet for admin/personnel – Login Page	44
44	Code Snippet for Admin/Personnel Login Page – Login.php	46
45	Admin – Dashboard	47
46	Code snippet for pie and bar graph – chart	49
47	Code Snippet for dashboard – conn.php	51
48	Code Snippet for Admin Dashboard – chart.php	52
49	Admin – Reports	53
50	Code snippet for card – reports	54
51	Code Snippet for admin reports – reports.php	55
52	Admin - List of Pending files	56
53	Admin - List of Approve Files	57
54	Admin - List of Decline Files	57
55	Code Snippet for display Pending / Approve / Decline - db_function.php	59
	Admin - List Departments	
56	Code Snippet for display department – department.php	61
57	Admin - Add Department	61
58	Code snippet for department - add department	62
59	Code snippet for department – edit department	63



60	Code Snippet for adding department – create.php	64
61	Admin - Create Parent folder	66
62	Code Snippet for creating parent folder – create.php	67
63	Admin – Display Folders	67
64	Code Snippet for display folder – folders.php	68
65	Code Snippet for displaying folders and files - get_folders.php	69
66	Admin – Create SubFolder	70
67	Set metadata and input type – create subfolder	71
68	Code Snippet for creating subfolder and set metadata title - Create.php	72
69	Admin – Display Files Card Style	74
70	Code snippet for card and table – display files	75
71	Code Snippet for display file card style – folders.php	76
72	Admin – Display Files Table Style	77
73	Code Snippet for display file table style – folders.php	78
74	Admin - Document PDF Viewer	79
75	Code Snippet for PDF VIEWER – open-file.js	80
76	Admin - Create Account	81
77	Code snippet for account – create account	82
78	Code Snippet for create personnel account – create.php	83
79	Admin – Display Accounts	84
80	Code snippet for account – display accounts	85
81	Code Snippet for retrieve personnel account – get.php	86
82	Personnel – Display Folders	87
83	Personnel – Display Files Card Style	88
84	Personnel – Display Files Table Style	89
85	Personnel – Document PDF Viewer	89
86	Personnel - Upload a file	90
87	Code snippet for modal – upload file	91
88	Code snippet for uploading – upload file	92
89	Code Snippet for uploading file – upload_file.php	94
90	Personnel – Profile	97



91	Code Snippet for retrieve information – login.php	99
92	Personnel – update profile	99
93	Code snippet for profile – edit profile	100
94	Code Snippet for updating profile – update.php	102
95	Personnel – Reports	103
96	Code snippet for reports total count of folders and uploaded files –	104
97	reports.php	105
98	Personnel – Add Parent Folder	106
99	Personnel - List of Pending Files	106
100	Personnel - List of Approved Files	107
101	Personnel – List of Declined File	107
102	Code snippet for generate pdf and print – report	109



LIST OF TABLES

Table	Pages
1 Software Requirements	38
2 Hardware Requirements	39
3 Functional Suitability	110
4 Performance Efficiency	111
5 Usability	111
6 Respondent's Distribution	112
7 Functional Suitability	112
8 Performance Efficiency	114
9 Usability	115
10 Summary Table of the Over-all Mean and Grand Distribution of the Acceptability Level	117

SAINT MICHAEL COLLEGE OF CARAGA

CHAPTER I

INTRODUCTION

1.1 Project Context

The Municipal Social Welfare Development is a local government agency charged with the responsibility of formulating a plan for administering and rendering social safety nets throughout their jurisdiction and implementing poverty alleviation programs in support of the poor, weak, or disadvantaged members of society. For this reason, a lot of documentation is managed within its offices. However, it is often difficult to find particular papers due to their improper monitoring or arrangement in the office because of the large document volume.

Global studies have highlighted the effectiveness of digital archiving systems in addressing manual document management challenges. For example, the web-based Archiving System implemented by Commission C of DPRD North Sumatra successfully resolved issues of document retrieval, significantly reducing errors and delays caused by manual processes, according to Syamia et al. [1]. However, gaps remain in tailoring these solutions to other specific contexts, highlighting the need for further research and customization of digital archiving systems.

In the local context, Rodriguez et al. developed the eDALAYON system in the Philippines to address document retrieval delays and storage challenges within the Department of the Interior and Local Government (DILG) in Negros Occidental [2]. The system integrates cloud storage and automation, allowing for better document tracking and retrieval while reducing human error. The introduction of workflows and version-control mechanisms has streamlined operations within the department. However, similar to the global examples, these solutions must also be tailored to other local government units (LGUs) with specific needs, such as the Buenavista Municipal Social Welfare Services, where a high volume of sensitive documents must be securely managed.



The Municipal Social Welfare Development office faces several issues due to file misplacements caused by the current filing methods. The office utilizes primarily hard copies for documentation storage. It essentially acts as a supervisor to their arrangement. Consequently, delays and mistakes in service provision compromise the quality of the service.

To address these issues and enhance operations, the researchers will develop a web-based archival and document mapping system for Municipal Social Welfare and Development (MSWD) customized to their needs. This approach is designed to enhance document organization for access and reduce the risk of losing files. The researchers prioritize protecting the confidentiality of their documents and resources. According to the Data Privacy Act regulations, the system will only be accessible to authorized individuals. Introducing this technology at the MSWD office is anticipated to increase efficiency and improve the delivery of services to the community.

The system for archiving and mapping documents at the Buenavista Municipal Social Welfare Services is crucial for improving the efficiency and accuracy of document management in the MSWD department to elevate the quality of services provided overall. Introducing archival procedures will help reduce errors made by humans while also making social welfare files and documents transparently tracked and accessible. This system guarantees the storage of records that are up to date and ready for auditing or any necessary purposes. They will facilitate effective service delivery in the future.

1.2 Objective of the Study

This study aims to design, develop, and implement a web-based system for Buenavista's Municipal Social Welfare Services, titled Web-Based Archival and Document Mapping System. Specifically, it aims to:

1. develop a storage system that can organize files efficiently to guarantee access to all records and documents from the Municipal Social Welfare Services.



-
2. establish Role-Based Access Control within the system using user authentication and permission settings to guarantee the confidentiality of private data, ensuring access to certain files is limited to authorized users.
 3. map needed files for quick and easy document retrieval and access.

1.3 Scope and Limitations

The system will provide comprehensive archival and document mapping services tailored for the Municipal Social Welfare and Development Office of Buenavista. It is designed to streamline the management and organization of both digital and physical records, ensuring secure storage and easy retrieval. This implementation will support the operational requirements of the office, enhancing document accessibility, confidentiality, and overall efficiency in handling essential records.

The system will allow personnel to upload files and set metadata, ensuring organized and searchable document storage. Only the administrator has the authority to create and manage user accounts, assign folders to personnel, and oversee all system functionalities. The system will also facilitate the archiving of both digital and physical records. Additionally, it includes a document mapping feature that enables personnel to locate hard copies of records by searching for their designated locations within the office. The system's interface is only accessible on smartphones, laptop, and computer through Google Chrome and Safari.

1.4 Definition of Terms

The following is a list of terms and their definitions relevant to the Web-based Archival and Document Mapping System for Buenavista's Municipal Social Welfare Services. These key terms are essential for comprehending the system's functionalities, roles, and processes, ensuring clarity and consistency throughout the implementation.



ADMMS (Archival and Document Mapping Management System) - This is a web-based platform designed to organize, preserve, and manage documents. It integrates archival management, document mapping, and metadata for efficient categorization and retrieval. The system includes security features like access controls, legal compliance through retention schedules, and workflow automation. ADMMS is vital for organizations to maintain a structured and accessible document repository.

Authorized Users – Individuals granted permission to access the web-based system are specifically Buenavista's Municipal Social Welfare Services workers.

Buenavista's Municipal Social Welfare Services – The specific department within the local government of Buenavista is responsible for providing social services to the community. The department's workers are the sole users of the developed system.

Categorized Folders – These are the documents that are assigned per category, which helps for easy management of the documents.

Digital formats – These formats are used for storing information electronically, one of the common formats is a PDF for files.

Document Archiving – Storing documents in digital format within the system for quick and easy records retrieval over time when needed.

Document Mapping – The process of systematizing and organizing documentation in the system. This function allows an individual to structure how they would like their files categorized and stored.

Document Repository – This is an organizational store where you can manage and track documents. Repositories typically handle different formats and types, such as documents, text files, sheets, and presentations. The main function of repositories is to provide a secure environment where you can access, manage, and store documents.



Integrity of Documents – This means keeping data safe from unwanted access. The system's design ensures that only authorized users can access and manage the documents.

I.R. - Information retrieval is the process of obtaining relevant information from a large repository based on a user's query. It involves searching through various content types, such as files and folders, to quickly and efficiently deliver accurate results that meet the user's needs.

MSWD (Municipal Social Welfare Development) – Is a branch of government that provides social protection to the less privileged and marginalized. It also encourages those disadvantaged sectors to become productive citizens.

Operational Needs – The specific requirements and tasks related to document management within Buenavista's Municipal Social Welfare Services. The system is designed to cater specifically to these needs, ensuring it supports the daily functions needed to help the workers.

RBAC (Role-Based Access Control) - This security model restricts system access based on users' roles within an organization. Permissions are assigned to roles, not individuals, allowing for streamlined user rights management. Each role defines specific actions users can perform, such as viewing and enhancing security by reducing unauthorized access. RBAC simplifies administration, especially in large organizations, and supports compliance with regulations by aligning access with job functions.



CHAPTER II

REVIEW OF RELATED LITERATURE

This chapter discusses the literature about web-based archive and document mapping systems in public administration. This review analyzes the current literature regarding the role of digital document management in public services and its possible effects on enhancing administrative efficiency and service delivery.

Web-based Archival for Secure and Centralized Storage

Document management has become one of the key areas for automation in modern office environments, as Nagrama highlighted in Ismael's study [3]. Though traditional paper-based management is still common, technological advancements have shifted organizations to electronic systems, and as Syamia noted in the study of Ibrahim et al., web-based digital archives enhance document storage and access, improving archiving efficiency [1]. These archives, accessible through websites, contribute significantly to improving document archiving processes. Furthermore, D. Nur, as cited by M. Churiyah et al., examined an integrated archiving system that optimizes document preparation and processing, thereby enhancing the efficiency of archiving tasks [4]. U. Alhasanah, also referenced by Churiyah et al., elaborates that web-based systems utilize the Internet for data storage and management, increasing accessibility and efficiency in document handling [4]. Web-based digital archives have proven important for document storage accessibility and efficiency because offices have already streamlined their workflows, ensuring documents can be easily retrieved and stored properly.

Biliran Province State University (BiPSU) faces rising costs and workload due to the challenges of managing and storing its growing volume of files. Such LAN-based systems are designed to meet the demands of efficient document storage and quick retrieval in academic



settings [5]. Despite the growing trend toward digitalization, Castro et al. argue that paper documents remain essential in daily office operations [6]. They stress the need for better integration between paper-based and electronic systems, as standardized source documents continue to play a crucial role in bookkeeping, transaction verification, and record-keeping. M.F. Castro and N. Soveizi reported that only 45% of organizations in the ASEAN region have adopted document management systems (DMS) despite the clear advantages they offer [6], [7]. A. Ikuomola further noted that 55% of organizations have refrained from adopting these systems due to negative user experiences and poorly designed interfaces, which lead to inefficiencies [8]. Digitalization may have great advantages, but let us not undermine the barriers they face, like limited adoption and negative user experience. Therefore, there is a need for better integration to be efficient and to satisfy the user.

Customer records in banks are often stored manually without centralized storage, making timely access difficult and leading to time-consuming retrieval, increased costs, and risks of damage or misplacement. These issues create significant inefficiencies and hinder business productivity and confidentiality [9]. In terms of organizational benefits, Jordan S. highlights that Document Management Systems (DMS) offer significant cost savings by reducing the financial burden of managing large volumes of documents while simplifying processes through automation, saving time with remote access, and enhancing efficiency by minimizing procedural steps [10]. Zahara emphasizes the importance of archives as a critical component of organizational information systems, underscoring the need for proper management [11]. A. Bani Ahmad et al. (2020) observed that paper-based systems consume valuable physical space and create inefficiencies in document retrieval and management, leading to increased costs and time delays [12]. L. Xing (2019) supported this by emphasizing that DMS provides better control, organization, and accessibility, which enables public institutions to streamline processes, reduce clutter, and



significantly enhance service delivery [13]. The development of a Web-Based Resident Information Management System simplifies the process of storing and accessing resident information compared to traditional management methods, making it an efficient solution for community organization [14]. LAN-based electronic document management systems, as outlined by a study, enhance office workflow by capturing, indexing, storing, and retrieving documents electronically while securely sharing information with stakeholders, reducing paper waste, and complying with standardized regulations [15]. Integrating a secure, web-based archival system ensures that documents are stored centrally, accessible by authorized personnel, and protected from damage, providing a sustainable solution for modern organizations [16]. With advancements in technology, automated systems have become essential, especially in managing information flow and interactions. One such system is the Digital Document Repository System of Researches, which aims to improve storage and access by digitalizing research documents and making them available online [17]. According to D.E. Mallares et al., based on studies by MES Hybrid Document Systems (MESHDS), archive document scanning refers to converting physical paper records into digital formats, such as images or text files, and establishing a systematic workflow to preserve and access these archived files electronically over an extended period [18]. Document management systems offer benefits like cost savings, improved efficiency, and accessibility. Streamlining operations can reduce reliance on the paper-based system, which could significantly impact document management and improve service delivery. The land-based and web-based systems enhance workflow and secure and store documents. This provides sustainable and efficient solutions for modern organizations.

Document Mapping for Quick Access and Retrieval

Document tracking systems improve the efficiency of retrieving documents online at any time, moving documents in and out of the office, and tracking the organization's processes, which



have been developed as a tracking solution system, as stated by E. A Bacalso et al. [19]. With the rapid growth of the Internet and new media, the explosive increase in multi-format media content has made it challenging for users to quickly and accurately find the required manuscript information in vast data collections [20]. Information retrieval (I.R.) systems, like search engines, have become essential for acquiring information and are also key components in dialogue, question-answering, and recommender systems [21]. Information retrieval (I.R.) involves retrieving relevant documents from large collections, and while its applications extend beyond online search engines to support organizational searches, efficiently retrieving documents from vast organizational data remains challenging [22]. In the context of big data and the Industrial Revolution 4.0 era, enhancing the efficiency of the document/information retrieval framework to handle the ever-growing volume of text data in an ever-more digital world is a must [23]. Keywords are used to index and retrieve the documents for the user query in a conventional information retrieval system. When more than one keyword is used to define a single concept in the documents and in the queries, inaccurate and incomplete results are produced by keyword-based retrieval systems [24]. As the volume of digital content and organizational data increases, document tracking and information retrieval (I.R.) systems are needed for effective document management. While tracking systems make document movement more accessible, I.R. systems make acquiring relevant information possible. However, traditional keyword-based I.R. systems tend to lose their degree of accuracy when directories are large and contain vast collections of data. The current big data outlook and the industry 4.0 era have created a call for better retrieval systems.

Usman et al. [25], Locating files using the manual method is a tedious and time-consuming process for most of the administrative staff of such institutions. Document tracking is essential for enhancing productivity in digital systems by offering a transparent overview of document



progression and status Salleh, S.F. [26]. Dislocation and overlooking of the timeline have always been the problems in document control Salleh, S.F. [26]. The field of challenges of fairness, accountability, transparency, and ethics in information retrieval has attracted a lot of attention, as stated by N. BERNARD et al. [27]. According to E. Avuclu et al., Digital law office tracking systems enhance efficiency by providing quick access to archived documents, ensuring transparency, traceability, and error reduction in case management [28]. Information retrieval (I.R.) is the process of obtaining relevant information from system resources to meet a specific need, and it has grown significantly over the years to become the primary method of accessing information [29]. Documents are integral to daily activities like communication and decision-making, and improper monitoring of sensitive or valuable documents can lead to their loss, delay, or intentional destruction, harming the owner or recipient [30]. Manual file retrieval is inefficient, making document tracking essential for boosting productivity, transparency, and document management. Digital tracking systems provide quick access, enhance traceability, and reduce errors, preventing sensitive documents from being lost, delayed, or mishandled. These systems improve overall workflow by offering apparent oversight of document status and progression, ensuring smoother operations in organizations.

Role-Based Access Control in Document Management

Role-based access control (RBAC) streamlines permission management by assigning permissions to roles, which are then linked to users based on their job functions [31]. This structure allows for easy role reassignment when users' responsibilities change without the need to adjust individual permissions [31]. Its alignment with organizational hierarchies makes RBAC the most widely adopted access control model, as it simplifies both the administration and review of access rights [32]. As noted by Junior M. et al., RBAC enhances authorization control by grouping users with similar access needs into roles, enabling flexible reassignment without disrupting the



overall access control system [33]. In document management systems, particularly in web-based environments, RBAC is crucial due to the increasing complexity of security policies and the challenges of remote access, which can create operational burdens and monitoring difficulties [34]. By assigning permissions based on user roles, RBAC simplifies access management and ensures users can only perform actions aligned with their specific responsibilities, effectively restricting unauthorized activities [35]. This method enhances security by integrating authentication and authorization, ensuring that users are authenticated before being granted access and then authorized based on their roles to interact with documents appropriately [36]. Thus, RBAC strengthens the protection of sensitive information and maintains the integrity of document handling within organizations [36]. Role-based access control (RBAC) simplifies permission management by assigning roles based on job functions, streamlining access control, and making it easy to adjust permissions when responsibilities change. Widely adopted for its alignment with organizational hierarchies, RBAC is crucial in document management systems, particularly for handling security policies and remote access challenges. By restricting actions based on roles, RBAC enhances security, ensuring only authorized users interact with sensitive documents, thereby protecting information integrity within organizations.



CHAPTER III

SOFTWARE REQUIREMENTS AND DESIGN SPECIFICATION

This chapter provides an in-depth analysis of the technical components of designing and developing the proposed system, including software and hardware requirements specifications. Additionally, it offers a comprehensive overview of the functionalities of the Web-based Archival and Documents Mapping Management System (ADMMS).

3.1 System Architecture

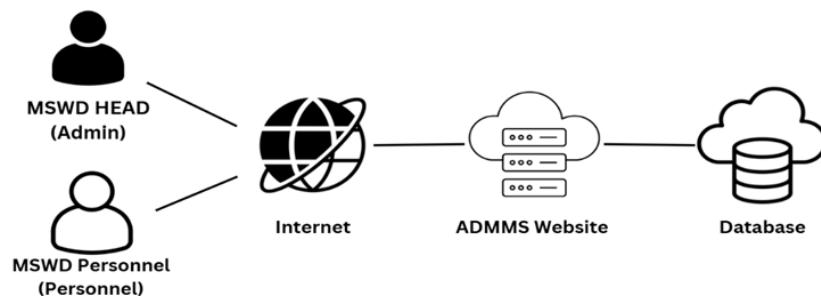


Figure 1. System Architecture

Figure 1 illustrates the fundamental process of accessing the Web-based Archival and Document Mapping Management System (ADMMS). This system facilitates user interaction for various roles, including administrator (MSWD Administrative Aide) and (MSWD personnel) accessing the platform through internet connectivity. Upon successful login, personnel can create folders and upload files, and administrators can manage the folders, create and view personnel accounts.

3.2 Conceptual Diagram

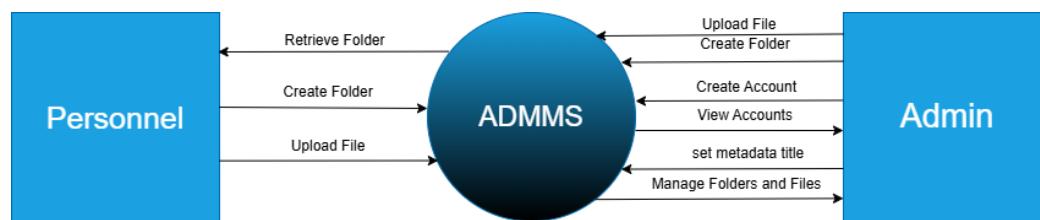


Figure 2. Conceptual Diagram (Level 0)



Figure 2 depicts the operational capabilities allotted to different users inside the ADMMS architecture. Administrators are granted the authority to manage personnel accounts and folders. In addition, the system allows for administrative functions such as upload files, creating accounts, and folders set metadata for personnel. On the other hand, personnel can create folders and upload relevant files in the ADMMS.

3.3 Use Case Diagram

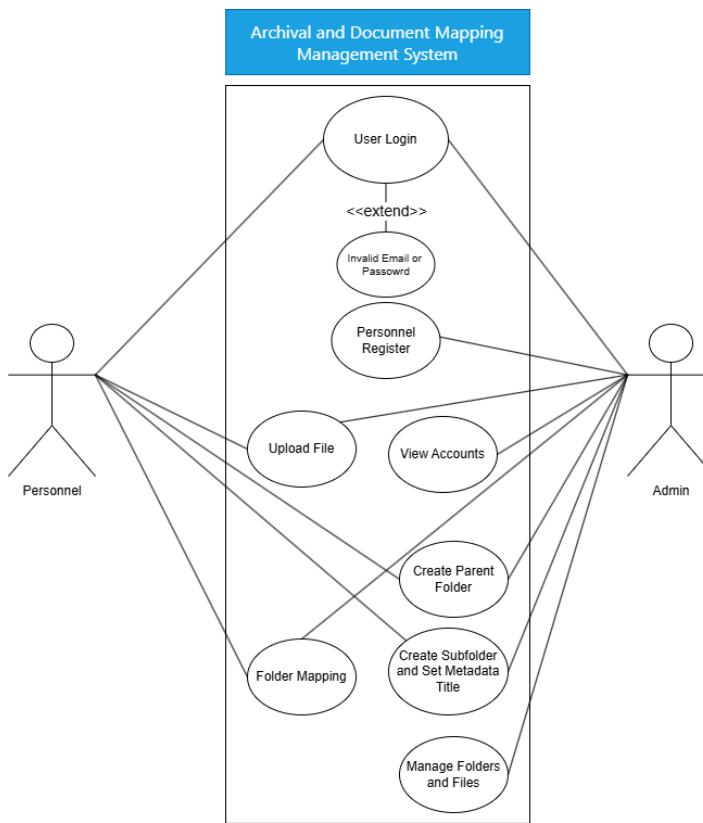


Figure 3. Use Case Diagram

Figure 3 shows the primary features available to each type of user and their corresponding roles within the system. All users must be logged in via login; the admin must register each personnel before logging in, except the admin, who does not have registration privileges. Personnel can upload relevant files and create folders. Administrators are granted the authority



to manage personnel accounts and allow for administrative functions such as upload files, creating accounts, and folders set metadata for personnel.

3.4 Activity Diagram

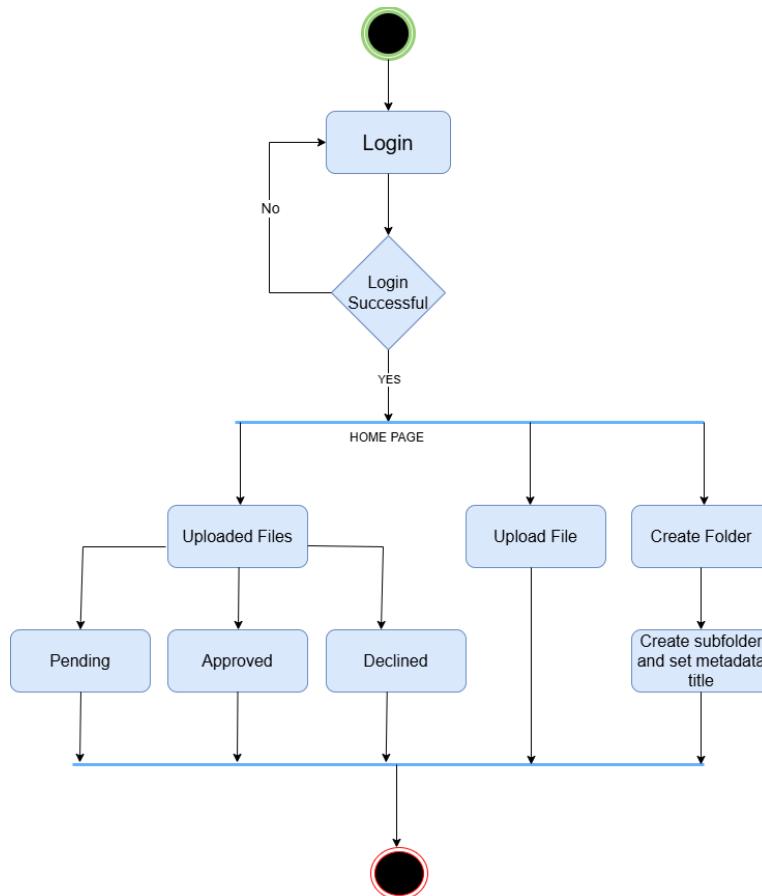


Figure 4. Activity Diagram – Personnel's

Figure 4 represents the system's series of activities available to personnel. Beginning with the login page, personnel can log in successfully. If they forget their username or password, they can ask the administrator. Alternatively, if they do not have an account yet, they can ask the administrator to create or give them one. After successfully logging in, personnel are directed to the ADMMS homepage, where they can upload files and create folders. Uploaded files fall into one of three categories pending, approved, and declined. The personnel may also log out of their accounts.

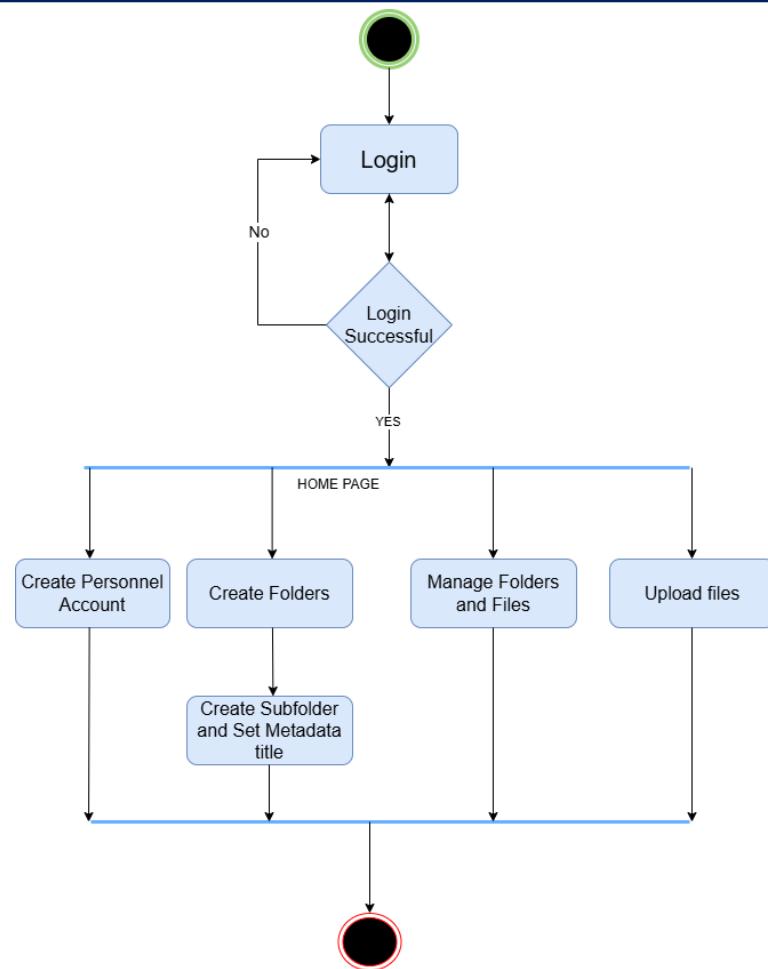


Figure 5. Activity Diagram – Admin

Figure 5 illustrates the series of activities available to the admin. On the login page, they are forwarded to the homepage after successful admin verification. The admin capabilities are viewing accounts, manage folders, creating accounts, folders, setting metadata for personnel, and upload file.



3.5 Sequence Diagram

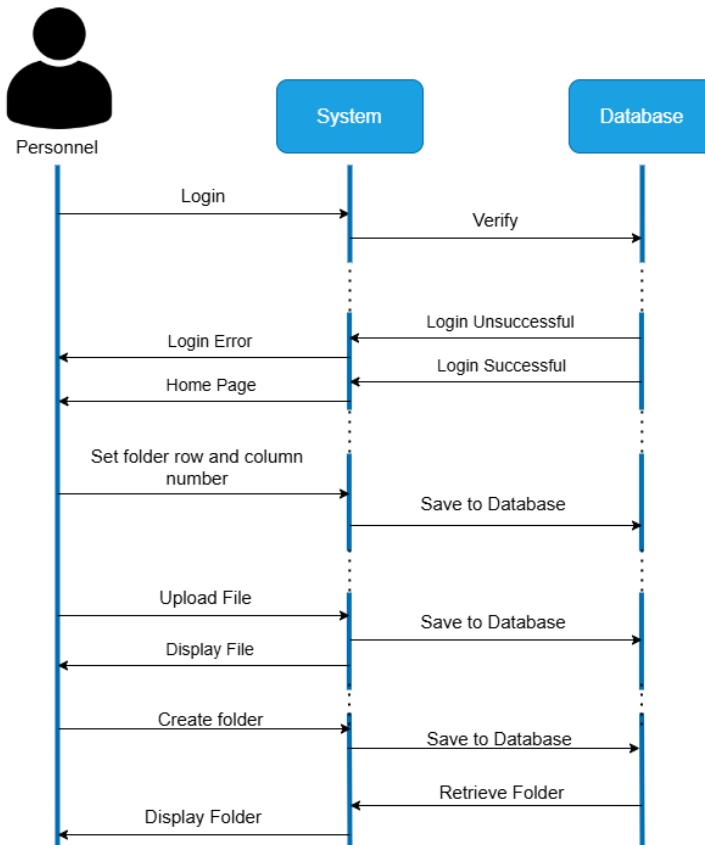


Figure 6. Sequence Diagram – Personnels

Figure 6 depicts the procedural steps available to the personnel side. Starting with the login interface, the personnel can complete verification procedures. After successful login, personnel can access functions like setting row and column number in the folder and uploading files for database storage.

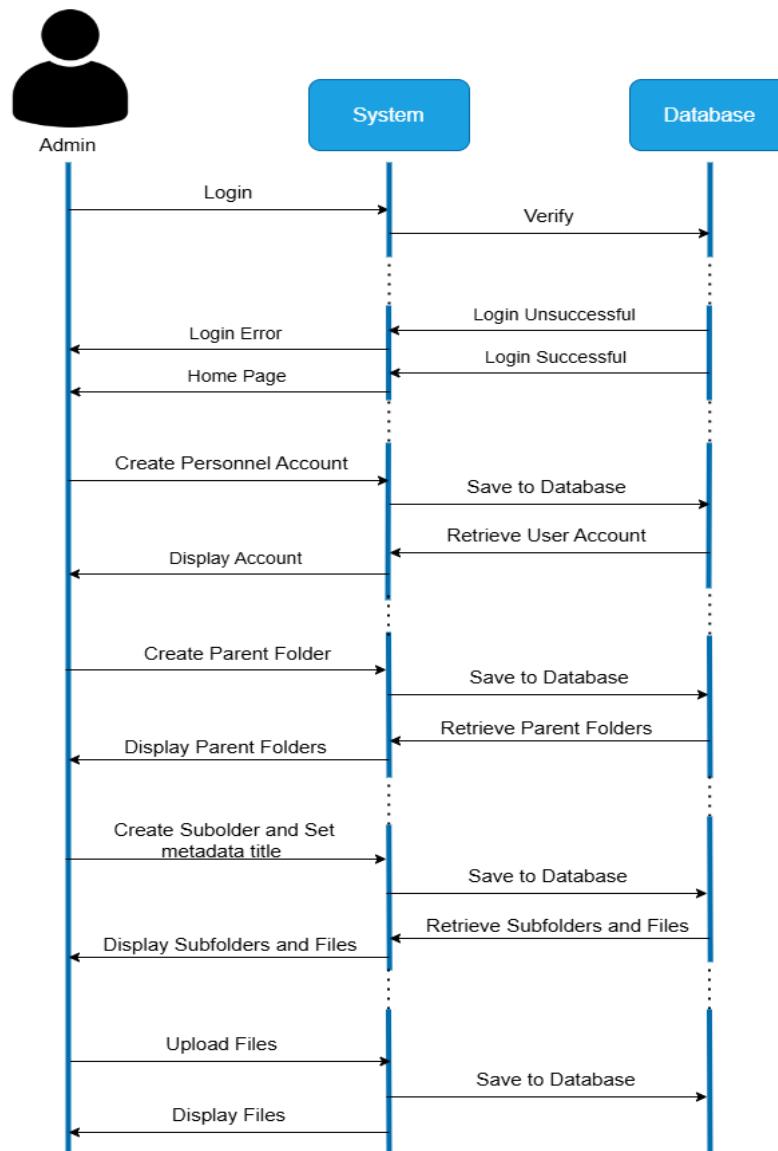


Figure 7. Sequence Diagram – Admin

Figure 7 shows the procedural steps available to administrator. Administrators can use the login interface to perform verification procedures, create accounts and folders, upload files, setting metadata for personnel, and manage all folders, files and accounts.



3.6 ERD (Database Design)

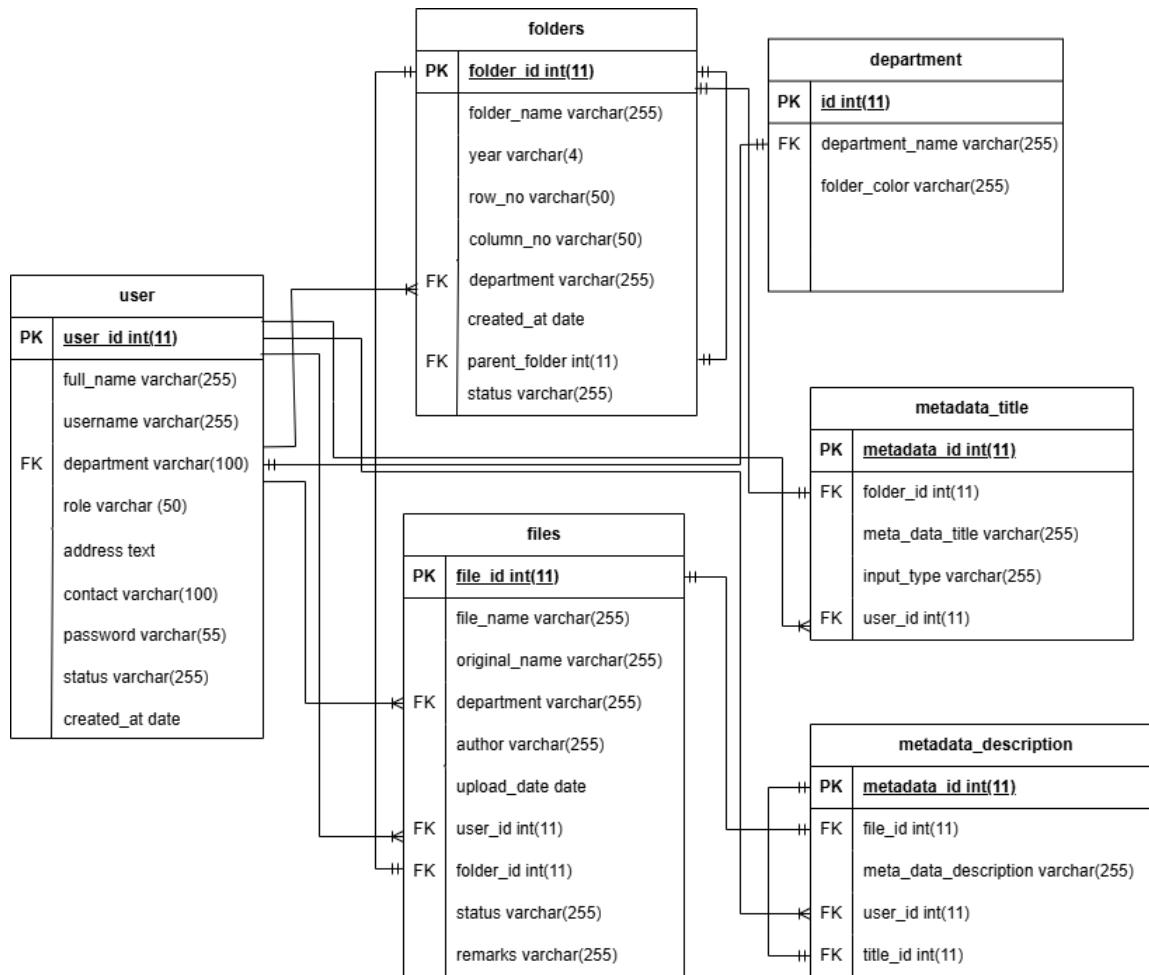


Figure 8. Entity Relationship Diagram – Database Design

Figure 8 presents a small schema for a Web-based Archival and Documents Mapping Management System (ADMMS). The ERD includes users, folder, department, files, metadata title, and metadata description. Each entity is methodically constructed using primary keys (P.K.) and foreign keys (F.K.) to ensure data integrity and relational connectivity throughout the database.



3.7 User Interface Design (Prototype)

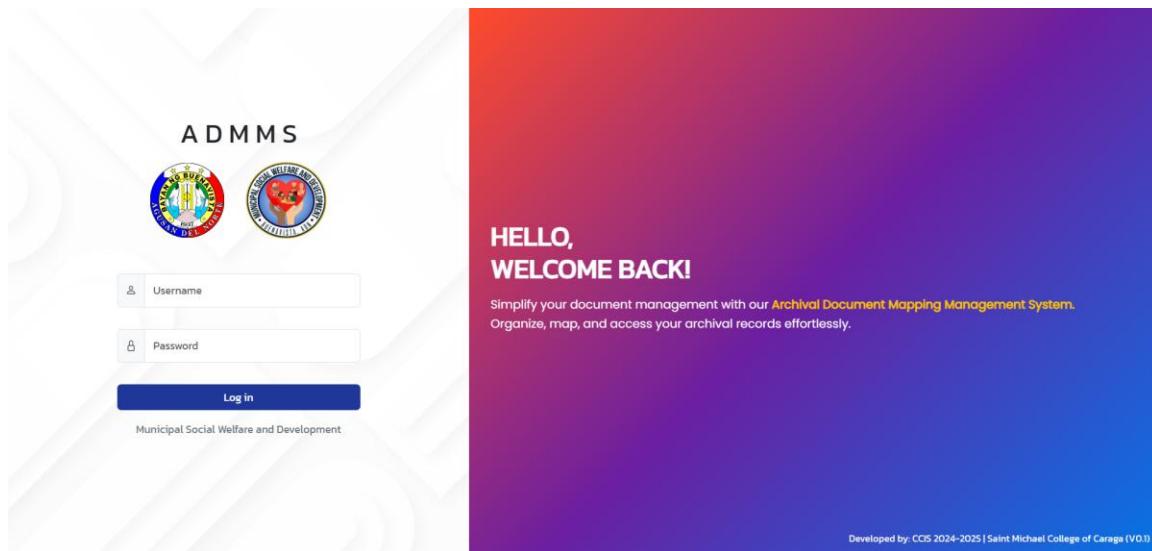


Figure 9. Personnel / Admin – Login Page

Figure 9 presents the Login Page for both personnel and administrators. Users are required to enter their username and password to access the system. If they forget their credentials, they can request assistance from the administrator.

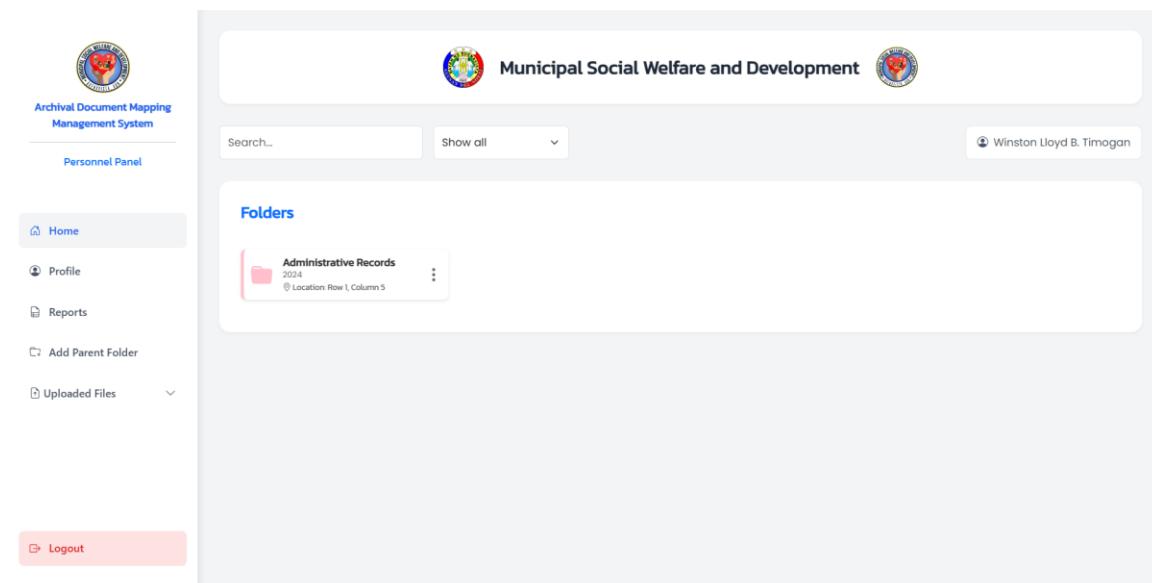


Figure 10. Personnel Side – Home Page (Personnel Interface Design)



Figure 10 displays the personnel side's home page. Archival and Document Mapping Management System of the MSWD. It allows personnel to upload files, search and display assigned folders.

The screenshot shows the 'Archival Document Mapping Management System' personnel interface. On the left sidebar, there are links for Home, Profile, Reports, Add Parent Folder, and Uploaded Files. A 'Logout' button is at the bottom. The main area has a header for 'Municipal Social Welfare and Development'. Below it is a search bar and a dropdown menu. The 'Folders' section lists 'Administrative Records' from 2024, located at Row 1, Column 5. A modal window titled 'Folder Details' is open, showing the following information:

Folder Details	
Folder Name:	Administrative Records
Year:	2024
Row Number:	1 <input checked="" type="checkbox"/>
Column Number:	5 <input checked="" type="checkbox"/>
Department:	Administrative Records
Created At:	April 7, 2025

Figure 11. Personnel Side – Folder Details (Personnel Interface Design)

Figure 11 shows the Folder Details modal on the personnel home screen, providing specific information (e.g., folder name, year, row and column numbers, department, and Created At.). Additionally, personnel can set the row and column number to locate the folder.

The screenshot shows the 'Archival Document Mapping Management System' personnel interface. The left sidebar includes a 'Logout' button. The main area features a header for 'Municipal Social Welfare and Development'. A search bar and a dropdown menu are present. The 'Folders' section shows 'Main Directory / Administrative Rec'. A modal window titled 'Upload File' is open, containing fields for file selection, file name, file owner, date of purchase, supplier, and amount. At the bottom are 'Cancel' and 'Upload' buttons. A blue circular '+' icon is located in the bottom right corner of the main interface.

Figure 12. Personnel Side – Upload File (Personnel Interface Design)



Figure 12 shows the upload file modal on the personnel home screen, enabling personnel to upload a file by selecting a file, entering the file name, specifying the file owner, and providing a metadata description.

Figure 13. Personnel Side – Display File Card Style (Personnel Interface Design)

Figure 13 shows uploaded files in a card-style format, providing a clear and structured view of each file. Personnel can double-tap a card to open and view the PDF file directly. Additionally, clicking the 3-dot icon (:) reveals detailed file information. This design ensures a streamlined and user-friendly file management experience.

#	File name	Upload date	File owner	Uploaded by	Date of Purchase	Supplier	Amount	File
1	Glass	April 07, 2025	Elmerito	Winston Lloyd B. Timogan	March 25, 2025	Jr Glass	10,500	

Figure 14. Personnel Side – Display File Table Style (Personnel Interface Design)



Figure 14 displays the file table style on the personnel side, presenting uploaded files in a structured tabular format. Each row in the table contains essential file details, including the file name, upload date, file owner, uploaded by, metadata title, and file. The table also includes action buttons for viewing pdf file. This layout ensures organized and efficient file management experience for personnel.

A screenshot of a web-based application interface. On the left, there's a sidebar titled "Archival Document Mapping Management System" with a "Personnel Panel" section containing links for "Home", "Profile", "Reports", "Add Parent Folder", and "Uploaded Files". Below these is a "Logout" button. The main area is titled "Document Viewer" and shows a PDF document titled "WEB-BASED ARCHIVAL AND DOCUMENT MAPPING FOR BUENAVENTURA'S MUNICIPAL SOCIAL WELFARE SERVICES". The PDF content includes a capstone project presentation to the Faculty of the College of Computing and Information Sciences at Saint Michael College of Caraga, Nasipit, Agusan del Norte. It also mentions partial fulfillment of requirements for a Bachelor of Science in Information Technology. On the right side of the viewer, there's a user profile for "Winston Lloyd B. Timogan" and a small "File" icon. The bottom right corner of the viewer has a blue circular button with a plus sign.

Figure 15. Personnel Side – View PDF File (Personnel Interface Design)

Figure 15 displays the modal document viewing page on the personnel home screen, allowing users to preview uploaded documents without leaving the interface. This feature provides a seamless viewing experience by enabling personnel to open PDFs or other supported file formats within a pop-up window. The modal includes options for zooming, scrolling, and closing the document, ensuring convenient access to file content while maintaining workflow efficiency.



Figure 16. Personnel Side – Generate PDF and Print (Personnel Interface Design)

Figure 16 illustrates the Generate PDF and Print feature on the personnel interface, allowing users to convert documents into PDF format and print them directly. This functionality ensures that personnel can efficiently generate professional and easily shareable document copies.

Figure 17. Personnel Side – Profile (Personnel Interface Design)

Figure 17 displays the Profile section on the personnel interface, providing users with a structured overview of their personal and account details. This section includes essential information such as full name, username, address, contact information, and department. The



profile interface may also offer options for updating personal details, changing passwords, and managing account settings, ensuring a personalized and secure user experience.

Figure 18. Personnel Side – Reports (Personnel Interface Design)

Figure 18 presents the Reports section on the personnel interface, displaying a summary of document records specific to the assigned department. This section includes key statistics such as the total number of folders and total number of files managed by the personnel. The report view provides a clear and organized overview, ensuring efficient record management within the designated department.

Figure 19. Personnel Side – Add Parent Folder (Personnel Interface Design)



Figure 19 display the Add Parent Folder feature enables personnel to create a new parent folder, limited to their designated department. Users are required to provide details such as the folder name and year, helping maintain structured and department-specific document organization.

A screenshot of a web-based application interface titled "Archival Document Mapping Management System". The main content area is titled "Municipal Social Welfare and Development". It shows a table of "List of Pending Files" with one item: "Food" uploaded on April 7, 2025, by file owner Juli-ann. Metadata includes "Date of Purchase" (April 7, 2025), "Supplier" (Marave), and "Amount" (15,000). The table also has columns for "Upload date", "File name", "File owner", "Metadata title", "Metadata description", and "File Path". Action buttons at the bottom include "View", "Edit", and "Delete". The left sidebar shows navigation links: Home, Profile, Reports, Add Parent Folder, and Uploaded Files (with a dropdown arrow). A "Logout" button is at the bottom of the sidebar.

Figure 20. Personnel Side – List of Pending Files (Personnel Interface Design)

Figure 20 displays the List of Pending Files section on the personnel interface, showing files that are awaiting approval. This section presents a structured view of all pending documents, including details such as upload date, file name, file owner, metadata title, and metadata description.



Upload date	File name	File owner
Apr 7, 2025	Glass	Elmerito

Metadata title	Metadata description
Date of Purchase	March 25, 2025
Supplier	Jr Glass
Amount	10,500

File Path
Administrative Records / Acknowledgement Receipt / Supplies / Glass

[View](#)

Figure 21. Personnel Side – List of Approved Files (Personnel Interface Design)

Figure 21 displays the List of Approved Files section on the personnel interface, showing files that have been reviewed and approved. This section presents a structured view of all approved documents, including details such as upload date, file name, file owner, metadata title, and metadata description. The interface enables personnel to easily access and manage their approved files for further use or reference.

Upload date	File name	File owner	Remarks
Apr 7, 2025	Food	Juli-ann	Wrong file

Metadata title	Metadata description
Date of Purchase	April 7, 2025
Supplier	Marave
Amount	15,000

File Path
Administrative Records / Acknowledgement Receipt / Supplies / Food

[View](#) [Resupload](#)

Figure 22. Personnel Side – List of Declined Files (Personnel Interface Design)

Figure 22 displays the List of Declined Files section on the personnel interface, showing files that have been reviewed and marked as declined. This section presents a structured view of



all declined documents, including details such as upload date, file name, file owner, metadata title, and metadata description. The interface allows personnel to review declined files and take necessary actions, such as re-uploading with corrections or seeking further clarification.

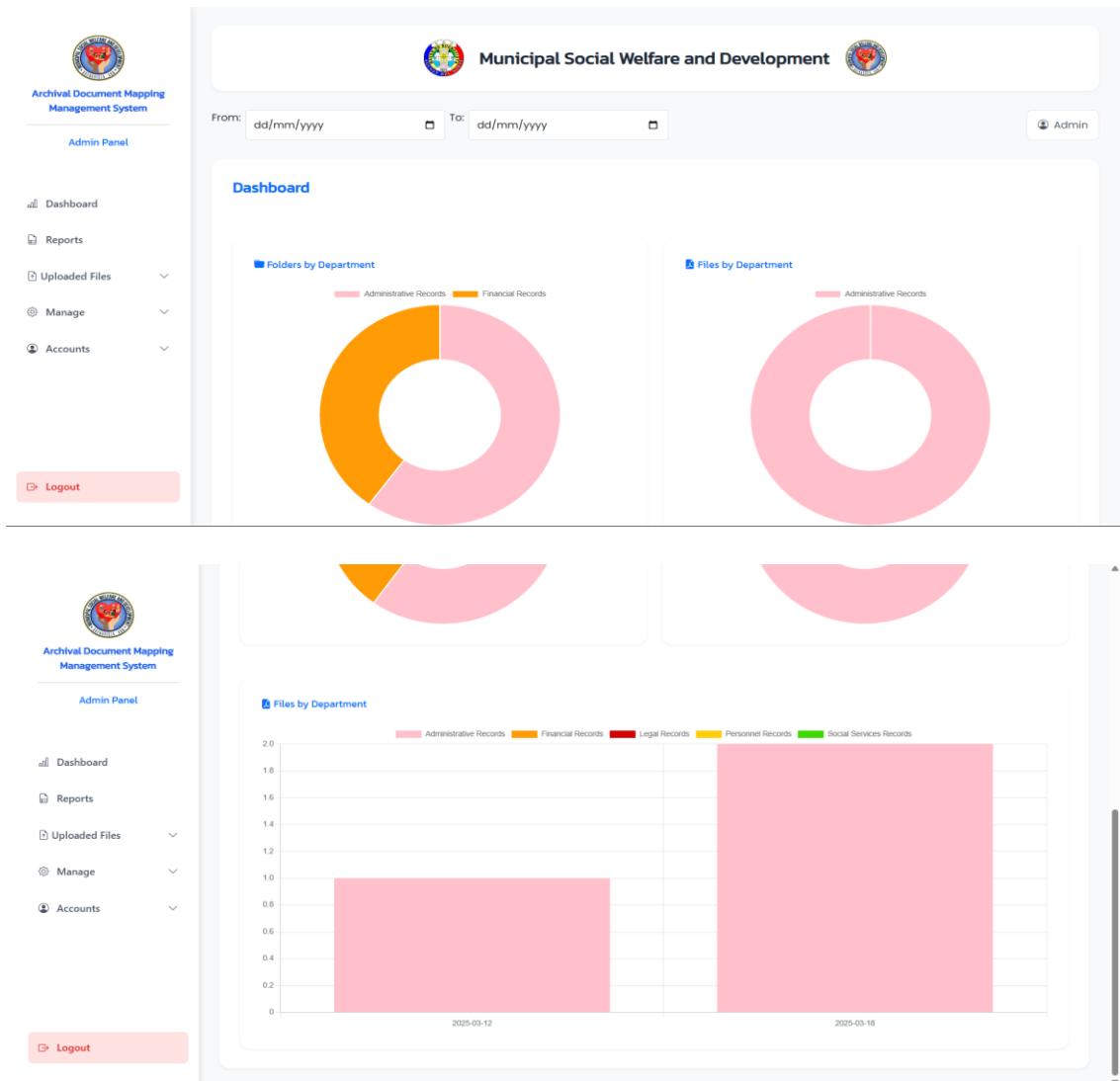


Figure 23. Admin Side – Dashboard (Admin Interface Design)

Figure 23 presents the Dashboard on the admin interface, displaying charts and graphs that provide a visual representation of system data. These graphical elements illustrate folders by department and files by department, offering administrators a clear overview of document



distribution across different departments. This data visualization enhances monitoring and helps in effective document management.

The screenshot shows the Admin Side - Reports section. On the left, there is a sidebar with the title "Archival Document Mapping Management System" and "Admin Panel". The sidebar includes links for Dashboard, Reports, Uploaded Files, Manage, and Accounts. At the bottom of the sidebar is a "Logout" button. The main area has a header "Municipal Social Welfare and Development" with date filters "From: dd/mm/yyyy" and "To: dd/mm/yyyy". Below the header is a "Reports" section with three cards:

- TOTAL USERS:** 4
 - Admin: 1
 - Administrative Records: 2
 - Financial Records: 1
- TOTAL FOLDERS:** 5
 - Administrative Records: 3
 - Financial Records: 2
- TOTAL FILES:** 3
 - Administrative Records: 3

Figure 24. Admin Side – Reports (Admin Interface Design)

Figure 24 presents the Reports section on the admin interface, displaying a structured summary of system data. This section includes total users by department, total folders by department, and total files by department, providing administrators with a clear overview of document and user distribution. The report interface helps in monitoring system usage and supports efficient decision-making for document management.

The screenshot shows the Admin Side - List of Pending Files section. On the left, there is a sidebar with the title "Archival Document Mapping Management System" and "Admin Panel". The sidebar includes links for Dashboard, Reports, Uploaded Files, Manage, and Accounts. At the bottom of the sidebar is a "Logout" button. The main area has a header "Municipal Social Welfare and Development" with date filters "From: dd/mm" and "To: dd/mm", a "Filter" button, a "Clear Filter" button, and a "Print" button. Below the header is a "List of Pending Files" section with a table:

Upload date	File name	Department	File owner	Uploaded by
Mar 18, 2025	Food	Administrative Records	Juli-ann	Winston Lloyd B. Timogan

Below the table is a "Metadata title" section with fields: Date of Purchase (February 27, 2025), Supplier (Marave), and Amount (15000). There is also a "File Path" section with the path: Administrative Records / Acknowledgement Receipt / Supplies / Food. At the bottom are buttons for "View", "Approve", and "Decline".

Figure 25. Admin Side – List of Pending Files (Admin Interface Design)



Figure 25 presents the List of Pending Files section on the admin interface, displaying all files that are awaiting approval. This section provides a structured view, including details such as uploaded date, file name, department, file owner, uploaded by, metadata title, and metadata description. Administrators can review each file and take appropriate actions, such as approving or declining, to ensure efficient document management.

The screenshot shows the Admin Panel interface for the Archival Document Mapping Management System. The left sidebar includes links for Dashboard, Reports, Uploaded Files, Manage, Accounts, and Logout. The main content area displays the 'List of Approved Files' for the Municipal Social Welfare and Development department. The table shows one record:

Upload date	File name	Department	File owner	Uploaded by
Mar 18, 2025	Glass	Administrative Records	Elmerito	Winston Lloyd B. Timogan

Below the table, detailed metadata information is shown:

Metadata title	Metadata description
Date of Purchase	February 27, 2025
Supplier	Jr Glass
Amount	10500

At the bottom, the File Path is listed as 'Administrative Records / Acknowledgement Receipt / Supplies / Glass' with a 'View' button.

Figure 26. Admin Side – List of Approved Files (Admin Interface Design)

Figure 26 presents the List of Approved Files section on the admin interface, displaying all files that have been reviewed and approved. This section provides a structured view, including details such as uploaded date, file name, department, file owner, uploaded by, metadata title, and metadata description. Administrators can access and manage approved files for monitoring, compliance, and record-keeping purposes, ensuring efficient document organization within the system.



Upload date	File name	Department	File owner	Uploaded by	Remarks
Mar 12, 2025	askdh	Administrative Records	oihih	Winston Lloyd B. Timogon	asd

Metadata title

Date of Purchase	ih
Supplier	ih
Amount	h

File Path

Administrative Records / Acknowledgement Receipt / Supplies / askdh

Figure 27. Admin Side – List of Declined Files (Admin Interface Design)

Figure 27 presents the List of Declined Files section on the admin interface, displaying all files that have been reviewed and marked as declined. This section provides a structured view, including details such as uploaded date, file name, department, file owner, uploaded by, remarks, metadata title, and metadata description. Administrators can review declined files, provide feedback through remarks, and guide personnel on necessary actions for resubmission, ensuring a transparent and efficient document management process.

Department Name	Color	Action
Administrative Records	#FF0000	Edit
Financial Records	#FFB6C1	Edit
Legal Records	#D0E0E0	Edit
Personnel Records	#FFCCBC	Edit
Social Services Records	#4DB6AC	Edit

Figure 28. Admin Side – List Departments (Admin Interface Design)

Figure 28 presents the List of Departments section on the admin interface, displaying all departments within the system. This section provides a structured view, including details such as



department name, assigned color, and an action button for editing. Administrators can modify department details as needed, ensuring an organized and customizable department management system.

Figure 29. Admin Side – Add Department (Admin Interface Design)

Figure 29 presents the Add Department section on the admin interface, allowing administrators to create new departments within the system. This section includes input fields for department name and assigned color, along with an action button to save the new department. This feature ensures an organized and customizable department management system.

Figure 30. Admin Side – Display Folders (Admin Interface Design)



Figure 30 presents the Display Folders section on the admin interface, showing a structured view of all folders within the system. This section includes details such as folder name, assigned department, and available actions for managing folders. Administrators can efficiently organize, view, and manage folders, ensuring proper document storage and accessibility.

A screenshot of the Admin Panel for the Archival Document Mapping Management System. The left sidebar shows navigation options like Dashboard, Reports, Uploaded Files, Manage, and Accounts. The main area displays a list of folders under 'Folders'. One folder, 'Administrative Records 2025', is highlighted. A modal window titled 'Folder Details' is open, showing the following information:

Folder Name:	Administrative Records
Year:	2025
Row Number:	
Column Number:	
Department:	Administrative Records
Created At:	March 12, 2025

With 'Edit' and 'Delete' buttons at the bottom.

Figure 31. Admin Side – Folder Details (Admin Interface Design)

Figure 31 presents the Folder Details section on the admin interface, displaying specific information about a selected folder. This section includes details such as folder name, year, row number, column number, department, and created at. Additionally, it features action buttons for editing and deleting the folder, allowing administrators to update folder information or remove folders as needed for efficient document management.



Figure 32. Admin Side – Display Files Card Style (Admin Interface Design)

Figure 32 presents the Display Files Card Style section on the admin interface, showcasing uploaded files in a visually organized card-style format. Each card includes an action button (three-dot icon) to view file details or perform other management actions, ensuring an efficient and structured document management system.

File name	Upload date	File owner	Uploaded by	Date of Purchase	Supplier	Amount	File
Glass	Mar 18, 2025	Elmerto	Winston Lloyd B. Timogan	February 27, 2025	Jr Glass	10500	

Figure 33. Admin Side – Display Files Table Style (Admin Interface Design)

Figure 33 presents the Display Files Table Style section on the admin interface, showcasing uploaded files in a structured tabular format. Each row in the table includes details such as file



name, uploaded date, department, file owner, uploaded by, metadata title, and metadata description ensuring an efficient and organized document management system.

Figure 34. Admin Side – View PDF File (Admin Interface Design)

Figure 34 presents the View PDF File section on the admin interface, allowing administrators to open and review PDF documents directly within the system. The feature ensures seamless document access and management for administrators.

Figure 35. Admin Side – Create Parent Folder (Admin Interface Design)



Figure 35 presents the Create Parent Folder section on the admin interface, allowing administrators to create a new parent folder for document organization. This section includes input fields for folder name, year, and assigned department, ensuring proper categorization and structured storage within the system.

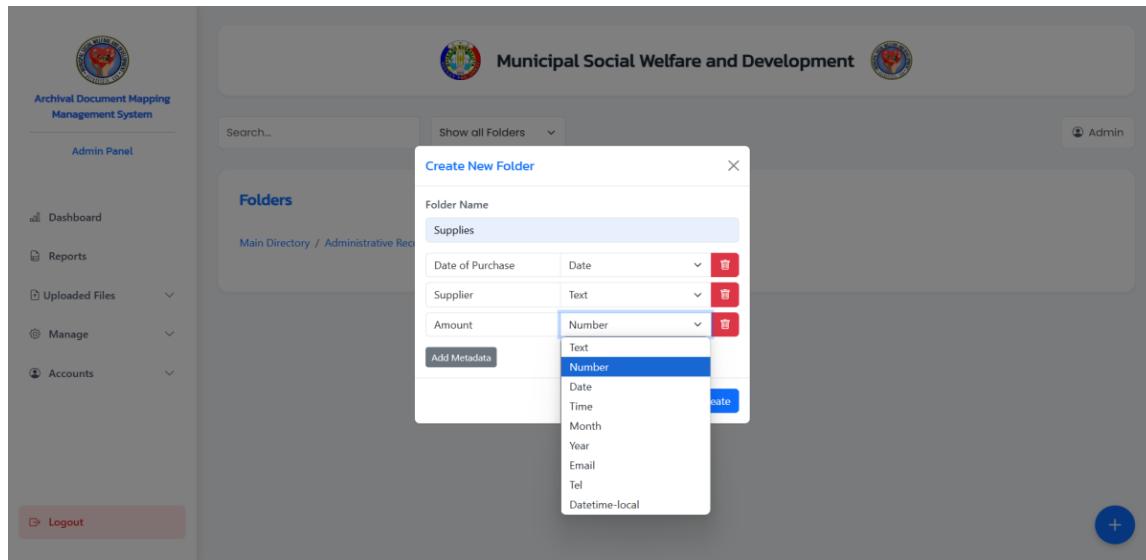


Figure 36. Admin Side – Create Subfolder and Set Metadata Title (Admin Interface Design)

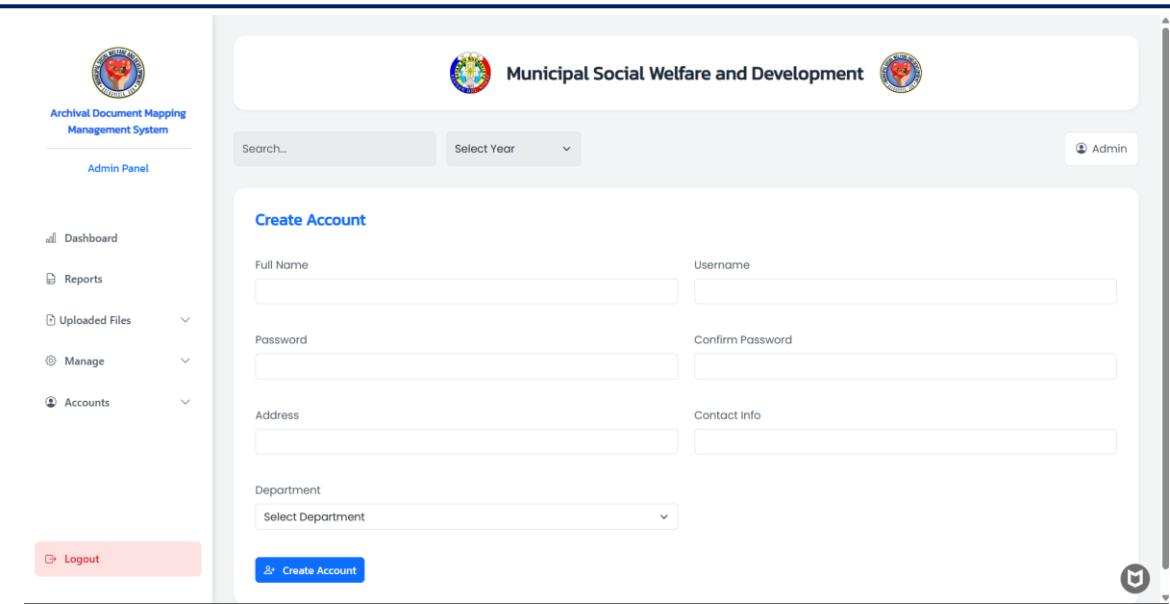
Figure 36 presents the Create Subfolder and Set Metadata Title section in the admin interface, enabling administrators to efficiently organize documents by creating subfolders within a parent folder and defining metadata titles. This section includes input fields for specifying the subfolder name and metadata title, along with options to set the input type, ensuring data consistency and accuracy. Administrators can select from predefined input types such as text, number, date, time, month, year, email, tel, datetime-local, allowing for a more structured and flexible document management system.

A screenshot of the Admin Side - Display All Accounts section. The interface includes a header with the college's name and a search bar. Below is a table showing three accounts with columns for Name, Username, Address, Contact No., Department, Change Password, and Actions. Buttons for Change Password, Edit, and Disable are visible for each account. A sidebar on the left shows navigation links like Dashboard, Reports, Uploaded Files, Manage, and Accounts, with Accounts currently selected. A Logout button is at the bottom left.

NAME	USERNAME	ADDRESS	CONTACT No.	DEPARTMENT	CHANGE PASSWORD	ACTIONS
James Nolie Piel	james	Buenavista Agusan del Norte	09553556487	FINANCIAL RECORDS	Change Password	Edit Disable
Juli-Ann Echalico	juli-ann	Guinabsan Buenavista Agusan Del Norte	09553556487	ADMINISTRATIVE RECORDS	Change Password	Edit Disable
Winston Lloyd B. Timogan	timogan001	Culit Nasipit Agusan del Norte	09553556487	ADMINISTRATIVE RECORDS	Change Password	Edit Disable

Figure 37. Admin Side – Display All Accounts (Admin Interface Design)

Figure 37 presents the Display All Accounts section on the admin interface, listing all personnel accounts within the system. This section includes details such as name, username, address, contact number, and department. Additionally, it provides options to change the password and action buttons for editing or disabling accounts, ensuring effective user management and system security.



The image shows the 'Create Account' section of the Admin Side interface. On the left, there's a sidebar with the college's logo at the top, followed by the text 'Archival Document Mapping Management System' and 'Admin Panel'. Below this are several menu items: 'Dashboard', 'Reports', 'Uploaded Files', 'Manage', and 'Accounts'. At the bottom of the sidebar is a red 'Logout' button. The main content area has a header with the college's logo, the text 'Municipal Social Welfare and Development', and another small logo. It includes search and year selection fields, and an 'Admin' user icon. The 'Create Account' form contains fields for 'Full Name' (with a placeholder 'John Doe'), 'Username' (placeholder 'johndoe'), 'Password' (placeholder 'password123'), 'Confirm Password' (placeholder 'password123'), 'Address' (placeholder '123 Main Street'), 'Contact Info' (placeholder '09123456789'), 'Department' (a dropdown menu with 'Select Department' placeholder), and a blue 'Create Account' button.

Figure 38. Admin Side – Register Personnel Account (Admin Interface Design)

Figure 38 presents the Register Personnel Account section on the admin interface, allowing administrators to create new personnel accounts within the system. This section includes input fields for fullname, username, password, confirm password, address, contact number, and department. An action button is provided to save the new account, ensuring secure and efficient personnel registration.



3.8 Software Platforms, Development Environment and Tools

Table 1

Software Requirements

Components	Specification	Usage
Internet Browser	Any	Internet browsers make web access and rendering easier. They load web pages and enable users to interact with the system. After a successful login, the browser connects users to the ADMMS (Archival and Document Mapping Management System), granting access to its functionalities, including uploading, mapping, and viewing documents.
Front-end	HTML	HTML provides structural support for web pages. It organizes content elements such as forms, images, and text. In ADMMS, it defines the layout and structure for forms and user interfaces.
	CSS	CSS enhances the appearance of HTML elements, allowing customization of fonts, colors, layout, and spacing. CSS is critical for creating responsive and visually appealing designs in ADMMS, such as for the layout of the home, buttons, and document display pages.
	Bootstrap	Bootstrap is an open-source front-end framework that simplifies creating responsive and user-friendly web applications. For your system, Web-Based Archival and Document Mapping for Buenavista's Municipal Social Welfare Services, it ensures a professional, intuitive interface and seamless access across devices.
	JavaScript	JavaScript adds interactive capabilities to web pages, allowing dynamic user input handling. In ADMMS, JavaScript is used for real-time data display on the main page.
	jQuery	jQuery is a lightweight JavaScript library that simplifies dynamic content updates and enhances interactivity. For your system, it ensures a responsive and seamless user experience.
Back-end	MySQL (Database)	MySQL is an open-source relational database management system. It provides efficient data management tools for storing and retrieving documents. In ADMMS, MySQL is used to manage documents, user



	PHP	information, and access control. phpMyAdmin is used for database administration.
Server	Apache	PHP is a server-side scripting language used to generate dynamic content. In ADMMS, PHP serves as the bridge between the front-end and the database, handling requests such as document uploads and create folders.

3.9 Hardware Requirements

Table 2
Hardware Requirements

Components	Specification	Usage
Device	RAM (8GB or 16GB)	Electronic devices such as computers and laptops RAM is used to quickly access stored data, such as documents. More RAM ensures smoother performance, faster search, and handling of multiple users simultaneously.
	Processor	The processor handles all document processing tasks, such as indexing and mapping. Multiple cores are essential for parallel processing, allowing multiple users to interact with the system simultaneously.
	HDD SSD	SSDs are recommended for fast data retrieval, such as accessing and uploading documents, searching folder to map. Faster read/write speeds ensure smoother performance for large document repositories. HDDs can be used for archival purposes.
Printer	Any Ink Jet Printer Units	While not directly tied to the system's core functions, an inkjet printer may be used for printing and scanning documents. It is suitable for routine office tasks, producing high-quality prints for document archiving.



CHAPTER IV

SOFTWARE DEVELOPMENT AND TESTING

This chapter covers the development and testing phases of the Web-Based Archival and Document Mapping System for Buenavista's Municipal Social Welfare Services, detailing the methodologies, frameworks, and tools utilized to build and optimize the system. It outlines the steps taken to ensure the system's functionality, user-friendliness, and efficiency in managing archival records and mapping documents. Furthermore, this chapter highlights the testing strategies implemented to validate the system's performance, security, and reliability, ensuring it meets the needs of the Municipal Social Welfare Services.

4.1 Development Process

The systematic software development process of the Web-based Archival and Document Mapping for Buenavista's Municipal Social Welfare Services is illustrated through the input-process-output diagram shown in Figure #.

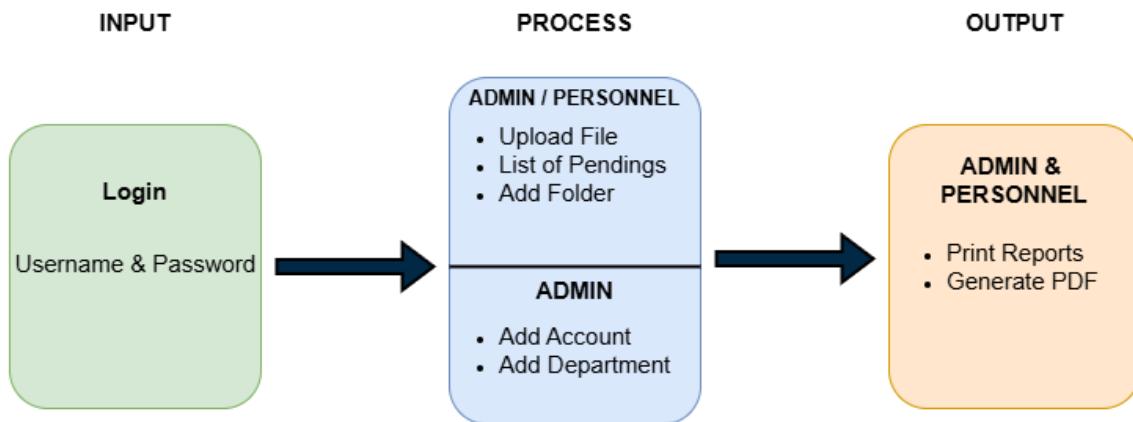


Figure 39. Input-Process-Output Diagram

Figure 39 shows the input-process-output diagram and illustrates the orderly software development process of the Archival Document Mapping Management System (ADMMS).



LOG IN

Administrators and personnel have access to the system's crucial functions. The login mechanism is a secure authentication process that allows them to access the system using a unique username and password. Developed to ensure the protection of user credentials, it enforces strict access controls and prevents unauthorized access to the system.

ADMINISTRATIVE & PERSONNEL

The process phase served as the operational core of the system, ensuring efficient document management and structured workflow within the MSWD office. As the system administrator, the MSWD head had the capability to upload files, add user accounts, oversee and manage pending requests, create folders, and assign departments. These functions allowed the administrator to maintain organized records, regulate user access, and facilitate smooth operations across different departments. By handling account creation and department assignments, the system administrator ensured that only authorized personnel could access specific files, enhancing data security and administrative control.

Meanwhile, MSWD personnel, including social workers, case officers, and clerks, had essential functionalities that supported their daily tasks. They were able to upload files, create folders for organizing documents, and track pending requests, ensuring that all records and applications were processed efficiently. The ability to monitor pending requests helped personnel stay updated on the status of submitted documents, minimizing delays and improving response times.

GENERATE PDF & PRINT

The Generate PDF & Print feature allows the MSWD head, as the system administrator, to oversee the creation, validation, and approval of structured reports, ensuring compliance with government record-keeping policies. MSWD personnel, including social workers, case officers,



and administrative staff, can generate and retrieve reports related to beneficiary records, financial assistance, and case progress. These reports can be exported in standardized formats, such as PDF, for documentation, submission, and future reference, ensuring data accuracy, accessibility, and proper institutional archiving.

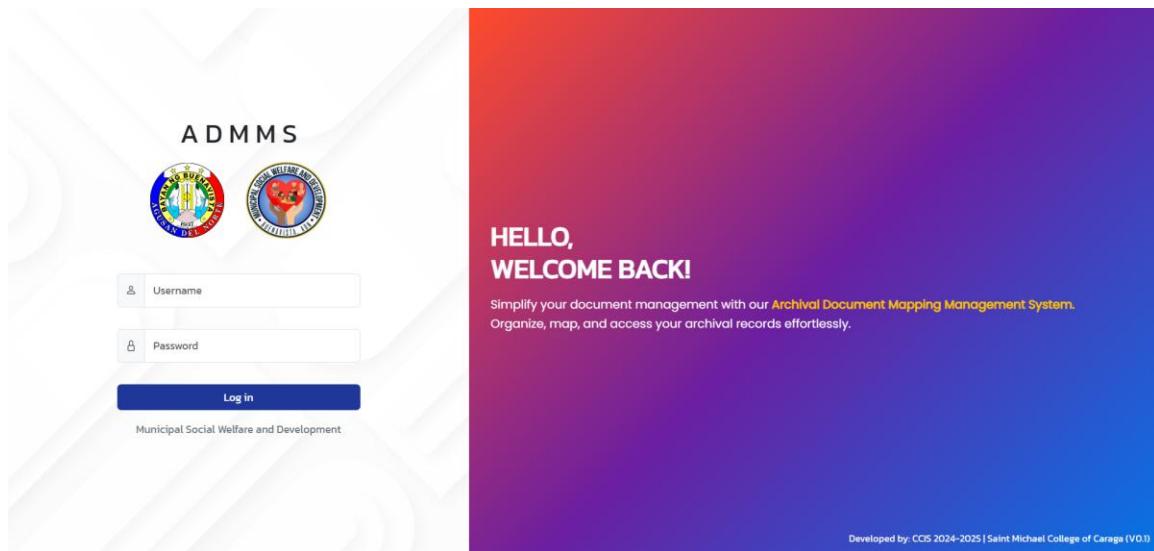


Figure 40. Admin/Personnel Login Page

Figure 40 shows the Admin and Personnel login page. This UI will be the first thing they see when accessing the ADMMS. The Username and Password input is where they will input the assigned Username and Password for their account.

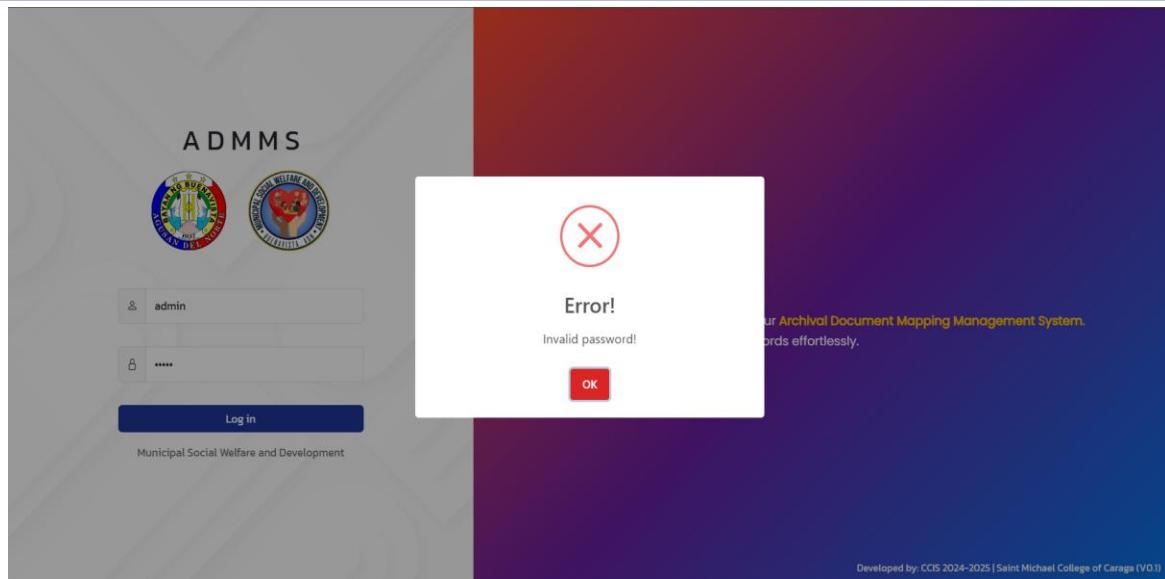


Figure 41 Admin/Personnel – Login Error Notification

Figure 41 displays an error modal when the Username and Password do not exist or match in the database. The system will let the user know they input an incorrect Username or Password.

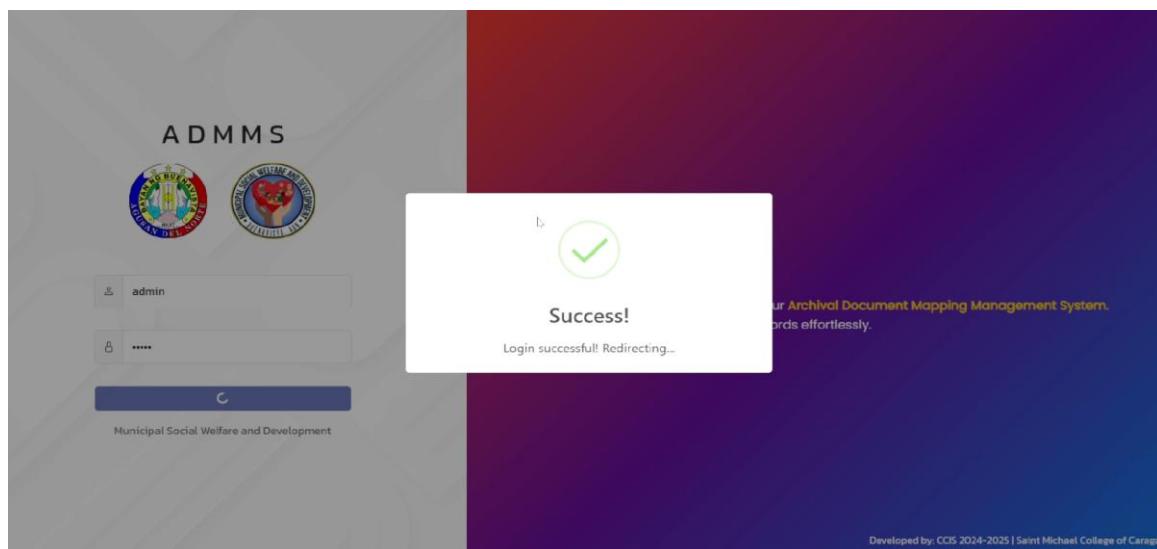


Figure 42. Admin/Personnel – Login Successful Notification

Figure 42 displays a successful modal showing the user with the Username and Password that has been found. The system will then redirect the admin to the dashboard and the personnel to the homepage.



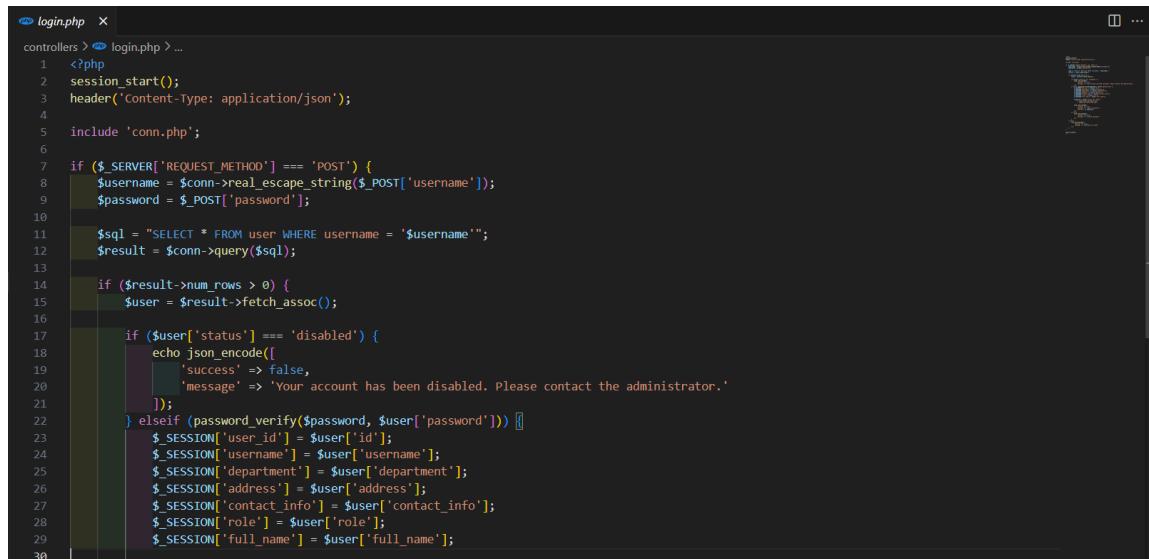
Figure 43. Code snippet for admin/personnel – Login Page

Figure 43 shows the code snippet for the admin/personnel login page. The `$('#loginForm').on('submit', function (e) { ... })` function handles the login form submission process, preventing the default form submission to allow for AJAX-based authentication. When the user submits the form, it retrieves the submit button and its elements—the `login-text` span, which displays the "Login" text, and the `spinner-border`, a loading animation. To enhance user experience, the function hides the login text and displays the spinner, signaling that the system is processing the request. Additionally, it disables the submit button to prevent multiple submissions, ensuring that users do not accidentally send duplicate



login requests while waiting for a response. This improves responsiveness and avoids unnecessary server load.

The AJAX request inside this function sends the serialized form data to controllers/login.php using a POST request and expects a JSON response. If the login is successful, the function displays a SweetAlert2 success popup with a message and a short delay before redirecting the user to the URL specified in response.redirect. If the login fails, it displays an error message using SweetAlert2, notifying the user of invalid credentials or another login issue. The function then restores the login button's original state by re-enabling it, showing the login text, and hiding the spinner. In the case of a server error, an additional error-handling function retrieves the response and displays a detailed error message, if available, ensuring transparency and assisting in debugging. This comprehensive approach ensures a smooth and informative login experience while minimizing user frustration.



A screenshot of a code editor window titled "login.php". The code is a PHP script for handling user login. It starts with basic setup like session_start() and header('Content-Type: application/json'). It checks if the request method is POST. It then retrieves the username and password from the \$_POST array. A SQL query is constructed to select all columns from the "user" table where the username matches the input. The result of this query is stored in \$result. If there are more than zero rows, it fetches the first row as an associative array and stores it in \$user. It then checks if the account is disabled. If it is, it encodes a JSON response with "success" set to false and a message about the account being disabled. If the account is enabled, it performs a password verification. If successful, it sets session variables for user_id, username, department, address, contact_info, role, and full_name. The code uses echo json_encode([...]) to output the JSON response.

```
<?php
session_start();
header('Content-Type: application/json');

include 'conn.php';

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $username = $conn->real_escape_string($_POST['username']);
    $password = $_POST['password'];

    $sql = "SELECT * FROM user WHERE username = '$username'";
    $result = $conn->query($sql);

    if ($result->num_rows > 0) {
        $user = $result->fetch_assoc();

        if ($user['status'] === 'disabled') {
            echo json_encode([
                'success' => false,
                'message' => 'Your account has been disabled. Please contact the administrator.'
            ]);
        } elseif (password_verify($password, $user['password'])) {
            $_SESSION['user_id'] = $user['id'];
            $_SESSION['username'] = $user['username'];
            $_SESSION['department'] = $user['department'];
            $_SESSION['address'] = $user['address'];
            $_SESSION['contact_info'] = $user['contact_info'];
            $_SESSION['role'] = $user['role'];
            $_SESSION['full_name'] = $user['full_name'];
        }
    }
}
```

A screenshot of a code editor showing a PHP script named "login.php". The code handles user authentication against a MySQL database. It uses prepared statements to prevent SQL injection. The script checks if the user role is 'Admin'. If successful, it outputs a JSON response indicating success and the redirect URL. If failed, it indicates invalid password or username. The script then closes the database connection.

```
30 |     $redirect = $user['role'] === 'Admin';
31 |     Output one or more strings
32 |     echo( string $arg1 [, string $... ]): void
33 |
34 |     echo json_encode([
35 |         'success' => true,
36 |         'message' => 'Login successful!',
37 |         'redirect' => $redirect
38 |     ]);
39 | } else {
40 |     echo json_encode([
41 |         'success' => false,
42 |         'message' => 'Invalid password!'
43 |     ]);
44 | }
45 | } else {
46 |     echo json_encode([
47 |         'success' => false,
48 |         'message' => 'Username not found!'
49 |     ]);
50 | }
51 |
52 |
53 |
54 | $conn->close();
55 | ?>
```

Figure 44. Code Snippet for Admin/Personnel Login Page – Login.php

Figure 44 shows the code snippet for the login page backend “login.php” script handles user authentication by verifying credentials against a MySQL database, starting a session, and returning a JSON response. It prevents SQL injection using `real_escape_string()` but should use prepared statements for better security. Passwords are verified using `password_verify()`, ensuring they were stored securely with `password_hash()`. The script checks if an account is disabled and sets session variables upon successful login. However, the `$redirect` logic is incomplete and should define actual URLs. Enhancements like HTTPS, login attempt limits, and improved error handling would strengthen security.

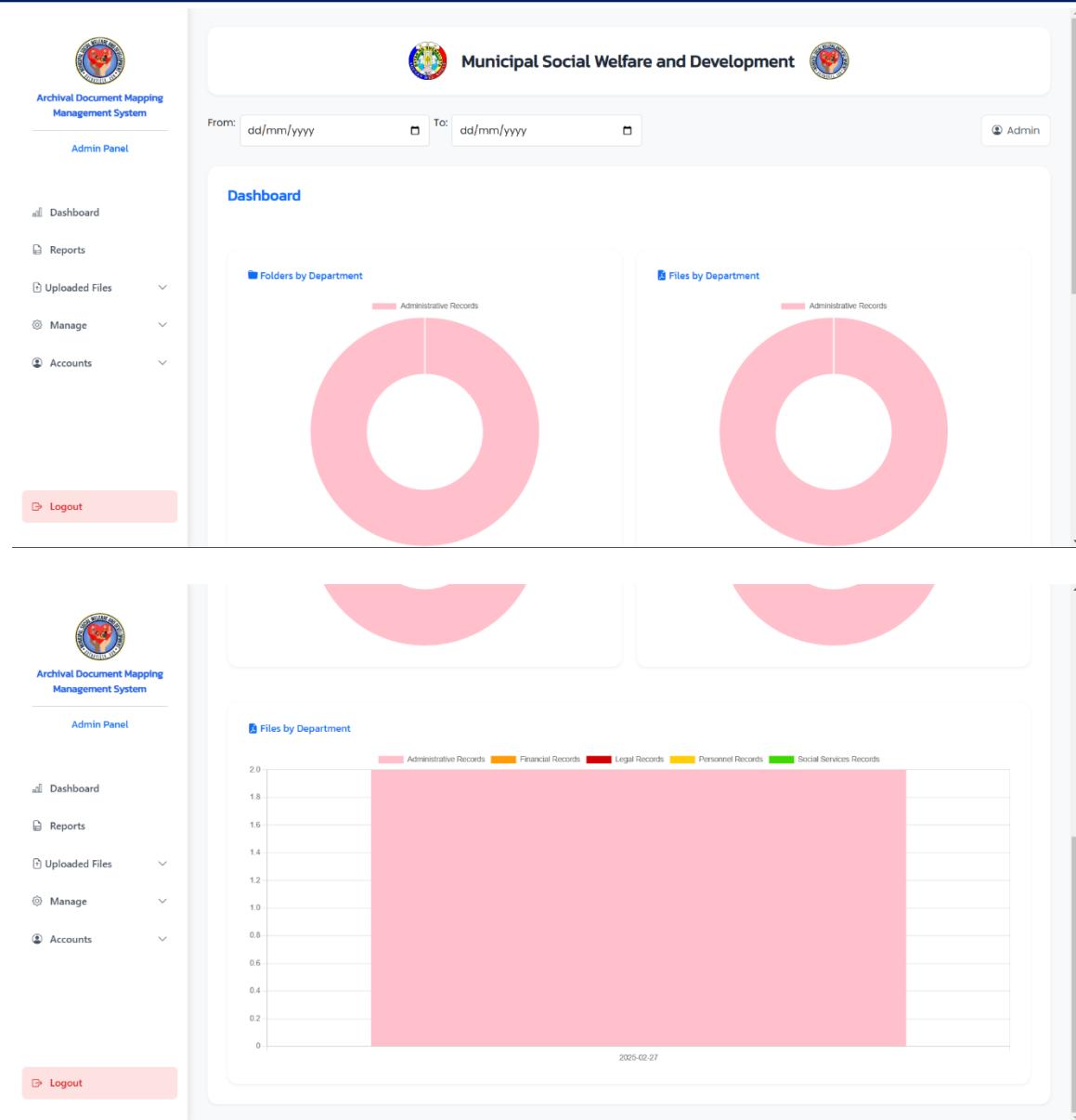


Figure 45. Admin – Dashboard

Figure 45 shows the Admin Panel Dashboard organizes and categorizes uploaded files by department, displaying them in folders with progress indicators showing storage percentages. Admin can refine searches using a date filter, while the system automatically classifies files into appropriate categories within each department. A bar graph visually represents the number of files in each category, enhancing data tracking and accessibility.



```
1 document.addEventListener('DOMContentLoaded', function () {
2
3     let charts = {};
4
5     function fetchChartData(fromDate = '', toDate = '') {
6         fetch(`./controllers/chart.php?fromDate=${fromDate}&toDate=${toDate}`)
7             .then(response => response.json())
8             .then(data => {
9                 renderDoughnutChart('foldersDoughnutChart', 'Folders by Department', data.folders, data.departments);
10                renderDoughnutChart('filesDoughnutChart', 'Files by Department', data.files, data.departments);
11                renderStackedBarChart('stackedBarChart', data.stackeds, data.departments);
12            })
13         .catch(error => console.error('Error fetching chart data:', error));
14     }
15
16     function destroyChart(chartId) {
17         if (charts[chartId]) {
18             charts[chartId].destroy();
19             charts[chartId] = null;
20         }
21     }
22
23     function renderDoughnutChart(canvasId, title, dataset, departments) {
24         const ctx = document.getElementById(canvasId).getContext('2d');
25         destroyChart(canvasId);
26
27         const departmentColors = {};
28         departments.forEach(dept => {
29             departmentColors[dept.department_name] = `#${dept.folder_color}`;
30         });
31
32         const dataCount = {};
33         dataset.forEach(item => {
34             dataCount[item.department] = (dataCount[item.department] || 0) + 1;
35         });
36
37         charts[canvasId] = new Chart(ctx, {
38             type: 'doughnut',
39             data: {
40                 labels: Object.keys(dataCount),
41                 datasets: [
42                     {
43                         data: Object.values(dataCount),
44                         backgroundColor: Object.keys(dataCount).map(dept => departmentColors[dept] || '#ccc')
45                     }
46                 ],
47                 options: { responsive: true, plugins: { legend: { position: 'top' } } }
48             });
49
50         function renderStackedBarChart(canvasId, dataset, departments) {
51             const ctx = document.getElementById(canvasId).getContext('2d');
52             destroyChart(canvasId);
53
54             const departmentColors = {};
55             departments.forEach(dept => {
56                 departmentColors[dept.department_name] = `#${dept.folder_color}`;
57             });
58
59             const groupedData = {};
60             dataset.forEach(item => {
61                 if (!groupedData[item.upload_date]) groupedData[item.upload_date] = {};
62                 groupedData[item.upload_date][item.department] = item.count;
63             });
64
65             const labels = Object.keys(groupedData);
66             const datasets = departments
67                 .filter(dept => dept.department_name !== 'Admin')
68                 .map(dept => ({
69                     label: dept.department_name,
70                     backgroundColor: `#${dept.folder_color}`,
71                     data: labels.map(date => groupedData[date][dept.department_name] || 0)
72                 }));
73         }
74     }
75 }
```



```

73
74
75     charts[canvasId] = new Chart(ctx, {
76       type: 'bar',
77       data: { labels, datasets },
78       options: { responsive: true, scales: { x: { stacked: true }, y: { stacked: true } } }
79     });
80   }
81
82   function updateFilterButtons() {
83     const fromDate = document.getElementById('fromDate').value;
84     const toDate = document.getElementById('toDate').value;
85     const filterButton = document.getElementById('filterButton');
86     const clearFilterButton = document.getElementById('clearFilterButton');
87
88     if (fromDate && toDate) {
89       filterButton.style.display = 'inline-block';
90       clearFilterButton.style.display = 'inline-block';
91     } else {
92       filterButton.style.display = 'none';
93       clearFilterButton.style.display = 'none';
94     }
95   }
96
97   document.getElementById('fromDate').addEventListener('change', updateFilterButtons);
98   document.getElementById('toDate').addEventListener('change', updateFilterButtons);
99
100  document.getElementById('filterButton').addEventListener('click', function () {
101    const fromDate = document.getElementById('fromDate').value;
102    const toDate = document.getElementById('toDate').value;
103    fetchChartData(fromDate, toDate);
104  });
105
106  document.getElementById('clearFilterButton').addEventListener('click', function () {
107    document.getElementById('fromDate').value = '';
108    document.getElementById('toDate').value = '';
109    updateFilterButtons();
110    fetchChartData();
111  });
112
113  updateFilterButtons();
114  fetchChartData();
115 });

```

Figure 46. Code snippet for pie and bar graph – chart

Figure 46 shows the code for `fetchChartData(fromDate, toDate)` function is responsible for fetching statistical data from the server using an AJAX request to `chart.php`, which retrieves document and file distribution data based on optional date filters. If date parameters are provided, they are appended to the URL; otherwise, an unfiltered request is made. Upon a successful response, the function processes the received JSON data and calls `renderDoughnutChart()` twice—once for folders and once for files—followed by `renderStackedBarChart()` to visualize file uploads across departments over time. This function ensures that charts dynamically update when the user applies a filter or clears it, providing an interactive data representation. If the fetch request encounters an error, it is logged to the console to aid debugging.



The `destroyChart(chartId)` function ensures that old chart instances are properly destroyed before creating new ones, preventing memory leaks and duplicate charts. It checks if a chart exists in the `charts` object and, if so, calls `.destroy()` on it before setting the corresponding entry to null. This is crucial for maintaining the integrity of the visualization when data updates. The `renderDoughnutChart(canvasId, title, dataset, departments)` function creates a Doughnut chart for folder and file distributions per department. It first initializes a color mapping by extracting department colors from the provided data. Then, it counts occurrences of department-related entries in the dataset and configures a `Chart.js` Doughnut chart with labels, background colors, and responsive settings. The function ensures that different departments have distinct colors and dynamically generates the chart upon receiving new data.

The `renderStackedBarChart(canvasId, dataset, departments)` function generates a stacked bar chart that visualizes document uploads over time. It follows a similar color assignment process to the doughnut chart but structures data differently by grouping file counts by department and upload date. The function filters out the "Admin" department to maintain meaningful visual comparisons among other departments. Using `Chart.js`, it configures the bar chart with stacked bars on both x and y axes, allowing users to see total uploads while still distinguishing departmental contributions. This function plays a key role in tracking trends in document submissions and identifying peak activity periods.

The `updateFilterButtons()` function manages the visibility of the filter buttons (`#filterButton` and `#clearFilterButton`) based on user input in the date fields. If both `fromDate` and `toDate` have values, the buttons are displayed; otherwise, they are hidden. This improves the user experience by ensuring that users only see relevant actions when applicable. The function is executed upon page load and whenever the date input fields change.



The event listeners attached to `fromDate` and `toDate` call this function dynamically, ensuring real-time updates.

The event listeners for `filterButton` and `clearFilterButton` enable filtering functionality. When `filterButton` is clicked, it retrieves the selected dates and calls `fetchChartData(fromDate, toDate)` to update the charts with the filtered data. Conversely, when `clearFilterButton` is clicked, it resets the date inputs, updates button visibility, and calls `fetchChartData()` without filters, restoring the default chart state. Lastly, `updateFilterButtons()` and `fetchChartData()` are called upon page load to set the initial button states and fetch unfiltered data, ensuring that users see relevant statistics immediately. This cohesive structure allows for seamless interaction and an intuitive user experience.

A screenshot of a code editor window titled "conn.php". The code is a PHP script for establishing a database connection. It includes variables for host, username, password, and database, and a conditional block to handle connection errors by encoding and outputting JSON data.

```
conn.php
controllers > conn.php > ...
1  <?php
2  // database connection
3  $host = "localhost";
4  $username = "root";
5  $password = "";
6  $database = "adimms";
7
8  $conn = new mysqli($host, $username, $password, $database);
9
10 if ($conn->connect_error) {
11     die(json_encode([
12         'success' => false,
13         'message' => "Connection failed: " . $conn->connect_error
14     ]));
15 }
16 ?>
```

Figure 47. Code Snippet for dashboard – conn.php

Figure 47 shows the code for establishing the connection between the front end and the database to send and retrieve data from the database.



```

chart.php x
controllers > chart.php > ...
1  <?php
2  // chart functions (admin | dashboard)
3  require 'conn.php';
4
5  $fromDate = isset($_GET['fromDate']) ? $_GET['fromDate'] : null;
6  $toDate = isset($_GET['toDate']) ? $_GET['toDate'] : null;
7
8  // empty chart arrays (default)
9  $data = [
10    'departments' => [],
11    'files' => [],
12    'folders' => [],
13    'stackedfiles' => []
14];
15
16 $deptQuery = "SELECT department_name, folder_color FROM department";
17 $result = $conn->query($deptQuery);
18 while ($row = $result->fetch_assoc()) {
19    $data['departments'][] = $row;
20 }
21
22 $fileQuery = "SELECT department, upload_date FROM files";
23 if ($fromDate && $toDate) {
24    $fileQuery .= " WHERE DATE(upload_date) BETWEEN '$fromDate' AND '$toDate'";
25 }
26 $result = $conn->query($fileQuery);
27 while ($row = $result->fetch_assoc()) {
28    $data['files'][] = $row;
29 }

30
31 $folderQuery = "SELECT department, created_at FROM folders";
32 if ($fromDate && $toDate) {
33    $folderQuery .= " WHERE DATE(created_at) BETWEEN '$fromDate' AND '$toDate'";
34 }
35 $result = $conn->query($folderQuery);
36 while ($row = $result->fetch_assoc()) {
37    $data['folders'][] = $row;
38 }
39
40 $stackedQuery = "SELECT department, DATE(upload_date) as upload_date, COUNT(*) as count FROM files";
41 if ($fromDate && $toDate) {
42    $stackedQuery .= " WHERE DATE(upload_date) BETWEEN '$fromDate' AND '$toDate'";
43 }
44 $stackedQuery .= " GROUP BY department, upload_date ORDER BY upload_date";
45 $result = $conn->query($stackedQuery);
46 while ($row = $result->fetch_assoc()) {
47    $data['stackedFiles'][] = $row;
48 }

49
50 header('Content-Type: application/json');
51 echo json_encode($data);
52

```

Figure 48. Code Snippet for Admin Dashboard – chart.php

Figure 48 show the code the “chart.php” script retrieves and structures data for visualization in the Admin Panel, fetching department details, file uploads, folder creation dates, and stacked file counts while supporting date-based filtering. It initializes an empty data array, executes SQL queries to gather relevant information, and populates the array with department names, folder colors, file upload dates, and categorized folder details. The script also groups file uploads by department and date to generate statistical insights. Finally, it returns the collected data in JSON format for easy integration with charts and graphs. For security, using prepared statements and validating user inputs would enhance protection against SQL injection.



The screenshot shows the Admin - Reports section of the Archival Document Mapping Management System. On the left, there's a sidebar with a logo and navigation links: Dashboard, Reports, Uploaded Files, Manage, Accounts, and Logout. The main area has a header for 'Municipal Social Welfare and Development'. It features three cards: 'TOTAL USERS' (4, Admin, Administrative Records, Financial Records), 'TOTAL FOLDERS' (3, Administrative Records), and 'TOTAL FILES' (2, Administrative Records).

Figure 49. Admin – Reports

Figure 49 shows the Reports Section includes a date filter at the top, allowing users to refine data based on a selected timeframe. It displays the total number of users, providing insight into system activity and engagement. Additionally, it shows the total number of folders created and the total files uploaded, offering a clear overview of document organization.

```

1  $(document).ready(function () {
2      function getQueryParams() {
3          let params = {};
4          window.location.search.substring(1).split('&').forEach(function (pair) {
5              let [key, value] = pair.split('=');
6              params[decodeURIComponent(key)] = decodeURIComponent(value);
7          });
8          return params;
9      }
10
11     function checkdates() {
12         if ($('#fromDate').val() && $('#toDate').val()) {
13             $('#filterButton').show();
14             $('#clearFilterButton').show();
15         } else {
16             $('#filterButton').hide();
17             $('#clearFilterButton').hide();
18         }
19     }
20
21     let params = getQueryParams();
22     if (params.fromDate) {
23         $('#fromDate').val(params.fromDate);
24     }
25     if (params.toDate) {
26         $('#toDate').val(params.toDate);
27     }
28
29     $('#fromDate, #toDate').on('change', function () {
30         checkdates();
31     });
32
33     $('#filterButton').on('click', function () {
34         let fromDate = $('#fromDate').val();
35         let toDate = $('#toDate').val();
36         window.location.href = '?fromDate=' + encodeURIComponent(fromDate) + '&toDate=' + encodeURIComponent(toDate);
37     });
38 }
```



```
38
39     $('#clearFilterButton').on('click', function () {
40         window.location.href = window.location.pathname;
41     });
42
43     checkDates();
44 });


```

Figure 50. Code snippet for card – reports

Figure 50 shows the code for `getqueryParams()` function extracts query parameters from the URL and returns them as a key-value object, allowing the script to dynamically retrieve and use parameters such as `fromDate` and `toDate`. It does this by splitting the URL's search string into individual key-value pairs, decoding them to handle special characters, and storing them in an object for easy access. This function is particularly useful for maintaining filter states across page reloads. The `checkDates()` function is responsible for controlling the visibility of the filter buttons (`#filterButton` and `#clearFilterButton`) based on the presence of values in the date input fields. If both the "From Date" and "To Date" fields have values, the buttons are displayed; otherwise, they remain hidden. This ensures a cleaner UI by only showing filter-related actions when they are applicable. The function is triggered upon page load and whenever date input fields change.

When the script initializes, it calls `getqueryParams()` to retrieve any existing date parameters from the URL and assigns them to the corresponding input fields, ensuring that the previously selected dates persist after a page reload. The event listener `$('#fromDate, #toDate').on('change', function () { ... })` detects changes in the date fields and calls `checkDates()` to dynamically update button visibility. The `$('#filterButton').on('click', function () { ... })` function captures user input from the date fields and modifies the URL with the selected `fromDate` and `toDate` parameters, effectively reloading the page with the new filters applied. Similarly, the `$('#clearFilterButton').on('click', function () { ... })` function resets the filters by reloading the page without query parameters, restoring the default unfiltered state.



Finally, `checkDates()` is executed at the end of the script to set the correct initial button states based on existing input values. This comprehensive filtering mechanism provides user-friendly experience for date-based searches.

```

reports.php X
pages > admin > reports.php > ...
1  <?php
2  session_start();
3  if (!isset($_SESSION['user_id'])) {
4      header('Location: ../../index.php');
5      exit;
6  }
7
8  if ($_SESSION['role'] === 'User') {
9      header('Location: ../../personnel/home.php');
10     exit;
11 }
12
13 require_once '../../controllers/conn.php';
14
15 $userStats = [];
16 $folderStats = [];
17 $fileStats = [];
18
19 $fromDate = isset($_GET['fromDate']) ? $_GET['fromDate'] : null;
20 $toDate = isset($_GET['toDate']) ? $_GET['toDate'] : null;
21
22 $dateCondition = "";
23 if ($fromDate && $toDate) {
24     $fileDate = "AND DATE(upload_date) BETWEEN '$fromDate' AND '$toDate'";
25     $createdDate = "AND DATE(created_at) BETWEEN '$fromDate' AND '$toDate'";
26 } else {
27     $fileDate = "";
28     $createdDate = "";
29 }
30
31
32 $filesQuery = "SELECT department, COUNT(*) as count
33 FROM files
34 WHERE 1=1 $fileDate
35 GROUP BY department";
36 $filesResult = mysqli_query($conn, $filesQuery);

36 $filesResult = mysqli_query($conn, $filesQuery);
37
38 $totalFiles = 0;
39 while ($row = mysqli_fetch_assoc($filesResult)) {
40     $fileStats[$row['department']] = $row['count'];
41     $totalFiles += $row['count'];
42 }
43
44 $foldersQuery = "SELECT department, COUNT(*) as count
45 FROM folders
46 WHERE 1=1 $createdDate
47 GROUP BY department";
48 $foldersResult = mysqli_query($conn, $foldersQuery);
49
50 $totalFolders = 0;
51 while ($row = mysqli_fetch_assoc($foldersResult)) {
52     $folderStats[$row['department']] = $row['count'];
53     $totalFolders += $row['count'];
54 }
55
56 $usersQuery = "SELECT department, COUNT(*) as count
57 FROM user
58 WHERE 1=1 $createdDate
59 GROUP BY department";
60 $usersResult = mysqli_query($conn, $usersQuery);
61
62 $totalUsers = 0;
63 while ($row = mysqli_fetch_assoc($usersResult)) {
64     $userStats[$row['department']] = $row['count'];
65     $totalUsers += $row['count'];
66 }
67
68 mysqli_close($conn);
69 ?>
```

Figure 51. Code Snippet for admin reports – reports.php



Figure 51 shows the code for “reports.php” script is responsible for generating statistical summaries of files, folders, and user counts per department while enforcing role-based access control. It starts a session and checks if a user is logged in, redirecting unauthorized users based on their roles. It retrieves optional date filters from GET parameters to apply conditions for filtering uploaded files and created folders within a given date range. The script then executes SQL queries to count the number of files, folders, and users per department, grouping them accordingly. The retrieved data is stored in associative arrays and accumulated into total counters, ensuring a comprehensive summary of all records. The results can be used to display statistical insights, providing an overview of system usage and data distribution. Finally, the script closes the database connection to optimize resource utilization. To enhance security and performance, implementing prepared statements would prevent SQL injection, and indexing database tables would improve query efficiency.

Upload date	File name	Department	File owner	Uploaded by
Feb 27, 2025	Glass	Administrative Records	Elmerto	Winston Lloyd B. Timogon

Metadata title	Metadata description
Date of Purchase	February 27, 2025
Supplier	Jr Glass
Amount	10500

File Path
Administrative Records / Acknowledgement Receipt / Supplies / Glass

[View](#) [Approve](#) [Decline](#)

Figure 52. Admin - List of Pending files

Figure 52 shows the system displays a list of files awaiting admin approval, providing a detailed description for each pending file. Admins can review the file, including its upload date,



file name, department, file owner, uploader, and associated metadata such as title and description.

They have the option to either accept or decline the file based on its relevance and compliance.

Upload date	File name	Department	File owner	Uploaded by
Feb 27, 2025	Glass	Administrative Records	Elmerito	Winston Lloyd B. Timogan

Metadata title	Metadata description
Date of Purchase	February 27, 2025
Supplier	Jr Glass
Amount	10500

File Path
Administrative Records / Acknowledgement Receipt / Supplies / Glass

Figure 53. Admin - List of Approve Files

Figure 53 shows the system displays of an approved file, providing essential details such as the upload date, file name, department, file owner, uploader, and metadata, including the title and description.

Upload date	File name	Department	File owner	Uploaded by	Remarks
Feb 27, 2025	Food	Administrative Records	John Carlo	Winston Lloyd B. Timogan	Wrong file

Metadata title	Metadata description
Date of Purchase	February 27, 2025
Supplier	Marave
Amount	15100

File Path
Administrative Records / Acknowledgement Receipt / Supplies / Food

Figure 54. Admin - List of Decline Files



Figure 54 shows the system lists all declined files along with details such as the upload date, file name, department, file owner, uploader, and metadata, including the title and description.

```

db_functions.php X
controllers > db_functions.php > getFilePath
1  <?php
2
3  function getFilePath($conn, $folder_id)
4  {
5      $path = [];
6
7      while ($folder_id) {
8          $folder_query = "SELECT folder_name, parent_folder FROM folders WHERE folder_id = ?";
9          $stmt = mysqli_prepare($conn, $folder_query);
10         mysqli_stmt_bind_param($stmt, 'i', $folder_id);
11         mysqli_stmt_execute($stmt);
12         $result = mysqli_stmt_get_result($stmt);
13
14         if ($folder = mysqli_fetch_assoc($result)) {
15             array_unshift($path, $folder['folder_name']);
16             $folder_id = $folder['parent_folder'];
17         } else {
18             break;
19         }
20     }
21
22     return implode(' / ', $path);
23 }

25 function getMetadataTitles($conn)
26 {
27     $metadata_titles = [];
28     $title_query = "SELECT DISTINCT meta_data_title, metadata_id FROM metadata_title";
29     $title_result = mysqli_query($conn, $title_query);
30     while ($row = mysqli_fetch_assoc($title_result)) {
31         $metadata_titles[$row['metadata_id']] = $row['meta_data_title'];
32     }
33     return $metadata_titles;
34 }

47 function getFiles($conn, $status, $user_id, $filters = [], $is_admin = false)
48 {
49     $file_query = "
50         SELECT f.file_id, f.original_name, f.upload_date, f.author, f.folder_id, f.user_id,
51             f.file_name, f.department, f.remarks, u.full_name
52         FROM files f
53         JOIN user u ON f.user_id = u.id
54         WHERE f.status = ?
55     ";
56
57     if (!$is_admin) {
58         $file_query .= " AND f.user_id = ? ";
59     }
60
61     $where_clauses = [];
62     $params = [$status];
63
64     if (!$is_admin) {
65         $params[] = $user_id;
66     }
67
68     if (isset($filters['fromDate']) && !empty($filters['fromDate'])) {
69         $where_clauses[] = "f.upload_date > ?";
70         $params[] = $filters['fromDate'];
71     }
72
73     if (isset($filters['toDate']) && !empty($filters['toDate'])) {
74         $where_clauses[] = "f.upload_date < ?";
75         $params[] = $filters['toDate'];
76     }
}

```



```

77     if (!isset($filters['department']) && !empty($filters['department'])) {
78         $where_clauses[] = "f.department = ?";
79         $params[] = $filters['department'];
80     }
81
82     if (!empty($where_clauses)) {
83         $file_query .= " AND " . implode(" AND ", $where_clauses);
84     }
85
86     $stmt = mysqli_prepare($conn, $file_query);
87     if (!empty($params)) {
88         $types = str_repeat('s', count($params));
89         mysqli_stmt_bind_param($stmt, $types, ...$params);
90     }
91     mysqli_stmt_execute($stmt);
92     return mysqli_stmt_get_result($stmt);
93 }
94

```

```

96     function getFileMetadata($conn, $file_id)
97     {
98         $metadata_query = "
99             SELECT mt.meta_data_title, md.meta_data_description
100            FROM metadata_description md
101           JOIN metadata_title mt ON md.title_id = mt.metadata_id
102          WHERE md.file_id = ?
103         ";
104         $stmt = mysqli_prepare($conn, $metadata_query);
105         mysqli_stmt_bind_param($stmt, 's', $file_id);
106         mysqli_stmt_execute($stmt);
107         return mysqli_stmt_get_result($stmt);
108     }

```

Figure 55. Code Snippet for display Pending / Approve / Decline - db_function.php

Figure 55 shows the code snippet for `getFilePath` function constructs the full file path by traversing the folder hierarchy from the given `folder_id` up to the root folder. It initializes an empty array `$path` to store folder names. Inside a while loop, it fetches the `folder_name` and `parent_folder` for the given `folder_id` using a prepared statement. The fetched folder name is added to the beginning of the `$path` array using `array_unshift()`, ensuring that the path is built in the correct order. The loop continues by setting `$folder_id` to its `parent_folder`, moving up the hierarchy until there are no more parent folders. Finally, the function returns the folder path as a string, with folder names separated by slashes (/), providing a readable and structured directory path.

`getMetadataTitles` function retrieves a list of distinct metadata titles from the `metadata_title` table. It initializes an empty array `$metadata_titles` to store the results. The query selects distinct `meta_data_title` and `metadata_id`, ensuring unique metadata titles are fetched. The function then executes the query using `mysqli_query()` and iterates through the results using `mysqli_fetch_assoc()`, storing each metadata title in an



associative array where the key is `metadata_id` and the value is `meta_data_title`. This structured approach allows efficient metadata retrieval, which can be useful for filtering or categorization.

This snippet handles dynamic file retrieval, specifically filtering based on the department. It first checks if a department filter exists in the `$filters` array. If present, it adds a WHERE clause to the query and stores the department parameter in the `$params` array. If filtering conditions exist, they are appended to the query using `implode()` to join conditions with AND. The query is then prepared with `mysqli_prepare()`, and parameters are bound dynamically using `mysqli_stmt_bind_param()` based on the number of conditions. The query is executed, and the result set is returned for further processing. This function provides a flexible way to filter files based on department or other criteria.

`getFileMetadata` Function retrieves metadata associated with a specific file. It performs a JOIN between `metadata_description` and `metadata_title` to fetch the metadata title and description for the given `file_id`. The function uses a prepared statement for security, binding `file_id` as a parameter to prevent SQL injection. After executing the query, it returns the result set for further processing. This function is essential for displaying detailed metadata about files, improving searchability and categorization in the system.



The screenshot shows the 'List Departments' page of the 'Archival Document Mapping Management System'. The page has a header with the college's crest and the title 'Municipal Social Welfare and Development'. On the left is a sidebar with 'Admin Panel' navigation. The main area displays a table titled 'List Departments' with columns for 'Department Name', 'Color', and 'Action'. The table lists five departments: Administrative Records (pink), Financial Records (orange), Legal Records (red), Personnel Records (yellow), and Social Services Records (green). Each row has an 'Edit' button. At the bottom, it says 'Showing 1 to 5 of 5 entries'.

Figure 56. Admin - List Departments

Figure 56 shows the system displays a list of all departments, from administrative records to social services. Users can navigate through the departments to view stored files. There is also an option to add a new department when needed.

```

91 <?php
92 require_once '../controllers/conn.php';
93 $query = "SELECT * FROM department WHERE department_name != 'Admin'";
94 $result = $conn->query($query);
95
96 if ($result && $result->num_rows > 0) {
97     while ($row = $result->fetch_assoc()) {
98         echo '<tr>';
99         echo '<td>' . htmlspecialchars($row['department_name']) . '</td>';
100        echo '<td><span class="badge" style="background-color: #' . htmlspecialchars($row['folder_color']) . '">' .
101             htmlspecialchars($row['folder_color']) . '</span></td>';
102        echo '<td>' .
103            '<button class="btn btn-warning btn-sm edit-btn"
104                  data-bs-toggle="modal"
105                  data-bs-target="#editDepartmentModal"
106                  data-id="' . htmlspecialchars($row['id']) . '"
107                  data-name="' . htmlspecialchars($row['department_name']) . '"
108                  data-color="#" . htmlspecialchars($row['folder_color']) . '">' .
109            '<i class="bi bi-pencil"></i> Edit
110        '</button>
111        '</td>';
112    echo '</tr>';
113 }
114 ?>
```

Figure 57. Code Snippet for display department – department.php

Figure 57 shows the code snippet for PHP script retrieves and displays department data from a database, excluding the "Admin" department to ensure only relevant records are shown. It begins by requiring the database connection file to establish communication. The query SELECT * FROM department WHERE department_name != 'Admin' fetches all departments



except "Admin" to maintain data integrity. The script executes this query using \$conn->query(\$query), ensuring valid results. If the query returns rows, a while loop iterates through each department record. The htmlspecialchars() function is applied to dynamic content to prevent XSS (Cross-Site Scripting) attacks, ensuring secure output handling. This implementation provides a structured and interactive approach to managing department records while prioritizing usability and maintainability

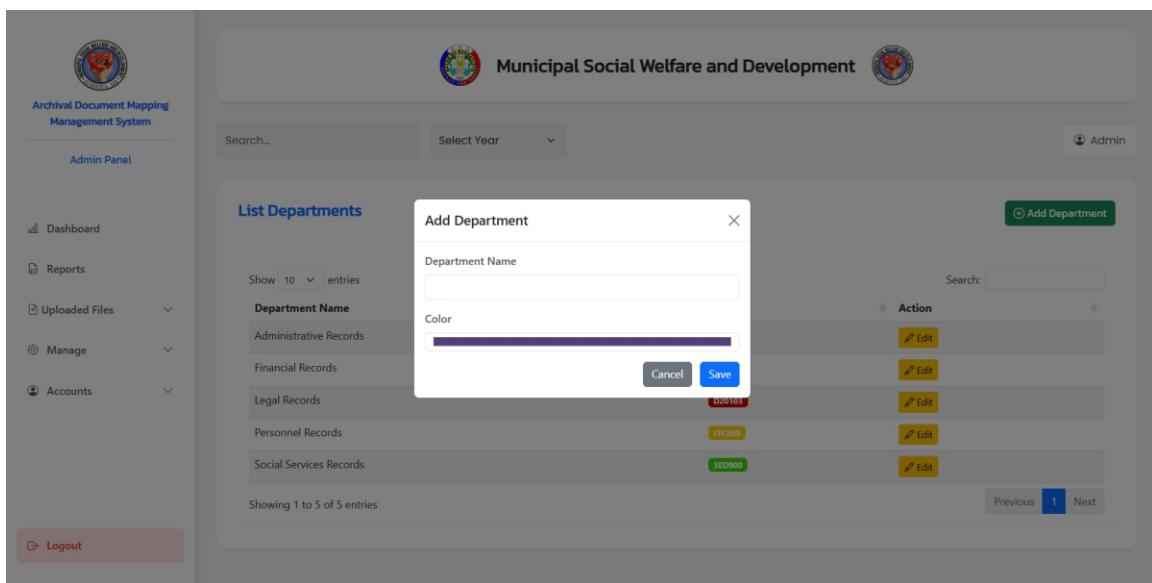


Figure 58. Admin - Add Department

Figure 58 shows the adding of a new department, users can enter the department name and select a color for it. This allows for easy identification and organization of departments within the system.

```

1 $(document).ready(function () {
2   $('#departmentsTable').DataTable();
3
4   $('.edit-btn').on('click', function () {
5     let id = $(this).data('id');
6     let name = $(this).data('name');
7     let color = $(this).data('color');
8
9     $('#editDepartmentId').val(id);
10    $('#editDepartmentName').val(name);
11    $('#editDepartmentColor').val(color);
12  });
13
14  $('#addDepartmentForm').on('submit', function (e) {
15    e.preventDefault();
16
17    const modal = bootstrap.Modal.getInstance($('#addDepartmentModal'));

```



```

18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
      $.ajax({
        url: '../../../../../controllers/create.php',
        type: 'POST',
        data: $(this).serialize() + '&action=add_department',
        dataType: 'json',
        success: function (response) {
          modal.hide();
          if (response.success) {
            Swal.fire({
              icon: 'success',
              title: 'Success',
              text: response.message
            }).then(() => {
              location.reload();
            });
          } else {
            Swal.fire({
              icon: 'error',
              title: 'Error',
              text: response.message
            });
          }
        },
        error: function (xhr, status, error) {
          modal.hide();
          console.error(xhr, status, error);
          Swal.fire({
            icon: 'error',
            title: 'Error',
            text: 'An error occurred while processing your request'
          });
        }
      });
    
```

Figure 59. Code snippet for department - add department

Figure 59 shows the code snippet for `$(document).ready(function () { ... })` function initializes various event handlers and functionalities as soon as the page loads, ensuring seamless user interactions with the department management interface. The first action it performs is initializing the DataTable plugin on the `#departmentsTable` element, enabling sorting, searching, and pagination for better data organization and navigation. Next, the `$('.edit-btn').on('click', function () { ... })` function sets up an event listener for edit buttons in the department list. When an edit button is clicked, it extracts the department's ID, name, and color from the button's data attributes and assigns these values to the corresponding input fields in the edit department modal, preparing the form for modification. The `$('#addDepartmentForm').on('submit', function (e) { ... })` function then handles the submission of the "Add Department" form, preventing the default submission process and instead sending the form data via an AJAX request to `../../../../controllers/create.php`. It serializes the form data and appends an `action=add_department` parameter before sending it to the server. Upon receiving a



response, the function hides the modal and displays a `Swal.fire` alert indicating success or failure. If the request is successful, the page reloads to display the newly added department; otherwise, an error alert is triggered. In case of an AJAX request failure, the function logs the error in the console and notifies the user with an error message, ensuring transparency and user feedback throughout the process.

```

55
56 $('#editDepartmentForm').on('submit', function (e) {
57   e.preventDefault();
58
59   const modal = bootstrap.Modal.getInstance($('#editDepartmentModal'));
60
61   const formData = {
62     action: 'update_department',
63     department_id: $('#editDepartmentId').val(),
64     department_name: $('#editDepartmentName').val(),
65     folder_color: $('#editDepartmentColor').val()
66   };
67
68   $.ajax({
69     url: '../../../../../controllers/update.php',
70     type: 'POST',
71     contentType: 'application/json',
72     data: JSON.stringify(formData),
73     dataType: 'json',
74     success: function (response) {
75       modal.hide();
76
77       if (response.success) {
78         Swal.fire({
79           icon: 'success',
80           title: 'Success',
81           text: response.message
82         }).then(() => {
83           location.reload();
84         });
85       } else {
86         Swal.fire({
87           icon: 'error',
88           title: 'Error',
89           text: response.message
90         });
91     },
92   },
93
94   error: function (xhr, status, error) {
95     modal.hide();
96
97     console.error(xhr, status, error);
98     Swal.fire({
99       icon: 'error',
100      title: 'Error',
101      text: 'An error occurred while processing your request'
102    });
103  });
104
105  $('#addDepartmentModal, #editDepartmentModal').on('hidden.bs.modal', function () {
106    $(this).find('form').trigger('reset');
107  });
108});
109});
```

Figure 60. Code snippet for department – edit department

Figure 60 shows the code snippet for the department – edit department.

`$('#editDepartmentForm').on('submit', function (e) { ... })` function handles the submission of the "Edit Department" form, preventing the default form submission



behavior and instead sending the form data via an AJAX request to update the department details.

It first retrieves the Bootstrap modal instance (`editDepartmentModal`) to allow for modal manipulation. The function constructs a `formData` object, collecting values from the form fields such as `department_id`, `department_name`, and `folder_color`, then sends this data as a JSON payload to `../controllers/update.php` using a POST request. Upon successful completion of the request, the modal is hidden, and a `Swal.fire` notification is triggered to inform the user of the update's success, followed by a page reload to reflect the changes. If an error occurs during the request, the modal is still hidden, an error message is logged in the console, and an alert notifies the user that an issue has occurred. Additionally, the function `$('#addDepartmentModal, #editDepartmentModal').on('hidden.bs.modal', function () { ... })` ensures that when either the "Add Department" or "Edit Department" modal is closed, their associated forms are reset, clearing any previously entered data. This prevents outdated values from persisting when the modal is reopened, ensuring a clean and user-friendly experience while managing department records.

```
create.php < ...
controllers > create.php > ...
153 } elseif ($action === 'add_department') {
154     // for adding of department (admin | manage | departments | add deprtment)
155     try {
156         if (!isset($_SESSION['user_id']) || $_SESSION['role'] !== 'Admin') {
157             throw new Exception('Unauthorized access');
158         }
159
160         $department_name = trim($_POST['department_name'] ?? '');
161         $folder_color = trim($_POST['folder_color'] ?? '');
162
163         $folder_color = str_replace('#', '', $folder_color);
164
165         if (empty($department_name) || empty($folder_color)) {
166             throw new Exception('All fields are required');
167         }
168
169         $check_query = "SELECT id FROM department WHERE department_name = ?";
170         $check_stmt = $conn->prepare($check_query);
171         if (!$check_stmt) {
172             throw new Exception("Database prepare error: " . $conn->error);
173         }
174
175         $check_stmt->bind_param("s", $department_name);
176         if (!$check_stmt->execute()) {
177             throw new Exception("Database execution error: " . $check_stmt->error);
178         }
179
180         $result = $check_stmt->get_result();
181         if ($result->num_rows > 0) {
182             throw new Exception('Department name already exists');
183         }
184
185         $query = "INSERT INTO department (department_name, folder_color) VALUES (?, ?)";
186         $stmt = $conn->prepare($query);
187         if (!$stmt) {
188             throw new Exception("Database prepare error: " . $conn->error);
189         }
```

A screenshot of a code editor showing a PHP script named 'create.php'. The code is part of a function for adding a new department. It includes error handling for database execution errors, JSON encoding of responses, and cleanup of prepared statements in a finally block. The code is numbered from 189 to 217.

```
189     }
190
191     $stmt->bind_param("ss", $department_name, $folder_color);
192     if (!$stmt->execute()) {
193         throw new Exception('Database execution error: ' . $stmt->error);
194     }
195
196     echo json_encode([
197         'success' => true,
198         'message' => 'Department added successfully'
199     ]);
200
201 } catch (Exception $e) {
202     echo json_encode([
203         'success' => false,
204         'message' => $e->getMessage()
205     ]);
206 } finally {
207     if (isset($stmt))
208         $stmt->close();
209     if (isset($check_stmt))
210         $check_stmt->close();
211 }
212 } else {
213     echo json_encode([
214         'success' => false,
215         'message' => 'Invalid action!'
216     ]);
217 }
```

Figure 61. Code Snippet for adding department – create.php

Figure 61 shows the code for `add_department` function in this script handles the addition of new departments while ensuring role-based access control for administrators. It begins by verifying the user's session and role, restricting access to only authorized administrators. It then retrieves and sanitizes the `department_name` and `folder_color` from the POST request, trimming whitespace and removing the `#` symbol from the color input. The script checks if both fields are provided and throws an exception if any are missing. Next, it prepares and executes a query to check if the department name already exists in the database, preventing duplicate entries. If the department name is unique, it proceeds to prepare and execute an INSERT statement to store the new department along with its folder color. Any database errors encountered during the preparation or execution of queries trigger exceptions, ensuring robust error handling. Upon successful insertion, the script returns a JSON response indicating success; otherwise, it provides an error message describing the issue. Finally, in the `finally` block, the script ensures that all prepared statements are closed properly, preventing memory leaks and maintaining efficient database operations. This implementation enhances security and



performance by utilizing prepared statements to prevent SQL injection while maintaining a structured and error-resilient execution flow.

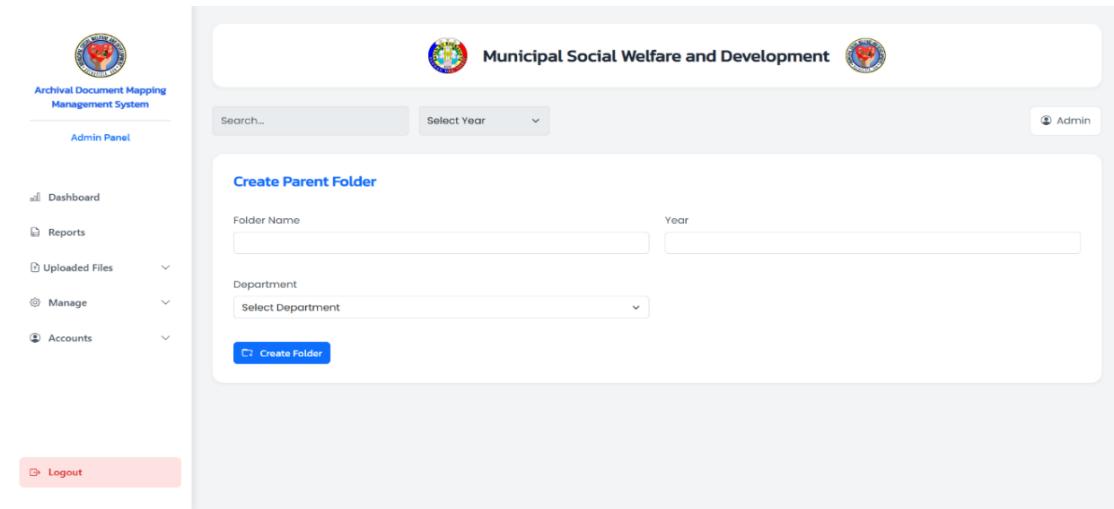


Figure 62. Admin - Create Parent folder

Figure 62 shows the creation of a parent folder, admin needs to enter the folder name, specify the year, and select a department.

```

37 } elseif ($action === 'create_folder') {
38     // for creating parent folder (admin | add folder)
39     date_default_timezone_set('Asia/Manila');
40     $currentDate = date('Y-m-d');
41
42     $folderName = $_POST['folderName'];
43     $year = $_POST['year'];
44     $department = $_POST['department'];
45
46     $stmt = $conn->prepare("INSERT INTO folders (folder_name, year, department, created_at) VALUES (?, ?, ?, ?)");
47     $stmt->bind_param("ssss", $folderName, $year, $department, $currentDate);
48
49     if ($stmt->execute()) {
50         echo json_encode([
51             'success' => true,
52             'message' => "Folder created successfully!"
53         ]);
54     } else {
55         echo json_encode([
56             'success' => false,
57             'message' => "Error: " . $stmt->error
58         ]);
59     }
60
61     $stmt->close();

```

Figure 63. Code Snippet for creating parent folder – create.php

Figure 63 shows the code for PHP script handles the creation of a new parent folder in the system while ensuring data integrity and security. It first sets the default timezone to Asia/Manila to ensure timestamps are accurate. Then, it retrieves the folder name, year, and department from the POST request. The script prepares an SQL INSERT statement using \$conn-



>prepare(), which safely inserts the folder name, year, department, and the current date (Y-m-d) into the folders table. It binds the parameters using \$stmt->bind_param("siss", \$folderName, \$year, \$department, \$currentDate), where "siss" represents the expected data types: a string for the folder name (s), an integer for the year (i), a string for the department (s), and another string for the created date (s). The script then executes the query using \$stmt->execute() and checks whether the execution was successful. If successful, it returns a JSON response with a success message; otherwise, it returns an error message with the SQL error details. Finally, \$stmt->close() ensures the prepared statement is properly closed, preventing memory leaks and maintaining optimal database performance.

A screenshot of the Admin Panel interface. The left sidebar has a logo and the text "Archival Document Mapping Management System". Below it are menu items: "Admin Panel", "Dashboard", "Reports", "Uploaded Files", "Manage", and "Accounts". At the bottom of the sidebar is a red "Logout" button. The main content area has a header with the college logo and the text "Municipal Social Welfare and Development". It includes a search bar, a "Show all Folders" dropdown, and an "Admin" user icon. A sub-section titled "Folders" displays a single folder entry: "Administrative Records" (2025) with a warning icon indicating "Missing location details".

Figure 64. Admin – Display Folders

Figure 64 showcases the Display Folders section on the admin interface, providing a structured overview of all folders in the system. It displays key details such as the folder name, assigned department, and management actions. This feature enables administrators to effectively organize, access, and oversee folders, ensuring efficient document storage and retrieval.



```

233 <!-- folders listahan / display sa folders -->
234 <?php while ($folder = $folders_result->fetch_assoc()):>
235     $folderColor = getFolderColor($folder['department']);
236     $rowNotEmpty = empty($folder['row_no']);
237     $columnNotEmpty = empty($folder['column_no']);
238   >
239   <div class="col-12 col-md-3 mb-3">
240     <div class="folder-card" ondblclick="openFolder(<?php echo $folder['folder_id']; ?>)">
241       data-folder-id=<?php echo $folder['folder_id']; ?>
242       style="border-left-color: <?php echo $folderColor; ?>">
243         <i class="bi bi-folder-fill folder-icon" style="color: <?php echo $folderColor; ?>"></i>
244         <div class="folder-info">
245           <h3 class="kanit"><?php echo htmlspecialchars($folder['folder_name']); ?></h3>
246           <p class="poppins">
247             <?php echo htmlspecialchars($folder['year']); ?>
248           </p>
249           <?php if ($rowNotEmpty || $columnNotEmpty): ?>
250             <p class="text-danger kanit">
251               <i class="bi bi-exclamation-triangle-fill"></i> Missing location details
252             </p>
253           </?php endif; ?>
254         </div>
255         <div class="folder-menu" data-bs-toggle="modal" data-bs-target="#folderModal"
256           data-folder-name=<?php echo htmlspecialchars($folder['folder_name']); ?>
257           data-year=<?php echo htmlspecialchars($folder['year']); ?>
258           data-row=<?php echo htmlspecialchars($folder['row_no']); ?>
259           data-column=<?php echo htmlspecialchars($folder['column_no']); ?>
260           data-department=<?php echo htmlspecialchars($folder['department']); ?>
261           data-created-at=<?php echo htmlspecialchars($folder['created_at']); ?>
262           data-parent-folder=<?php echo htmlspecialchars($folder['parent_folder']); ?>>
263         <!-- added parent folder for checking (to only be viewed to subfolders) -->
264         <i class="bi bi-three-dots-vertical"></i>
265       </div>
266     </div>
267   </?php endwhile; ?>
268 
```

Figure 65. Code Snippet for display folder – folders.php

Figure 65 shows the code for the responsible for retrieving folder data from the database and dynamically generating HTML to display the folders. It loops through the query results using `$folders_result->fetch_assoc()`, extracting relevant details such as `folder_id`, `folder_name`, `year`, `department`, `created_at`, `row_no`, and `column_no`. The function `getFolderColor($folder['department'])` assigns a specific color based on the department, ensuring visual differentiation. Each folder is wrapped in a div with an onclick event that calls `openFolder(<?php echo $folder['folder_id']; ?>)`, which likely triggers a JavaScript function to open the selected folder. Security measures such as `htmlspecialchars()` are applied to prevent XSS attacks when rendering user-generated content. Additional folder details are stored as data- attributes within `div.folder-menu`, making them accessible for JavaScript interactions such as opening modals, displaying metadata, or performing further backend operations. This setup enables an interactive folder navigation system while maintaining structured, secure, and scalable folder management.



```

12 if ($user_department === 'admin') {
13     // for admin side folders
14     if ($parent_folder_id) {
15         $query = "SELECT * FROM folders WHERE parent_folder = ?";
16         $stmt = $conn->prepare($query);
17         $stmt->bind_param("i", $parent_folder_id);
18         $stmt->execute();
19         $folders_result = $stmt->get_result();
20
21         $files_query = "SELECT * FROM files WHERE folder_id = ? AND status = 'approved'";
22         $files_stmt = $conn->prepare($files_query);
23         $files_stmt->bind_param("i", $parent_folder_id);
24         $files_stmt->execute();
25         $files_result = $files_stmt->get_result();
26
27         $parent_query = "SELECT folder_name FROM folders WHERE folder_id = ?";
28         $parent_stmt = $conn->prepare($parent_query);
29         $parent_stmt->bind_param("i", $parent_folder_id);
30         $parent_stmt->execute();
31         $parent_result = $parent_stmt->get_result();
32         $parent_folder = $parent_result->fetch_assoc();
33     } else {
34         $query = "SELECT * FROM folders WHERE (parent_folder IS NULL OR parent_folder = 0)";
35         $stmt = $conn->prepare($query);
36         $stmt->execute();
37         $folders_result = $stmt->get_result();
38
39         $files_result = false;
40     }
41
42     $years_query = "SELECT DISTINCT year FROM folders ORDER BY year DESC";
43     $years_stmt = $conn->prepare($years_query);
44     $years_stmt->execute();
45     $years_result = $years_stmt->get_result();
46 } else {

```

Figure 66. Code Snippet for displaying folders and files - get_folders.php

Figure 66 shows the code for the PHP script responsible for retrieving and displaying folders and files based on the user's role and selected parent folder. It begins by checking if the user belongs to the "admin" department. If so, it determines whether a specific parent folder ID is provided. If a `parent_folder_id` is specified, it fetches all subfolders under the given parent folder using a prepared SQL query, ensuring secure database interaction. Additionally, it retrieves files associated with the parent folder that have an "approved" status, ensuring that only approved files are displayed. The script also fetches the parent folder's name for reference. If no `parent_folder_id` is provided, it retrieves only top-level folders where `parent_folder` is either NULL or 0, indicating that these are root-level directories. For non-admin users, the script simply retrieves a distinct list of years from the `folders` table, ordering them in descending order to prioritize recent years. The use of prepared statements throughout ensures security by preventing SQL injection, while the structured logic ensures the appropriate display of folders and files based on user privileges and selections.

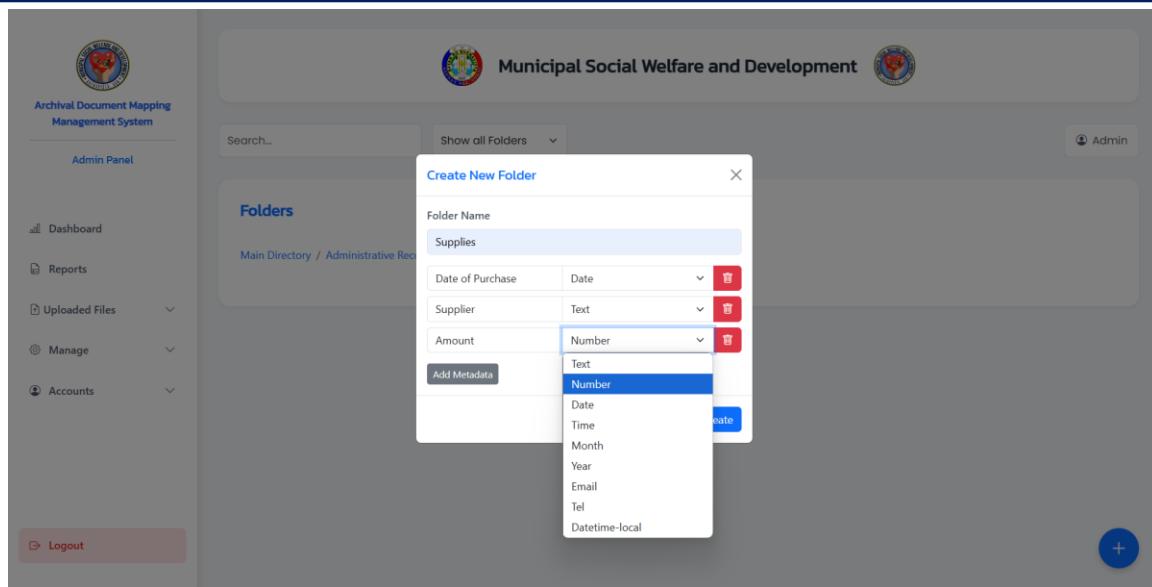


Figure 67. Admin – Create SubFolder

Figure 67 shows the creation of a new folder; users must enter the folder name and have the option to add metadata for better organization. When adding metadata, users can specify the input type, choosing from options such as text, number, date, time, month, year, email, tel, or datetime-local to ensure structured and consistent data entry. After providing the necessary details, they can click Create to finalize the folder creation, ensuring a more flexible and efficient document management system.

```

700 function addMetadataTitleField() {
701   const metadataTitlesList = document.getElementById('metadata-titles-list');
702   const metadataItem = document.createElement('div');
703   metadataItem.className = 'metadata-item d-flex justify-content-between align-items-center mb-2';
704
705   const options = ['number', 'text', 'date', 'time', 'month', 'year', 'email', 'tel', 'datetime-local'];
706
707   let selectOptions = options.map(opt => `<option value="${opt}">${opt.charAt(0).toUpperCase()} ${opt.slice(1)}</option>`).join('');
708
709   metadataItem.innerHTML = `
710     <input type="text" class="form-control form-control-sm me-2" id="new-metadata-title" placeholder="Enter metadata title">
711     <select class="form-select form-select-sm me-2" id="new-input-type">
712       ${selectOptions}
713     </select>
714   </div>
715   <button type="button" class="btn btn-sm btn-outline-success" onclick="saveNewMetadataTitle()">
716     <i class="bi bi-check"></i>
717   </button>
718   <button type="button" class="btn btn-sm btn-outline-danger" onclick="cancelNewMetadataTitle()">
719     <i class="bi bi-x"></i>
720   </button>
721 </div>
722
723   metadataTitlesList.appendChild(metadataItem);
724 }
725 
```



```

726 function saveNewMetadataTitle() {
727   const newMetadataTitle = document.getElementById('new-metadata-title').value;
728   const newInputType = document.getElementById('new-input-type').value;
729
730   if (newMetadataTitle.trim() === '') {
731     Swal.fire({
732       title: 'Error!',
733       text: 'Metadata title cannot be empty',
734       icon: 'error',
735       confirmButtonText: 'ok'
736     });
737     return;
738   }
739
740   fetch('../controllers/create.php', {
741     method: 'POST',
742     headers: {
743       'Content-Type': 'application/json',
744     },
745     body: JSON.stringify({
746       action: 'add metadata_title',
747       folder_id: currentEditFolderId,
748       meta_data_title: newMetadataTitle,
749       input_type: newInputType
750     })
751   })
752   .then(response => response.json())
753   .then(data => {
754     if (data.success) {
755       fetchMetadataTitles(currentEditFolderId);
756       Swal.fire({
757         title: 'Success!',
758         text: 'Metadata title added successfully',
759         icon: 'success',
760         timer: 1500,
761         showConfirmButton: false
762       });
763     } else {
764       Swal.fire({
765         title: 'Error!',
766         text: data.message || 'Unknown error',
767         icon: 'error',
768         confirmButtonText: 'ok'
769       });
770     }
771   })
772   .catch(error => {
773     console.error('Error:', error);
774     Swal.fire({
775       title: 'Error!',
776       text: error.message || 'An unexpected error occurred',
777       icon: 'error',
778       confirmButtonText: 'ok'
779     });
780   });
781 }
782
783

```

Figure 68. Set metadata and input type – create subfolder

Figure 68 shows the code for `addMetadataTitleField()` function dynamically adds a new metadata title input field and an input type selector to the user interface. It first retrieves the container element (`metadata-titles-list`) where metadata fields are listed. Then, it creates a new `<div>` element (`metadataItem`) with Bootstrap classes for styling and alignment. This element contains an input field for entering the metadata title, a `<select>` dropdown for choosing the input type, and two buttons for saving or canceling the metadata entry.

The function defines an array of possible input types (options), such as text, date, number, and email, ensuring flexibility in metadata collection. These options are dynamically mapped into



<option> elements and inserted into the <select> dropdown. Once fully constructed, the new metadata field is appended to metadata-titles-list, allowing users to define metadata titles with specified input types in real time.

The `saveNewMetadataTitle()` function handles the process of saving a newly added metadata title. It first retrieves the values entered in the metadata title input field and the selected input type from the dropdown. If the metadata title is empty, an alert is displayed using `SweetAlert (Swal.fire)` to notify the user that the field cannot be left blank, preventing submission errors.

If the input is valid, the function sends an HTTP POST request to `.../.../controllers/create.php` with a JSON payload containing the `folder_id`, `meta_data_title`, and `input_type`. The server processes this request and returns a JSON response. If the metadata title is successfully added, the function triggers `fetchMetadataTitles(currentEditFolderId)` to refresh the list and display the new metadata. Additionally, a success message is shown using `SweetAlert`. In case of an error, an error message is displayed, ensuring users are informed of any issues in the saving process.

```

62 } elseif ($action === 'create_subfolder') {
63     // for creating sub folder (admin folders, under fab icon (katong button nga naa sa lower right side))
64     $data = json_decode(file_get_contents('php://input'), true);
65
66     if (empty($data['folderName']) || empty($data['parentFolderId'])) {
67         echo json_encode(["success" => false, "message" => "All fields are required."]);
68         exit;
69     }
70
71     $folderName = $data['folderName'];
72     $parentFolderId = $data['parentFolderId'];
73     $metadata = $data['metadata'] ?? [];
74     $userId = $_SESSION['user_id'];
75
76     $conn->begin_transaction();
77
78     try {
79         $query = "SELECT year, department FROM folders WHERE folder_id = ?";
80         $stmt = $conn->prepare($query);
81         $stmt->bind_param('i', $parentFolderId);
82         $stmt->execute();
83         $result = $stmt->get_result();
84
85         if ($result->num_rows === 0) {
86             throw new Exception("Parent folder not found.");
87         }
88
89         $parentFolder = $result->fetch_assoc();
90         $year = $parentFolder['year'];
91         $department = $parentFolder['department'];
92         $createdAt = date('Y-m-d H:i:s');
93
94         $query = "INSERT INTO folders (folder_name, year, department, created_at, parent_folder) VALUES (?, ?, ?, ?, ?)";
95         $stmt = $conn->prepare($query);
96         $stmt->bind_param('ssssi', $folderName, $year, $department, $createdAt, $parentFolderId);

```



```

97     if (!$stmt->execute()) {
98         throw new Exception("Failed to create folder: " . $stmt->error);
99     }
100
101     $newFolderId = $conn->insert_id;
102
103     if (!empty($metadata)) {
104         $metadataQuery = "INSERT INTO metadata_title (folder_id, meta_data_title, user_id) VALUES (?, ?, ?)";
105         $metadataStmt = $conn->prepare($metadataQuery);
106
107         foreach ($metadata as $metaTitle) {
108             $metadataStmt->bind_param('isi', $newFolderId, $metaTitle, $userId);
109             if (!$metadataStmt->execute()) {
110                 throw new Exception("Failed to insert metadata: " . $metadataStmt->error);
111             }
112         }
113         $metadataStmt->close();
114     }
115
116     $conn->commit();
117     echo json_encode(['success' => true, "message" => "Folder created successfully."]);
118 }
119 } catch (Exception $e) {
120     $conn->rollback();
121     echo json_encode(['success' => false, "message" => $e->getMessage()]);
122 } finally {
123     $stmt->close();
124 }
125
126 } elseif ($action === 'add_metadata_title') {

```

Figure 69. Code Snippet for creating subfolder and set metadata title - Create.php

Figure 69 shows the code for the PHP script handles the creation of subfolders within a structured document management system. It begins by ensuring the required `folderName` and `parentFolderId` fields are present before proceeding. The script then retrieves the parent folder's year and department to maintain hierarchical consistency and prevent misclassification. Using a transaction, it inserts a new folder into the `folders` table while ensuring data integrity by committing only if all operations succeed. If metadata is provided, the script iterates through each metadata title and links it to the newly created folder via the `metadata_title` table, allowing for structured categorization. Transaction control (`begin_transaction`, `commit`, and `rollback`) is implemented to ensure atomicity—if any step fails, the `rollback` function reverts changes to prevent partial or inconsistent data. The script employs prepared statements to prevent SQL injection and maintain security. Exception handling through a `try...catch` block ensures that database errors are caught, an appropriate response is returned, and the system remains stable. Finally, resources are properly released using `$stmt->close()` to optimize performance and avoid memory leaks, ensuring efficient execution and secure folder creation.



A screenshot of a web-based administrative interface titled "Archival Document Mapping Management System". The main area is titled "Municipal Social Welfare and Development". It features a search bar, a "Show all Folders" dropdown, and a folder navigation menu. The "Folders" section shows a single item named "Glass" uploaded on "Feb 27, 2025". A three-dot action button is visible next to the file card. On the left, there's a sidebar with "Admin Panel" and a list of menu items: Dashboard, Reports, Uploaded Files (with a dropdown arrow), Manage (with a dropdown arrow), Accounts (with a dropdown arrow), and a red "Logout" button. A blue "+" button is located in the bottom right corner of the main content area.

Figure 70. Admin – Display Files Card Style

Figure 70 shows the Display Files Card Style section on the admin interface presents uploaded files in a visually structured card-based format. Each card have a three-dot action button that allows administrators to view file details or perform management actions, ensuring an organized and efficient document handling system.



 A screenshot of a code editor window showing a snippet of JavaScript code. The code is written in a modern, readable style with color-coded syntax highlighting. It uses ES6 features like arrow functions and spread operators. The code handles DOM events, specifically the 'DOMContentLoaded' event, to switch between card and table displays based on user interaction with a toggle button and a search bar.

```

3  document.addEventListener('DOMContentLoaded', function () {
4    const viewToggle = document.getElementById('viewToggle');
5    const cardView = document.querySelector('.row');
6    const tableView = document.querySelector('.table-view');
7    const folderCards = document.querySelectorAll('.folder-card');
8    const tableSearchContainer = document.getElementById('tableSearchContainer');
9    const tableSearchInput = document.getElementById('tableSearch');
10   const printButton = document.getElementById('printButton');
11
12   if (folderCards.length === 0 && document.querySelectorAll('.file-card').length > 0) {
13     viewToggle.style.display = 'flex';
14   }
15
16   viewToggle.querySelectorAll('button').forEach(button => {
17     button.addEventListener('click', function () {
18       viewToggle.querySelectorAll('button').forEach(btn => {
19         btn.classList.remove('active');
20       });
21
22       this.classList.add('active');
23
24       if (this.dataset.view === 'card') {
25         cardView.style.display = 'flex';
26         tableView.style.display = 'none';
27         tableSearchContainer.style.display = 'none';
28       } else {
29         cardView.style.display = 'none';
30         tableView.style.display = 'block';
31         tableSearchContainer.style.display = 'block';
32       }
33     });
34   });

```

Figure 71. Code snippet for card and table – display files

Figure 71 shows the code snippet for the card and table – display files. The code ensures that all DOM elements are fully loaded before executing any script, preventing errors caused by accessing elements that are not yet available. Within this event listener, multiple key elements are referenced, including `viewToggle` (the toggle button for switching views), `cardView` (the grid-style card display), `tableView` (the structured table display), and `tableSearchContainer` (the search bar for table filtering). The script first checks if there are folder cards present; if not, but file cards exist, it ensures that the toggle button is visible, allowing users to switch between views. The event listener then iterates over all buttons within `viewToggle`, adding a click event that dynamically switches between card and table views by adjusting their display styles accordingly. It also highlights the active button by removing the active class from all buttons and adding it to the clicked one, providing visual feedback to the user. Furthermore, when switching to the table view, the search container is revealed, enhancing usability by allowing users to search for specific records. This implementation improves user interaction by making the document browsing experience more flexible and intuitive, allowing



seamless toggling between different display modes while ensuring that interface elements respond dynamically to available content.

```

271 <!-- files ni nga list -->
272 <?php if ($files_result && $files_result->num_rows > 0): ?>
273     <?php while ($file = $files_result->fetch_assoc()): ?>
274         $extension = pathinfo($file['original_name'], PATHINFO_EXTENSION);
275         $icon_class = getFileIcon($extension);
276     ?>
277     <div class="col-12 col-md-3 mb-3">
278         <div class="file-card" onclick="openFile(<?php echo htmlspecialchars($file['file_name']); ?>)"/>
279             <i class="bi-file-earmark-pdf file-icon"></i>
280             <div class="file-info">
281                 <h6 class="kanit"><?php echo htmlspecialchars($file['original_name']); ?></h6>
282                 <p class="poppins">
283                     <?php
284                         $uploadDate = new DateTime($file['upload_date']);
285                         $uploadDate->setTimezone(new DateTimeZone('Asia/Dhaka'));
286                     ?>
287                 </p>
288             </div>
289             <div class="file-menu" onclick="showFileOptions(event, <?php echo $file['file_id']; ?>)"/>
290                 data-bs-toggle="modal" data-bs-target="#fileModal"
291                 data-file-name=<?php echo htmlspecialchars($file['file_name']); ?>" 
292                 data-author=<?php echo htmlspecialchars($file['author']); ?>" 
293                 data-original-name=<?php echo htmlspecialchars($file['original_name']); ?>" 
294                 data-upload-date=<?php echo htmlspecialchars($file['upload_date']); ?>" 
295                 data-user-id=<?php echo htmlspecialchars($file['user_id']); ?>" 
296                 data-file-id=<?php echo htmlspecialchars($file['file_id']); ?>" 
297                 data-folder-id=<?php echo htmlspecialchars($file['folder_id']); ?>" 
298                 data-is-owner=<?php echo ($SESSION['user_id'] == $file['user_id']) ? 'true' : 'false'; ?>" 
299                 <i class="bi bi-three-dots-vertical"></i>
300             </div>
301         </div>
302     </?php endwhile; ?>
303 <?php endif; ?>

```

Figure 72. Code Snippet for display file card style – folders.php

Figure 72 shows the code dynamically generates a card-style layout for displaying files retrieved from the database. It loops through `$files_result->fetch_assoc()` to fetch each file's details, including `file_name`, `original_name`, `author`, `upload_date`, and `user_id`. The file extension is extracted using `pathinfo()` and then used to determine an appropriate icon via `getFileIcon($extension)`. Each file is enclosed within a `div.file-card`, with an `onclick` event that likely triggers a function to open or preview the file. The file's metadata is formatted using `htmlspecialchars()` for security, ensuring protection against XSS attacks. The upload date is converted into a `DateTime` object and adjusted to the Asia/Dhaka timezone before display. Additionally, a `div.file-menu` contains `data-` attributes that store file-related information, such as `file_id`, `folder_id`, and ownership status, which can be used by JavaScript to display options like renaming, deleting, or moving files. This



structure enables a visually organized and interactive file management system within a web-based document repository.

File name	Upload date	File owner	Uploaded by	Date of Purchase	Supplier	Amount	File
Glass	Feb 27, 2025	Elmerito	Winston Lloyd B. Timogan	February 27, 2025	Jr Glass	10500	View

Figure 73. Admin – Display Files Table Style

Figure 73 shows the Display Files Table Style section on the admin interface presents uploaded files in a structured tabular format. Each row includes essential details such as file name, upload date, department, file owner, uploaded by, metadata title, and metadata description ensuring efficient and organized document handling.

```

326 <?php
327 if (isset($_GET['folder_id'])) {
328     $folder_id = $_GET['folder_id'];
329     $meta_titles_query = "SELECT DISTINCT meta_data_title, metadata_id
330     FROM metadata_title
331     WHERE folder_id = ?
332     ORDER BY metadata_id";
333     $stmt = $conn->prepare($meta_titles_query);
334     $stmt->bind_param("i", $folder_id);
335     $stmt->execute();
336     $meta_titles_result = $stmt->get_result();
337
338     $metadata_titles = [];
339     while ($title = $meta_titles_result->fetch_assoc()) {
340         $metadata_titles[] = [
341             'id' => $title['meta_data_id'],
342             'title' => $title['meta_data_title']
343         ];
344     }
345     echo "<th>" . htmlspecialchars($title['meta_data_title']) . "</th>";
346 }
347 ?>

```

```
352
353     <?php
354     if ($files_result && $files_result->num_rows > 0):
355         $files_result->data_seek(0);
356         while ($file = $files_result->fetch_assoc()):
357             $uploader_query = "SELECT full_name FROM user WHERE id = ?";
358             $stmt = $conn->prepare($uploader_query);
359             $stmt->bind_param("i", $file['user_id']);
360             $stmt->execute();
361             $uploader_result = $stmt->get_result();
362             $uploader = $uploader_result->fetch_assoc();
363
364             $metadata_query = "SELECT mt.metadata_id, mt.meta_data_title, md.meta_data_description
365             FROM metadata_title mt
366             LEFT JOIN metadata_description md
367             ON mt.metadata_id = md.title_id
368             AND md.file_id = ?
369             WHERE mt.folder_id = ?
370             ORDER BY mt.metadata_id";
371             $stmt = $conn->prepare($metadata_query);
372             $stmt->bind_param("ii", $file['file_id'], $file['folder_id']);
373             $stmt->execute();
374             $metadata_result = $stmt->get_result();
375
376             $file_metadata = [];
377             while ($metadata = $metadata_result->fetch_assoc()) {
378                 $file_metadata[$metadata['metadata_id']] = [
379                     'title' => $metadata['meta_data_title'],
380                     'description' => $metadata['meta_data_description']
381                 ];
382             }
383         >?
384
385
386
387
388     <?php
389     foreach ($metadata_titles as $title_info) {
390         $metadata = $file_metadata[$title_info['id']] ?? ['title' => $title_info['title'], 'description' => ''];
391         echo "<td> " . htmlspecialchars($metadata['description']) . "</td>";
392     }
393     ?>
```

Figure 74. Code Snippet for display file table style – folders.php

Figure 74 shows the code for the PHP code is designed to dynamically generate an HTML table that organizes and displays metadata associated with files stored in a system. It retrieves distinct metadata titles from the `metadata_title` table, ensuring that each title appears only once, and then it fetches corresponding metadata descriptions from the `metadata_description` table using a LEFT JOIN, which allows files with missing descriptions to still be included. The retrieved metadata is stored in arrays, making it easier to iterate over the results and populate the table structure efficiently. The table headers are generated dynamically based on the metadata titles, ensuring flexibility even if new metadata fields are added in the database. Each row represents a file, and metadata descriptions are displayed in the corresponding columns, with missing values handled gracefully using the ?? operator, which assigns default values when data is unavailable. To enhance the table's readability and usability, consider applying CSS for styling, implementing alternate row shading, and using "N/A" or placeholders for missing data. Additionally, improving database performance by indexing key



columns (metadata_id, folder_id, file_id) will optimize query execution. Implementing client-side features such as sorting, filtering, and searching can further improve user experience, making it easier to navigate large datasets.

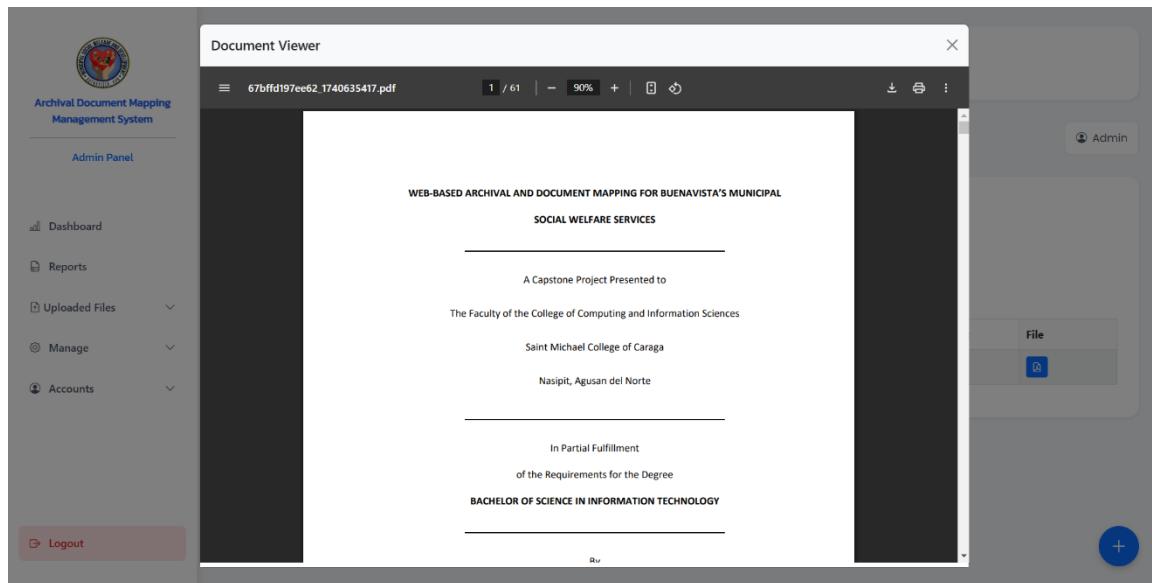


Figure 75. Admin - Document PDF Viewer

Figure 75 shows the Document PDF Viewer allows administrators to open, view, and navigate PDF files within the system, providing essential controls like zoom and scroll for efficient document review.



```

open-file.js x
assets > js > open-file.js > addEventListener(hidden.bs.modal) callback
1  function openFile(filename, shouldShowModal = false) {
2    if (shouldShowModal) {
3      const url = new URL(window.location.href);
4      url.searchParams.set('file', filename);
5      window.history.pushState({}, '', url);
6
7      // file id
8      const fileCard = document.querySelector('.file-card [data-file-name="${filename}"]');
9      const fileId = fileCard.getAttribute('data-file-id');
10
11     document.getElementById('pdfViewer').src = `.../uploads/${filename}`;
12     fetchAndDisplayMetadata(fileId);
13
14     const pdfModal = new bootstrap.Modal(document.getElementById('pdfViewerModal'));
15     pdfModal.show();
16   }
17 }
18
19 document.getElementById('pdfViewerModal').addEventListener('hidden.bs.modal', function () {
20   const url = new URL(window.location.href);
21   url.searchParams.delete('file');
22   window.history.pushState({}, '', url);
23
24   document.getElementById('pdfViewer').src = '';
25 });
26
27 document.querySelectorAll('.file-card').forEach(card => {
28   let tapedTwice = false;
29   let lastTap = 0;
30   let clickTimeout = null;
31
32   card.addEventListener('touchend', function (e) {
33     const currentTime = new Date().getTime();
34     const tapLength = currentTime - lastTap;
35
36     if (tapLength < 300 && tapLength > 0) {
37       e.preventDefault();
38       const filename = this.querySelector('.file-menu').getAttribute('data-file-name');
39       openFile(filename, true);
40     }
41
42     lastTap = currentTime;
43   });
44
45   card.addEventListener('dblclick', function (e) {
46     if (clickTimeout) {
47       clearTimeout(clickTimeout);
48       clickTimeout = null;
49     }
50
51     const filename = this.querySelector('.file-menu').getAttribute('data-file-name');
52     openFile(filename, true);
53   });
54 });
55
56 document.addEventListener('DOMContentLoaded', function () {
57   const urlParams = new URLSearchParams(window.location.search);
58   const fileParam = urlParams.get('file');
59   if (fileParam) {
60     openFile(fileParam, true);
61   }
62 });

```

Figure 76. Code Snippet for PDF VIEWER – open-file.js

Figure 76 shows the code snippet for the provided JavaScript code implements a PDF viewer function that enables users to open PDF files in a modal when they interact with file elements. The `openFile` function takes a filename as an argument and dynamically updates the URL parameters while modifying the history state for seamless navigation. It identifies the corresponding file card element using `querySelector`, extracts metadata, and assigns the file path to an embedded PDF viewer. The Bootstrap modal (`pdfViewerModal`) is then displayed



to showcase the document. An event listener ensures that when the modal is closed, the PDF source is cleared to optimize memory usage. The script also includes touch and click event listeners, allowing users to open files via single or double taps, preventing unintended interactions by distinguishing between quick and delayed taps. Additionally, when the page loads, it checks for URL parameters to auto-open a PDF if a file is specified, enhancing user experience by enabling direct access through shared or bookmarked links.

A screenshot of the Admin Panel showing the 'Create Account' form. The left sidebar shows 'Archival Document Mapping Management System' and 'Admin Panel' with options like Dashboard, Reports, Uploaded Files, Manage, Accounts, and Logout. The main area has a header 'Municipal Social Welfare and Development'. The 'Create Account' form contains fields for Full Name, Username, Password, Confirm Password, Address, Contact Info, and Department (with a dropdown menu for Select Department). A blue 'Create Account' button is at the bottom.

Figure 77. Admin - Create Account

Figure 77 shows the creation of an account, users must provide their full name, username, password, address, and contact information. All fields must be completed for the account to be successfully created.



```

216 // create account function (sugod diri padung ubos)
217 $(document).ready(function () {
218   const $password = $('#password');
219   const $confirmPassword = $('#confirmPassword');
220   const $passwordMatchMessage = $('#passwordMatchMessage');
221   const $submitButton = $('#submitButton');
222   let passwordsMatch = false;
223
224   function validatePasswords() {
225     const password = $password.val();
226     const confirmPassword = $confirmPassword.val();
227
228     if (confirmPassword === '') {
229       $confirmPassword.removeClass('password-mismatch');
230       $passwordMatchMessage.text('');
231       passwordsMatch = false;
232     } else if (password !== confirmPassword) {
233       $confirmPassword.addClass('password-mismatch');
234       $passwordMatchMessage.text('Passwords do not match');
235       passwordsMatch = false;
236     } else {
237       $confirmPassword.removeClass('password-mismatch');
238       $passwordMatchMessage.text('');
239       passwordsMatch = true;
240     }
241   }
242
243   $password.on('input', validatePasswords);
244   $confirmPassword.on('input', validatePasswords);
245
246   $('#createAccountForm').on('submit', function (e) {
247     e.preventDefault();
248
249     validatePasswords();
250   });
251
252   if (!passwordsMatch) {
253     Swal.fire({
254       title: 'Error!',
255       text: 'Passwords do not match',
256       icon: 'error',
257       confirmButtonColor: '#dc2626'
258     });
259     return;
260   }
261
262   $.ajax({
263     url: '../../../../../controllers/create.php',
264     type: 'POST',
265     data: $(this).serialize() + '&action=create_account',
266     datatype: 'json',
267     success: function (response) {
268       if (response.success) {
269         Swal.fire({
270           title: 'Success!',
271           text: 'Account has been created successfully!',
272           icon: 'success',
273           confirmButtonColor: '#2563eb'
274         }).then(result => {
275           if (result.isConfirmed || result.isDismissed) {
276             window.location.href = 'accounts.php';
277           }
278         });
279       } else {
280         Swal.fire({
281           title: 'Error!',
282           text: response.message || 'Something went wrong!',
283           icon: 'error',
284           confirmButtonColor: '#dc2626'
285         });
286       }
287     },
288     error: function (xhr, status, error) {
289       let errorMessage = xhr.responseJSON && xhr.responseJSON.message
290       ? xhr.responseJSON.message
291       : 'Something went wrong with the server!';
292
293       Swal.fire({
294         title: 'Error!',
295         text: errorMessage,
296         icon: 'error',
297         confirmButtonColor: '#dc2626'
298       });
299     });
300   });
301 });

```

Figure 78. Code snippet for account – create account



Figure 78 shows the code snippet for the validatePasswords function ensures that the password and confirm password fields match before allowing form submission. It continuously monitors user input and provides immediate feedback by checking whether the confirm password field is empty, mismatched, or correctly matching the original password. If the passwords do not match, a visual warning (password-mismatch class) is applied, and an error message appears. If they match, the warning is removed, and the form is allowed to proceed. This function enhances user experience by preventing incorrect password submissions and reducing frustration during account creation. The `$('#createAccountForm').on('submit', function (e) { . . . })` event handler intercepts the form submission to validate the passwords before sending data to the server. If the passwords do not match, a SweetAlert error message is displayed, and submission is halted. If validation passes, an AJAX request sends the form data to `../../controllers/create.php`, handling success and failure responses accordingly. A successful response triggers a success message and redirects the user to `accounts.php`, while errors prompt an appropriate warning. This implementation improves efficiency by preventing unnecessary page reloads, enhances security through client-side validation, and ensures a seamless user experience by providing real-time feedback and error handling.

```

12 // for creating account (admin | add account)
13 if ($action === 'create_account') {
14     $full_name = $conn->real_escape_string($_POST['full_name']);
15     $username = $conn->real_escape_string($_POST['username']);
16     $department = $conn->real_escape_string($_POST['department']);
17     $address = $conn->real_escape_string($_POST['address']);
18     $contact_info = $conn->real_escape_string($_POST['contact_info']);
19     $role = 'User';
20     $status = 'enabled';
21     $password = password_hash($conn->real_escape_string($_POST['password']), PASSWORD_BCRYPT);
22
23     $sql = "INSERT INTO user (full_name, username, department, address, contact_info, password, role, status, created_at)
24           VALUES ('$full_name', '$username', '$department', '$address', '$contact_info', '$password', '$role', '$status', NOW())";
25
26     if ($conn->query($sql) === TRUE) {
27         echo json_encode([
28             'success' => true,
29             'message' => 'Account created successfully!'
30         ]);
31     } else {
32         echo json_encode([
33             'success' => false,
34             'message' => "Error: " . $conn->error
35         ]);
36     }
37 } elseif ($action === 'create_folder') {

```



Figure 79. Code Snippet for create personnel account – create.php



Figure 79 shows the code snippet for the provided PHP script handles the creation of user accounts through an HTTP POST request, ensuring secure data handling and database insertion. It first checks if the \$action variable is set to 'create_account', indicating a request to register a new user. The script retrieves and sanitizes input data, including the full name, username, department, address, and contact information, using `real_escape_string()` to prevent SQL injection. A default role of 'User' and status of 'enabled' are assigned. The password is securely hashed using `password_hash()` with the PASSWORD_BCRYPT algorithm before storage. The script then constructs an SQL INSERT statement to store the user details in the database, including a timestamp generated by `NOW()`. If the database query executes successfully, it returns a JSON-encoded success message; otherwise, it returns an error message with the MySQL error for debugging. This implementation follows essential security practices like password hashing and escaping inputs but lacks prepared statements, which could further enhance security by preventing SQL injection.

NAME	USERNAME	ADDRESS	CONTACT No.	DEPARTMENT	CHANGE PASSWORD	ACTIONS
James Nolie Piel	james	Buenavista Agusan del Norte	09553556487	FINANCIAL RECORDS	<button>Change Password</button>	<button>Edit</button> <button>Disable</button>
Juli-Ann Echalco	juli-ann	Guinabsan Buenavista Agusan Del Norte	09553556487	ADMINISTRATIVE RECORDS	<button>Change Password</button>	<button>Edit</button> <button>Disable</button>
Winston Lloyd B. Timogan	timogan001	Cult Nasipit Agusan del Norte	09553556487	ADMINISTRATIVE RECORDS	<button>Change Password</button>	<button>Edit</button> <button>Disable</button>

Figure 80. Admin – Display Accounts



Figure 80 shows the Accounts section displays all created accounts along with their details and assigned departments.

```

10 // function to display lists of accounts (admin | accounts list)
11 function loadAccounts() {
12     const tableName = 'user';
13     $.getJSON('../controllers/get.php?table=' + tableName, function (data) {
14         if (data.error) {
15             Swal.fire('Error', data.error, 'error');
16             return;
17         }
18
19         let rows = '';
20         if (data.length > 0) {
21             data.forEach(item => {
22                 let badgeHTML = '';
23                 if (item.folder_color) {
24                     badgeHTML = `<span class="badge text-dark" style="background-color: ${item.folder_color};">${item.department.toUpperCase()}</span>`;
25                 } else {
26                     badgeHTML = `<span class="badge bg-primary">SYSTEM ADMIN</span>`;
27                 }
28
29                 const statusBtn = item.status === 'enabled' ?
30                     '<button class="btn btn-sm btn-danger" onclick="toggleStatus(${item.id}, \'disabled\')>Disable</button>' :
31                     '<button class="btn btn-sm btn-success" onclick="toggleStatus(${item.id}, \'enabled\')>Enable</button>';
32
33                 rows += `
34                     <tr>
35                         <td>${item.full_name}</td>
36                         <td>${item.username}</td>
37                         <td>${item.address}</td>
38                         <td>${item.contact_info}</td>
39                         <td class="text-center" style="text-align: center; font-size: small;">${badgeHTML}</td>
40                         <td class="text-center kanit">
41                             <button class="btn btn-sm btn-warning" onclick="changePassword(${item.id})">
42                                 Change Password
43                             </button>
44                         </td>
45                         <td class="text-center kanit">
46                             <button class="btn btn-sm btn-warning" onclick="editUser(${item.id})">
47                                 Edit
48                             </button>
49                             ${statusBtn}
50                         </td>
51                     </tr>
52                 `;
53             });
54         } else {
55             rows = '<tr><td colspan="6" class="text-center">No data</td></tr>';
56         }
57
58         $('#accounts-table').html(rows);
59
60         try {
61             if ($.fn.DataTable.isDataTable('#table')) {
62                 $('#table').DataTable().destroy();
63             }
64             $('#table').DataTable();
65         } catch (error) {
66             console.error('Error initializing DataTable:', error);
67         }
68     }).fail(function (jqXHR, textStatus, errorThrown) {
69         Swal.fire('Error', 'Failed to load accounts', 'error');
70     });
71 }

```

Figure 81. Code snippet for account – display accounts

Figure 81 shows the code snippet for the `loadAccounts` function is responsible for dynamically retrieving and displaying user account data in a table format by making an AJAX request to fetch JSON data from the server. It starts by defining `tableName` as 'user', which is used to request user data from `get.php` via jQuery's `$.getJSON` method. Upon receiving the data, it first checks for an error response and displays a SweetAlert error notification if an issue occurs. If data is successfully retrieved, it iterates through the user list and constructs table rows



dynamically, incorporating various elements such as department badges (styled with the user's folder color if available or defaulting to a "SYSTEM ADMIN" badge), a status toggle button that allows enabling or disabling accounts, and action buttons for editing user details and changing passwords. The HTML generated is then injected into the #accounts-table table body. The function also ensures proper table formatting by attempting to destroy any existing DataTable instance before reinitializing it using jQuery DataTables, preventing duplicate or conflicting table setups. Finally, if an AJAX request fails due to network issues or server errors, the function gracefully handles it by displaying an appropriate SweetAlert error message, ensuring a seamless and user-friendly experience.

```

124 // handle requests for user table ----
125 $tableName = $_GET['table'];
126
127 // Fetching a single user
128 if (isset($_GET['id'])) {
129     $id = $conn->real_escape_string($_GET['id']);
130     $query = "SELECT id, full_name, username, address, department, contact_info FROM $tableName WHERE id = '$id'";
131     $result = $conn->query($query);
132
133     if ($result && $result->num_rows > 0) {
134         echo json_encode($result->fetch_assoc());
135     } else {
136         echo json_encode(['error' => 'User not found']);
137     }
138 }
139 // get user full name ----
140 elseif (isset($_GET['user_id'])) {
141     $user_id = intval($_GET['user_id']);
142     $query = "SELECT full_name FROM $tableName WHERE id = ?";
143     $stmt = $conn->prepare($query);
144     $stmt->bind_param("i", $user_id);
145     $stmt->execute();
146     $result = $stmt->get_result();
147
148     if ($user = $result->fetch_assoc()) {
149         echo json_encode(['success' => true, 'full_name' => $user['full_name']]);
150     } else {
151         echo json_encode(['success' => false, 'error' => 'User not found']);
152     }
153 }

154 // get all users except the admin ----
155 else {
156     $sql = "SELECT u.*, d.folder_color
157           FROM $tableName u
158           LEFT JOIN department d ON u.department = d.department_name
159           WHERE u.role != 'Admin'";
160     $result = $conn->query($sql);

161     $data = [];
162     if ($result && $result->num_rows > 0) {
163         while ($row = $result->fetch_assoc()) {
164             $data[] = $row;
165         }
166     }
167
168     echo json_encode($data);
169 }
170
171 $conn->close();
172 ?>

```

Figure 82. Code Snippet for retrieve personnel account – get.php



Figure 82 shows the code snippet for the provided PHP script handles different operations for retrieving user data from a database. The first function fetches a single user's details using their ID, where it takes the id from the GET request, sanitizes it, and queries the database for fields like full name, username, address, department, and contact info. If a result is found, it returns the user data in JSON format; otherwise, it responds with an error message. The second function retrieves only the full name of a user based on a `User_id` parameter from the GET request. It prepares a SQL statement, binds the user ID parameter securely, executes the query, and returns the full name if the user exists, otherwise, it returns an error. The third function retrieves all users except those with the role of "Admin" by joining the user table with the department table based on the department field. It fetches all matching users, stores them in an array, encodes them as JSON, and outputs the data, ensuring that only non-admin users are retrieved. Finally, the script closes the database connection to free resources after execution.

A screenshot of a web-based personnel management system. The left sidebar has a header "Archival Document Mapping Management System" and a "Personnel Panel" link. Below are links for "Home", "Profile", "Reports", "Add Parent Folder", and "Uploaded Files". At the bottom is a red "Logout" button. The main content area has a header "Municipal Social Welfare and Development" with the college's logo on either side. It includes a search bar, a dropdown menu, and a user profile for "Winston Lloyd B. Timogan". A "Folders" section displays a single folder named "Administrative Records" with a sub-note "2024" and "Location Row 1, Column 5".

Figure 83. Personnel – Display Folders

Figure 83 shows the Display Folders section on the personnel interface presenting a structured view of available folders. Each folder has a three-dot action button that allows



personnel to view folder details or perform related actions, ensuring efficient document organization and access.

Figure 84. Personnel – Display Files Card Style

Figure 84 shows the Display Files Card Style section on the personnel interface presents uploaded files in a visually organized card-based format. Each card has a three-dot action button that allows personnel to view file details or perform other actions, while a double tap opens the PDF file for quick access.

#	File name	Upload date	File owner	Uploaded by	Date of Purchase	Supplier	Amount	File
1	Glass	Apr 07, 2025	Elmerto	Winston Lloyd B. Timogan	March 25, 2025	Jr Glass	10,500	

Figure 85. Personnel – Display Files Table Style



Figure 85 shows the Display Files Table Style section on the personnel interface presents uploaded files in a structured tabular format. Each row includes details such as file name, upload date, file owner, uploaded by, metadata title, and metadata description ensuring efficient file management.

A screenshot of a web-based document viewer. The top bar shows the title "Document Viewer" and the file path "67f3a7e4c83eb_1744021476.pdf". Below this is a toolbar with zoom controls (90%), navigation icons, and a download button. The main content area displays a PDF titled "WEB-BASED ARCHIVAL AND DOCUMENT MAPPING FOR BUENAVISTA'S MUNICIPAL SOCIAL WELFARE SERVICES". The PDF contains text about a capstone project presented to the Faculty of the College of Computing and Information Sciences at Saint Michael College of Caraga, Nasipit, Agusan del Norte. It also mentions the requirements for the Bachelor of Science in Information Technology degree. On the left side of the viewer, there is a sidebar for the "Archival Document Mapping Management System" under "Personnel Panel", showing links for Home, Profile, Reports, Add Parent Folder, and Uploaded Files. A "Logout" button is at the bottom of this sidebar. On the right side, there is a user profile for "Winston Lloyd B. Timogon" and a small "File" icon. The bottom right corner of the viewer has a blue circular button with a plus sign.

Figure 86. Personnel – Document PDF Viewer

Figure 86 shows the Document PDF Viewer allows personnel to open, view, and navigate PDF files within the system. It includes essential controls like zoom, scroll, and page navigation, ensuring easy access and efficient document review.

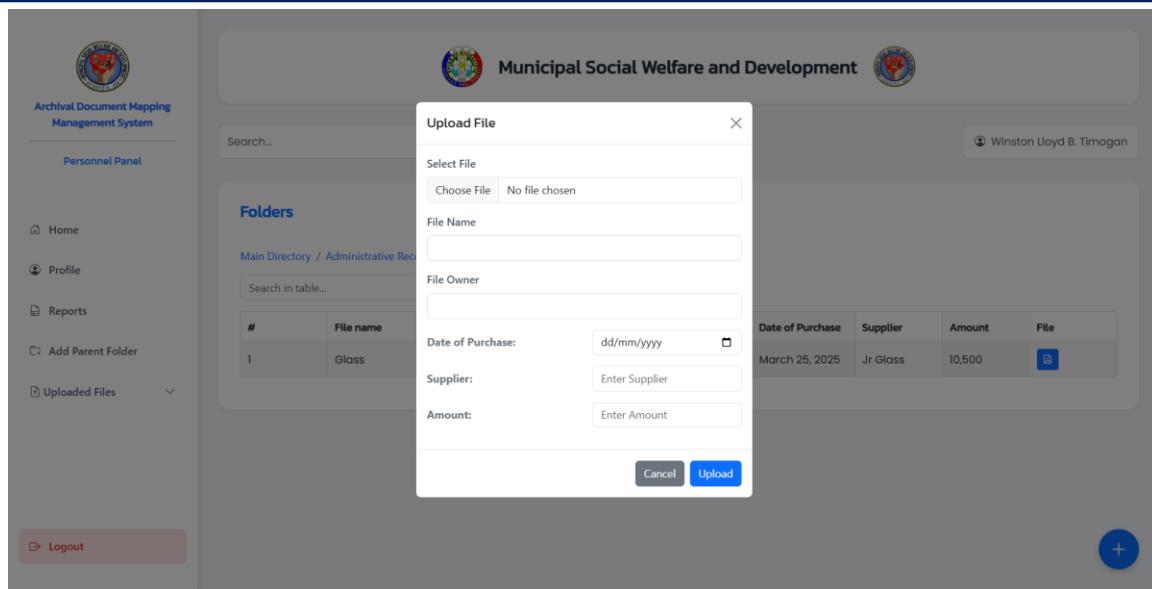


Figure 87. Personnel - Upload a file

Figure 87 shows the Upload a File section allows personnel to select and upload files to the system. Users can input details such as select file, file name, file owner, and metadata description before submission. This feature ensures organized and efficient document management.

```

630 function showUploadModal() {
631   const modal = new bootstrap.Modal(document.getElementById('uploadModal'));
632   const folderId = new URLSearchParams(window.location.search).get('folder_id');
633
634   fetch(`.../controllers/get.php?table=metadata_titles&folder_id=${folderId}`)
635     .then(response => response.json())
636     .then(data => {
637       const metadataSection = document.getElementById('metadataSection');
638       metadataSection.innerHTML = '';
639
640       if (data && data.length > 0) {
641         data.forEach(metadata => {
642           const metadataRow = document.createElement('div');
643           metadataRow.className = 'row mb-3 align-items-center';
644
645           const labelCol = document.createElement('div');
646           labelCol.className = 'col-6 fw-bold text-secondary text-start';
647           labelCol.textContent = `${metadata.meta_data_title}:`;
648
649           const inputCol = document.createElement('div');
650           inputCol.className = 'col-6';
651
652           const input = document.createElement('input');
653           input.type = metadata.input_type || 'text'; // Use stored input type
654           input.className = 'form-control';
655           input.name = `description_${metadata.metadata_id}`;
656           input.placeholder = `Enter ${metadata.meta_data_title}`;
657
658           inputCol.appendChild(input);
659           metadataRow.appendChild(labelCol);
660           metadataRow.appendChild(inputCol);
661           metadataSection.appendChild(metadataRow);
662         });
663       }
664     });
665   }
666 
```

A screenshot of a code editor showing a snippet of JavaScript code. The code is part of a function named `showUploadModal`. It starts by logging the created metadata fields to the console. It then checks if there are any metadata titles. If there are, it iterates through them and creates input fields with labels. If there are no titles, it displays a message saying "No metadata titles found for this folder.". After creating the fields, it logs the generated metadata fields again. Finally, it shows the modal using `modal.show()` and hides any floating action button (FAB) options by removing the `show` class from the FAB selector.

Figure 88. Code snippet for modal – upload file

Figure 88 shows the code for the `showUploadModal()` function initializes and displays the file upload modal while dynamically generating metadata input fields based on the selected folder's predefined metadata titles. It starts by retrieving the `folder_id` from the URL parameters and sending a `GET` request to `.../.../controllers/get.php` to fetch metadata titles along with their input types. After clearing any existing content in the `metadataSection`, it iterates through the retrieved metadata list, creating labeled input fields that match the stored input types (e.g., text, date, number, email). These fields are structured using Bootstrap classes for a clean layout, ensuring proper alignment between labels and inputs. If no metadata titles are found, a message indicating this is displayed. The function logs the generated metadata fields for debugging purposes, shows the modal using Bootstrap's `modal.show()`, and hides any floating action button (FAB) options by removing the `show` class from `fab-options`, ensuring a smooth and dynamic user experience.



```

682 // upload file func.
683 function uploadfile() {
684   const fileNameInput = document.getElementById('fileNameInput');
685   const fileInput = document.getElementById('fileInput');
686   const authorInput = document.getElementById('author');
687   const metadataInputs = document.querySelectorAll('#metadataSection input');
688   const folderId = new URLSearchParams(window.location.search).get('folder_id');
689
690   const missingFields = new Set();
691
692   if (!fileNameInput.value.trim()) {
693     missingFields.add('File Name');
694   }
695
696   if (!fileInput.files[0]) {
697     missingFields.add('File');
698   }
699
700   if (!authorInput.value.trim()) {
701     missingFields.add('Author');
702   }
703
704   let hasMetadataFields = metadataInputs.length > 0;
705   let hasEmptyMetadata = false;
706
707   if (hasMetadataFields) {
708     metadataInputs.forEach(input => {
709       if (!input.value.trim()) {
710         hasEmptyMetadata = true;
711       }
712     });
713
714   if (hasEmptyMetadata) {
715     missingFields.add('Metadata description');
716   }
717 }
718

```

```

718 if (missingFields.size > 0) {
719   Swal.fire({
720     icon: 'warning',
721     title: 'Required Fields Missing',
722     text: 'Please fill in all required fields: ${Array.from(missingFields).join(', ')}`);
723   });
724   return;
725 }
726
727 const metadataData = {};
728 metadataInputs.forEach(input => {
729   const metadataId = parseInt(input.name.split('_')[1], 10);
730   if (!isNaN(metadataId)) {
731     metadataData[metadataId] = input.value.trim();
732   }
733 });
734
735 console.log('Metadata being sent:', metadataData);
736
737 const formData = new FormData();
738 formData.append('file', fileInput.files[0]);
739 formData.append('folder_id', folderId);
740 formData.append('author', authorInput.value.trim());
741 formData.append('file_name', fileNameInput.value.trim());
742
743 if (hasMetadataFields) {
744   formData.append('metadata_json', JSON.stringify(metadataData));
745 }
746

```

```

747
748 const uploadButton = document.querySelector('#uploadModal .btn-primary');
749 uploadButton.disabled = true;
750 uploadButton.innerHTML = '<span class="spinner-border spinner-border-sm" role="status" aria-hidden="true"></span> Uploading...';
751
752 fetch('../../controllers/upload_file.php', {
753   method: 'POST',
754   body: formData
755 })
756   .then(response => {
757     if (!response.ok) {
758       return response.text().then(text => {
759         console.error('Server response:', text);
760         throw new Error(`Server returned ${response.status}: ${text}`);
761       });
762     }
763     return response.json();
764   })
765   .then(data => {
766     if (data.success) {
767       bootstrap.Modal.getInstance(document.getElementById('uploadModal')).hide();
768       swal.fire({
769         icon: 'success',
770         title: 'Success!',
771         text: 'File uploaded successfully',
772         timer: 2000
773       }).then(() => {
774         location.reload();
775       });
776     } else {
777       throw new Error(data.message || 'Upload failed');
778     }
779   })

```

A screenshot of a code editor showing a snippet of JavaScript code. The code is part of a function named `uploadFile`. It includes error handling with a catch block for errors, displaying an alert with an error message, and finally updating the disabled state and innerHTML of the upload button after the process.

Figure 89. Code snippet for uploading – upload file

Figure 89 shows the code snippet for the `uploadFile()` function. This function is responsible for handling file uploads while ensuring that all required fields, including metadata, are properly validated before submission. It starts by retrieving key input elements such as the file name, file selection, author name, and metadata input fields. It also extracts the `folder_id` from the URL parameters to associate the uploaded file with the correct directory. The function then performs validation checks by iterating through the required fields and adding any empty fields to the `missingFields` set. If any required field is missing, a SweetAlert notification is triggered to alert the user and halt the upload process, ensuring data integrity before proceeding.

Once validation is complete, the function processes metadata inputs, storing them in a structured object (`metadataData`) where each metadata entry is mapped by its `metadata_id`. The collected metadata, along with the file and other details, are packaged into a `FormData` object, allowing for efficient transmission of file data and additional attributes to the server. The function then disables the upload button and updates its text to indicate that the upload process is in progress, enhancing user experience by preventing multiple submissions. A `fetch` request is made to `../../controllers/upload_file.php` with the `POST` method, sending the `FormData` payload for processing.

On receiving a response, the function first checks if the request was successful by handling different response scenarios. If the server returns an error status, the error message is logged for debugging, and an alert is shown to notify the user of the failure. If the upload is successful, the



modal is closed using Bootstrap's `Modal.getInstance().hide()` method, and a success message is displayed using SweetAlert, followed by a page reload to reflect the changes. If an error occurs at any stage, it is caught in the `catch` block and presented to the user in a meaningful way. Finally, in the `finally` block, the upload button is re-enabled, and its text is reset to its original state, ensuring that the user interface remains responsive. This function ensures a seamless and error-free file upload process, integrating real-time validation, metadata handling, and error management.

A screenshot of a code editor showing a PHP file named "upload_file.php". The code is a try-catch block that handles file upload validation. It checks if session variables "user_id" and "department" are set, and if a file was uploaded. It also checks if the "author" field is present in the POST data. The code uses trim() to clean input and \$_SESSION to store user information. It handles errors by throwing exceptions with descriptive messages. The code editor has syntax highlighting and a sidebar with file navigation.



```

39
40     $metadata_json = $_POST['metadata_json'] ?? null;
41     $metadata = ($metadata_json) ? json_decode($metadata_json, true) : [];
42
43     if (json_last_error() !== JSON_ERROR_NONE) {
44         throw new Exception('Invalid metadata format');
45     }
46
47     $metadata_ids = array_keys($metadata);
48     $valid_ids = [];
49
50     if (!empty($metadata_ids)) {
51         $placeholders = str_repeat('?', count($metadata_ids) - 1) . '?';
52
53         $validation_query = "SELECT metadata_id FROM metadata_title
54             WHERE metadata_id IN ($placeholders)
55             AND folder_id = ?";
56
57         $validation_stmt = $conn->prepare($validation_query);
58
59         $types = str_repeat('i', count($metadata_ids)) . 'i';
60         $args = array_merge([$types], $metadata_ids, [$folder_id]);
61
62         call_user_func_array([$validation_stmt, 'bind_param'], $args);
63         $validation_stmt->execute();
64         $result = $validation_stmt->get_result();
65
66         while ($row = $result->fetch_assoc()) {
67             $valid_ids[] = $row['metadata_id'];
68         }
69         $validation_stmt->close();
70
71         $invalid_ids = array_diff($metadata_ids, $valid_ids);
72         if (!empty($invalid_ids)) {
73             throw new Exception('Invalid metadata IDs: ' . implode(', ', $invalid_ids));
74         }
75     }

```

```

76
77     $conn->begin_transaction();
78
79     try {
80         date_default_timezone_set('Asia/Manila');
81         $upload_dir = '../uploads/';
82         if (!file_exists($upload_dir)) {
83             mkdir($upload_dir, 0777, true);
84         }
85
86         $file_extension = pathinfo($file['name'], PATHINFO_EXTENSION);
87         $unique_filename = uniqid() . '_' . time() . '.' . $file_extension;
88         $upload_path = $upload_dir . $unique_filename;
89
90         if (!move_uploaded_file($file['tmp_name'], $upload_path)) {
91             throw new Exception('Failed to move uploaded file');
92         }
93
94         $query = "INSERT INTO files (file_name, original_name, upload_date, user_id, folder_id, author, department)
95             VALUES (?, ?, ?, ?, ?, ?, ?)";
96         $stmt = $conn->prepare($query);
97         $current_date = date('Y-m-d H:i:s');
98
99         $stmt->bind_param(
100             "ssisss",
101             $unique_filename,
102             $file_name,
103             $current_date,
104             $user_id,
105             $folder_id,
106             $author,
107             $department
108         );
109
110         if (!$stmt->execute()) {
111             throw new Exception('Failed to insert file record: ' . $stmt->error);
112         }

```



```

113     $file_id = $stmt->insert_id;
114     $stmt->close();
115
116     if (!empty($valid_ids)) {
117         foreach ($valid_ids as $meta_id) {
118             if (isset($metadata[$meta_id]) && !empty(trim($metadata[$meta_id]))) {
119                 $meta_query = "INSERT INTO metadata_description (file_id, title_id, meta_data_description, user_id)
120                                         VALUES (?, ?, ?, ?)";
121                 $meta_stmt = $conn->prepare($meta_query);
122                 $meta_description = trim($metadata[$meta_id]);
123                 $meta_stmt->bind_param("iisi", $file_id, $meta_id, $meta_description, $user_id);
124
125                 if (!$meta_stmt->execute()) {
126                     throw new Exception('Failed to insert metadata: ' . $meta_stmt->error);
127                 }
128                 $meta_stmt->close();
129             }
130         }
131     }
132
133     $conn->commit();
134
135     echo json_encode([
136         'success' => true,
137         'message' => 'File uploaded successfully',
138         'file_name' => $unique_filename
139     ]);
140
141 } catch (Exception $e) {
142     $conn->rollback();
143     throw $e;
144 }
145
146 } catch (Exception $e) {
147     error_log('Upload error: ' . $e->getMessage());
148     echo json_encode([
149         'success' => false,
150         'message' => $e->getMessage()
151     ]);
152 }
153
154 $conn->close();
155 ?>

```

Figure 90. Code Snippet for uploading file – upload_file.php

Figure 90 shows the code snippet for the PHP script handles file uploads with metadata validation and database storage. It starts with initializing a session and including the database connection file. Error reporting settings are configured to log errors while preventing direct display. The script checks if the user is authenticated by verifying `$_SESSION['user_id']` and `$_SESSION['department']`, ensuring only logged-in users can upload files. It then verifies if a file has been uploaded and that required fields, such as author and `file_name`, are provided. If any of these conditions fail, an exception is thrown, halting execution and returning error message in JSON format. The script also validates metadata by checking if it is a properly formatted JSON string and then extracts metadata IDs to verify against the database, ensuring that only metadata associated with the specified `folder_id` is accepted.



Next, the script starts a transaction to ensure atomicity, meaning any failure during execution will result in a rollback to prevent partial data insertion. It sets the `timezone` to '`Asia/Manila`' and prepares the upload directory, creating it if it does not exist. The uploaded file's name is modified to a unique format using `uniqid()` combined with a timestamp to prevent name conflicts. The file is then moved from its temporary location to the designated upload directory. If the move fails, an exception is thrown, preventing further execution. If successful, the script prepares an SQL statement to insert a record into the `files` table, storing details such as the unique filename, original name, upload date, user ID, folder ID, author, and department. The statement is executed, and upon success, the new file's ID is retrieved for use in metadata storage.

The script then processes metadata, inserting descriptions for each valid metadata ID into the `metadata_description` table. It loops through `valid_ids`, ensuring that each corresponding metadata description is not empty before executing an `insert` query with `bind_param` to securely add metadata records linked to the uploaded file. If an error occurs at any point, the transaction is rolled back to maintain database integrity, and an exception is thrown. Otherwise, the transaction is committed, and a JSON response is returned, confirming a successful upload along with the unique filename. Finally, the script catches any thrown exceptions, logs errors to a file (`upload_errors.log`), and returns a structured JSON response indicating failure. The database connection is closed at the end to free up resources and maintain optimal performance.



Figure 91. Personnel – Profile

Figure 91 shows the Profile section displays personnel information, including full name, username, address, contact info, and department. It also includes an edit button that allows users to update their personal details as needed.

```

14     if ($result->num_rows > 0) {
15         $user = $result->fetch_assoc();
16
17         if ($user['status'] === 'disabled') {
18             echo json_encode([
19                 'success' => false,
20                 'message' => 'Your account has been disabled. Please contact the administrator.'
21             ]);
22         } elseif (password_verify($password, $user['password'])) {
23             $_SESSION['user_id'] = $user['id'];
24             $_SESSION['username'] = $user['username'];
25             $_SESSION['department'] = $user['department'];
26             $_SESSION['address'] = $user['address'];
27             $_SESSION['contact_info'] = $user['contact_info'];
28             $_SESSION['role'] = $user['role'];
29             $_SESSION['full_name'] = $user['full_name'];
30
31             $redirect = $user['role'] === 'Admin'
32                 ? 'pages/admin/dashboard.php'
33                 : 'pages/personnel/home.php';
34
35             echo json_encode([
36                 'success' => true,
37                 'message' => 'Login successful!',
38                 'redirect' => $redirect
39             ]);
40         } else {
41             echo json_encode([
42                 'success' => false,
43                 'message' => 'Invalid password!'
44             ]);
45         }

```

Figure 92. Code Snippet for retrieve information – login.php

Figure 92 shows the code for the provided PHP script securely retrieves a user's information, including `full_name`, `username`, `address`, `contact_info`, and `department`, by checking if the user is logged in via a session variable. It establishes a database connection, prepares a parameterized SQL query to prevent SQL injection, and executes it using



the logged-in user's ID. If a matching record is found in the user table, the script fetches and returns the data in JSON format, making it easily accessible for frontend applications. In case the user is not found, or if they are not logged in, an appropriate error message is returned. This approach ensures security, efficiency, and scalability, allowing the frontend to request and display user profile details dynamically while maintaining data integrity and preventing unauthorized access.

A screenshot of a web-based personnel management system. The left sidebar shows navigation links: Home, Profile, Reports, Add Parent Folder, and Uploaded Files. The main content area has a header for "Municipal Social Welfare and Development". Below the header is a search bar and a dropdown for "Select Year". A user profile card for "Winston Lloyd B. Timogan" is shown. The "Profile" section contains fields for Full Name (Winston Lloyd B. Timogan), Username (timogan001), New Password, Confirm Password, Address (Culit Nasipit Agusan del Norte), Contact Info (09553556487), and Department (Administrative Records). At the bottom are "Cancel" and "Save Changes" buttons.

Figure 93. Personnel – update profile

Figure 93 shows the Update Profile section allows personnel to edit their full name, address, and contact info. Updating the password is optional, providing flexibility while ensuring secure account management. An update button is included to save changes.



```

136     $(document).ready(function () {
137         const originalValues = {
138             fullName: $('#fullName').val(),
139             address: $('#address').val(),
140             contactInfo: $('#contactInfo').val()
141         };
142
143         $('#editButton').click(function () {
144             $('profile input:not(#username, #department)').prop('disabled', false);
145             $('.password-field').show();
146             $(this).addClass('d-none');
147             $('#saveButton').removeClass('d-none');
148             $('#cancelButton').removeClass('d-none');
149         });
150
151         $('#cancelButton').click(function () {
152             $('#fullName').val(originalValues.fullName);
153             $('#address').val(originalValues.address);
154             $('#contactInfo').val(originalValues.contactInfo);
155
156             $('#password').val('');
157             $('#confirmPassword').val('');
158             $('#passwordMatchMessage').text('');
159
160             $('profile input:not(#username, #department)').prop('disabled', true);
161             $('.password-field').hide();
162
163             $('#editButton').removeClass('d-none');
164             $('#saveButton').addClass('d-none');
165             $(this).addClass('d-none');
166         });
167
168         const $password = $('#password');
169         const $confirmPassword = $('#confirmPassword');
170         const $passwordMatchMessage = $('#passwordMatchMessage');
171

```

```

171
172     $password.add($confirmPassword).on('input', function () {
173         validatePasswords();
174     });
175
176     function validatePasswords() {
177         const password = $password.val();
178         const confirmPassword = $confirmPassword.val();
179
180         if (confirmPassword === '') {
181             $confirmPassword.removeClass('password-mismatch');
182             $passwordMatchMessage.text('');
183         } else if (password !== confirmPassword) {
184             $confirmPassword.addClass('password-mismatch');
185             $passwordMatchMessage.text('Passwords do not match');
186         } else {
187             $confirmPassword.removeClass('password-mismatch');
188             $passwordMatchMessage.text('');
189         }
190     }
191
192     $('#profile').submit(function (e) {
193         e.preventDefault();
194
195         const formData = {
196             action: 'update_profile',
197             full_name: $('#fullName').val(),
198             address: $('#address').val(),
199             contact_info: $('#contactInfo').val(),
200             password: $('#password').val()
201         };

```

A screenshot of a code editor showing a block of JavaScript code. The code is part of a larger file, with line numbers 202 to 228 visible on the left. The code handles an AJAX POST request to 'controllers/update.php' with JSON-formatted data. It uses the Swal library for modal dialogs. If successful, it reloads the page. If there's an error, it shows an error message. The code also includes a password validation mechanism using the validatePasswords function.

Figure 94. Code snippet for profile – edit profile

Figure 94 shows the code snippet for the provided jQuery script manages user profile editing by enabling input fields, validating password inputs, and handling form submission through AJAX. When the document is ready, it first stores the original values of the user's full name, address, and contact information to allow restoration in case of cancellation. The #editButton click event enables the form inputs, making them editable while revealing the password fields and replacing the edit button with save and cancel buttons to indicate an active edit mode. If the user decides to cancel the changes, the #cancelButton click event restores the original values, clears the password fields, hides the password section, disables the input fields, and reverts the interface to its default state, effectively discarding any unsaved modifications. The script also includes a real-time password validation mechanism by listening for input events on the password and confirm password fields, triggering the validatePasswords function, which ensures that the confirmation field matches the entered password, provides immediate user feedback, and highlights mismatches visually to prevent errors. The most crucial part of the script is the AJAX-powered form submission within the #profile submit event, which prevents the default form submission behavior and instead packages the updated user details, including the password (if modified), into a JSON object that is sent to a PHP script (update.php). Upon a successful



response, it parses the JSON reply and uses SweetAlert to notify the user about the update's success before refreshing the page to reflect the changes. If an error occurs, it displays an appropriate error message, ensuring a seamless and interactive user experience while maintaining robust validation and data integrity in the update process.

```

213     } elseif ($data['action'] === 'update_profile') {
214         $userId = $_SESSION['user_id'];
215         $fullName = $data['full_name'];
216         $address = $data['address'];
217         $contactInfo = $data['contact_info'];
218         $password = $data['password'] ?? '';
219
220         try {
221             if (!empty($password)) {
222                 $passwordHash = password_hash($password, PASSWORD_BCRYPT);
223                 $sql = "UPDATE user SET full_name = ?, contact_info = ?, address = ?, password = ? WHERE id = ?";
224                 $stmt = $conn->prepare($sql);
225                 $stmt->bind_param("ssssi", $fullName, $contactInfo, $address, $passwordHash, $userId);
226             } else {
227                 $sql = "UPDATE user SET full_name = ?, contact_info = ?, address = ? WHERE id = ?";
228                 $stmt = $conn->prepare($sql);
229                 $stmt->bind_param("sssi", $fullName, $contactInfo, $address, $userId);
230             }
231
232             if ($stmt->execute()) {
233                 if ($stmt->affected_rows > 0) {
234                     $_SESSION['full_name'] = $fullName;
235                     $_SESSION['address'] = $address;
236                     $_SESSION['contact_info'] = $contactInfo;
237
238                     echo json_encode(['success' => true, 'message' => 'Profile updated successfully']);
239                 } else {
240                     echo json_encode(['success' => true, 'message' => 'No changes detected']);
241                 }
242             } else {
243                 echo json_encode(['success' => false, 'error' => 'Database error: ' . $stmt->error]);
244             }
245
246             $stmt->close();
247         } catch (Exception $e) {
248             echo json_encode(['success' => false, 'error' => $e->getMessage()]);
249         }
250     }

```

Figure 95. Code Snippet for updating profile – update.php

Figure 95 shows the code snippet for the function in the provided PHP script handles the profile update process for a user, allowing changes to their `full_name`, `contact_info`, `address`, and optionally their `password`. It starts by retrieving the user's session ID to identify which account should be updated. The script then securely captures input values, ensuring that if a password is provided, it is hashed using `PASSWORD_BCRYPT` before being stored in the database. A prepared SQL statement is used to prevent SQL injection, where two variations of the `UPDATE` query are executed—one including the password update if provided, and the other excluding it if left empty. Once executed, the script checks if any rows were affected; if so, it updates the session variables to reflect the changes and returns a success message. If no rows were affected, it indicates that no modifications were made. In case of errors, appropriate error messages are



returned, ensuring that database failures are properly handled. The use of try-catch enhances the robustness of the function by catching exceptions and preventing system crashes, ensuring smooth user experience and security in profile management.

The screenshot shows the 'Archival Document Mapping Management System' interface. On the left, a sidebar menu includes 'Home', 'Profile', 'Reports' (which is selected and highlighted in blue), 'Add Parent Folder', and 'Uploaded Files'. At the bottom of the sidebar is a 'Logout' button. The main content area is titled 'Municipal Social Welfare and Development'. It features a search bar with 'From:' and 'To:' fields and a user dropdown for 'Winston Lloyd B. Timogan'. Below this is a 'Reports' section with two cards: one for 'TOTAL FOLDERS' (6) and one for 'TOTAL FILES' (2), both categorized under 'Administrative Records'.

Figure 96. Personnel – Reports

Figure 96 shows the Reports section displays the total number of folders and uploaded files in the system by department.

```

15  $userStats = [];
16  $folderStats = [];
17  $fileStats = [];
18
19  $fromDate = isset($_GET['fromDate']) ? $_GET['fromDate'] : null;
20  $toDate = isset($_GET['toDate']) ? $_GET['toDate'] : null;
21
22  $dateCondition = "";
23  if ($fromDate && $toDate) {
24      $fileDate = "AND DATE(upload_date) BETWEEN '$fromDate' AND '$toDate'";
25      $createdDate = "AND DATE(created_at) BETWEEN '$fromDate' AND '$toDate'";
26  } else {
27      $fileDate = "";
28      $createdDate = "";
29  }
30
31  $userDepartment = $_SESSION['department'];
32
33  $filesQuery = "SELECT department, COUNT(*) as count
34          FROM files
35         WHERE department = '$userDepartment' $fileDate
36         GROUP BY department";
37  $filesResult = mysqli_query($conn, $filesQuery);
38
39  $totalFiles = 0;
40  while ($row = mysqli_fetch_assoc($filesResult)) {
41      $fileStats[$row['department']] = $row['count'];
42      $totalFiles += $row['count'];
43  }
44
45  $foldersQuery = "SELECT department, COUNT(*) as count
46          FROM folders
47         WHERE department = '$userDepartment' $createdDate
48         GROUP BY department";
49  $foldersResult = mysqli_query($conn, $foldersQuery);
50

```





```
50 $totalFolders = 0;
51 while ($row = mysqli_fetch_assoc($foldersResult)) {
52     $folderStats[$row['department']] = $row['count'];
53     $totalFolders += $row['count'];
54 }
55
56 mysqli_close($conn);
57 }>
```

Figure 97. Code snippet for reports total count of folders and uploaded files – reports.php

Figure 97 shows the code snippet for the script designed to generate statistical data on files and folders within a specific department, considering optional date filters. It begins by initializing three arrays: \$userStats, \$folderStats, and \$fileStats, which will store statistical data. It then retrieves fromDate and toDate parameters from the GET request to filter results based on date ranges; if both are provided, conditions are dynamically added to restrict queries to the specified period. The current user's department is extracted from the session, ensuring that the statistics reflect only the relevant department's data. The script first executes a query on the files table to count the number of files associated with the user's department while applying the date condition if available. The results are stored in \$fileStats, and the total number of files is accumulated in \$totalFiles. Similarly, a second query retrieves the count of folders from the folders table, filtered by department and optional date range, storing results in \$folderStats while updating \$totalFolders. The use of mysqli_query and mysqli_fetch_assoc ensures an iterative approach to collecting data efficiently, and finally, mysqli_close(\$conn) closes the database connection, ensuring proper resource management. The script provides a structured way to generate department-based file and folder statistics, making it useful for tracking content growth over time.



The screenshot shows the 'Archival Document Mapping Management System' interface. On the left, a sidebar titled 'Personnel Panel' includes links for Home, Profile, Reports, Add Parent Folder, and Uploaded Files. A 'Logout' button is at the bottom of the sidebar. The main content area is titled 'Municipal Social Welfare and Development'. It features a search bar, a dropdown for 'Select Year', and a user profile for 'Winston Lloyd B. Timogan'. Below this is a form titled 'Create Parent Folder' with fields for 'Folder Name' and 'Year', and a 'Create Folder' button.

Figure 98. Personnel – Add Parent Folder

Figure 98 shows the Add Parent Folder section allows personnel to create a parent folder within their assigned department only. The form includes input fields for folder name and year, ensuring proper organization and classification of documents.

The screenshot shows the 'Archival Document Mapping Management System' interface. The sidebar and 'Logout' button are identical to Figure 98. The main content area is titled 'Municipal Social Welfare and Development'. It includes a date range filter ('From: dd/mm/yyyy To: dd/mm/yyyy'), 'Filter', 'Clear', and 'Print' buttons, and a user profile for 'Winston Lloyd B. Timogan'. Below this is a table titled 'List of Pending Files' with columns for Upload date, File name, and File owner. The table shows one entry: Bond Paper uploaded by Juli-ann on April 7, 2025. At the bottom, there is a 'File Path' section and buttons for View, Edit, and Delete.

Figure 99. Personnel - List of Pending Files

Figure 99 shows the List of Pending Files section displays files that are awaiting review and approval. Each entry includes details such as uploaded date, filename, file owner, metadata title, and metadata description.



The screenshot shows the "List of Approved Files" section of the personnel management system. It displays a single file entry:

Upload date	File name	File owner
Apr 7, 2025	Glass	Elmerito

Below the main table, there is a "Metadata" section with the following details:

Metadata title	Metadata description
Date of Purchase	March 25, 2025
Supplier	Jr Glass
Amount	10.500

At the bottom, there is a "File Path" section with the path: "Administrative Records / Acknowledgement Receipt / Supplies / Glass". A "View" button is located at the bottom right.

Figure 100. Personnel – List of Approved Files

Figure 100 shows the List of Approved Files section displays all files that have been reviewed and approved. Each entry includes details such as uploaded date, filename, file owner, metadata title, metadata description, and file path.

The screenshot shows the "List of Declined Files" section of the personnel management system. It displays a single file entry:

Upload date	File name	File owner	Remarks
Apr 7, 2025	Food	Juli-ann	Wrong file

Below the main table, there is a "Metadata" section with the following details:

Metadata title	Metadata description
Date of Purchase	April 7, 2025
Supplier	Marave
Amount	15,000

At the bottom, there is a "File Path" section with the path: "Administrative Records / Acknowledgement Receipt / Supplies / Food". Buttons for "View" and "Reupload" are located at the bottom right.

Figure 101. Personnel – List of Declined Files



Figure 101 shows the List of Declined Files section displays all files that have been reviewed and rejected. Each entry includes details such as uploaded date, filename, file owner, remarks, metadata title, and metadata description.

```

172     if (isAdmin) {
173       if (!isDepartmentFilterApplied) {
174         departmentColumn = `<th style="width: 20%;" class="text-center">Department</th>`;
175       }
176       uploadedByColumn = `<th style="width: 20%;" class="text-center">Uploaded By</th>`;
177     }
178
179     if (showRemarks) {
180       remarksColumn = `<th style="width: 20%;" class="text-center">Remarks</th>`;
181     }
182
183     return `
184       <table class="table table-bordered" style="width: 100%; margin-bottom: 0;">
185         <tr>
186           <th style="width: 15%;" class="text-center">Upload Date</th>
187           <th style="width: 30%;" class="text-center">File Name</th>
188           ${departmentColumn}
189           <th style="width: 20%;" class="text-center">File Owner</th>
190           ${uploadedByColumn}
191           ${remarksColumn}
192         </tr>
193         <tr>
194           <td>${uploadDate}</td>
195           <td>${fileName}</td>
196           ${isAdmin && !isDepartmentFilterApplied ? `<td>${department}</td>` : ''}
197           <td>${fileOwner}</td>
198           ${isAdmin ? `<td>${uploadedBy}</td>` : ''}
199           ${showRemarks ? `<td>${remarks}</td>` : ''}
200         </tr>
201       </table>
202     `;
203   }
204 }

207   generateMetadataPrintTable(metadataTable) {
208     let content = `
209       <table class="table table-bordered" style="width: 100%; margin-bottom: 0;">
210         <tr>
211           <th style="width: 50%;" class="text-center">Metadata title</th>
212           <th style="width: 50%;" class="text-center">Metadata description</th>
213         </tr>
214     `;
215     const metadataRows = metadataTable.querySelectorAll('tr:not(:first-child)');
216     let hasValidData = false;
217
218     metadataRows.forEach(row => {
219       const cells = row.querySelectorAll('td');
220       if (cells.length === 2) {
221         const title = cells[0].textContent.trim();
222         const description = cells[1].textContent.trim();
223         if (title || description) {
224           hasValidData = true;
225           content += `
226             <tr>
227               <td>${title} || 'No data found'</td>
228               <td>${description} || 'No data found'</td>
229             </tr>;
230         }
231       }
232     });
233
234     if (!hasValidData) {
235       content += `
236         <tr>
237           <td colspan="2" class="text-center">No data found</td>
238         </tr>;
239     }
240
241     content += `</table>`;
242     return content;
243   }
244
245   generateFilePathPrintTable(filePathTable) {
246     const filePath = filePathTable.querySelector('tr:last-child td')?.textContent || '';
247     return `
248       <table class="table table-bordered mb-4" style="width: 100%; margin-bottom: 2rem;">
249         <tr>
250           <th class="text-center">File Path</th>
251         </tr>
252         <tr>
253           <td>${filePath}</td>
254         </tr>
255       </table>;
256   
```

A screenshot of a code editor showing a block of JavaScript code. The code defines a function `openPrintWindow` that takes `titleText` and `printContent` as parameters. It uses `window.userRole`, `window.userFullName`, and `window.userDepartment` to determine the signatory section. If the user is an admin, it creates a row with two columns: name and position. Otherwise, it creates a row with one column containing both name and position. The code also includes a section for generating metadata and file paths.

Figure 102. Code snippet for generate pdf and print – report

Figure 102 shows the code snippet for the provided script containing multiple JavaScript functions that dynamically generate HTML tables and print-ready content related to file records, metadata, and file paths, along with an open print window function that includes a signatory section. The first function constructs a table displaying uploaded files with conditional columns such as "Department," "Uploaded By," and "Remarks," ensuring that administrators see department-specific details while general users have limited visibility. It dynamically inserts file information such as the upload date, file name, file owner, and optional remarks based on predefined conditions. The `generateMetadataPrintTable` function extracts metadata from an existing table, ensuring only valid metadata entries with a title and description are included while gracefully handling missing data by inserting a "No data found" message. This function processes table rows dynamically and builds a structured output, allowing for seamless metadata printing. Similarly, the `generateFilePathPrintTable` function retrieves and displays the file path from the last row of a given table, ensuring that file paths are clearly presented for printing purposes. Finally, the `openPrintWindow` function prepares a formatted print preview that includes document details, and a signatory section based on the user's role. If



the user is an admin, only the designated official's signature appears; otherwise, both the user's name and department, along with the official's credentials, are displayed. The script enhances usability by dynamically structuring content based on user roles, ensuring well-formatted print-ready outputs while optimizing data presentation for different levels of access.

4.2 Testing Process

The testing process outlines the various procedures conducted to assess the system's accuracy, functionality, and overall performance. To evaluate its effectiveness, efficiency, and reliability, the researchers utilized a survey instrument based on the System Usability, Functionality, and Efficiency Scale test. This instrument was adapted from the ISO 25010 Software Product Quality Standards. Respondents rated the system using a four-point scale, where 4 represented the highest rating and 1 the lowest.

Data were analyzed and interpreted based on the following parameters:

Table 3
Functional Suitability

Adjectival Rating	Scale Range (Mean)	Verbal Interpretation
4	3.50 – 4.0	Very Functional
3	2.50 – 3.49	Functional
2	1.50 – 2.49	Moderately Functional
1	1.0 – 1.49	Poor Functional

Table 3 presents the criteria used to evaluate a product or system's functional suitability according to the ISO survey.

**Table 4****Performance Efficiency**

Adjectival Rating	Scale Range (Mean)	Verbal Interpretation
4	3.50 – 4.0	Very Efficient
3	2.50 – 3.49	Efficient
2	1.50 – 2.49	Moderately Efficient
1	1.0 – 1.49	Poor Efficient

Table 4 outlines the scoring parameters used to assess the system's overall performance efficiency in executing its functions and tasks.

Table 5**Usability**

Adjectival Rating	Scale Range (Mean)	Verbal Interpretation
4	3.50 – 4.0	Very Usable
3	2.50 – 3.49	Usable
2	1.50 – 2.49	Moderately Usable
1	1.0 – 1.49	Poor Usable

Table 5 presents the scoring parameters used to evaluate how effectively, efficiently, and satisfactorily a system or product enables specific users to achieve their objectives within a given context of use.

**Table 6****Respondent's Distribution****Buenavista Municipal Social Welfare and Development**

Position	N (Population)	Percentage (%)
Admin	1	12.5%
Personnel	7	87.5%
Total	8	100

This table outlines the distribution of respondents who participated in evaluating the system within the Buenavista Municipal Social Welfare and Development. The respondents are categorized based on their positions, with 12.5% holding administrative roles and 87.5% being personnel. This indicates that the majority of the feedback comes from personnel who directly interact with the system. Since they are the primary users, their insights provide a more accurate assessment of the system's functionality, efficiency, and usability in real-world scenarios.

Descriptive Statistics Result**Table 7****Functional Suitability**

A. Functional Suitability	MEAN	VERBAL INTERPRETATION
1. Functional completeness - Degree to which the set of functions covers all the specified tasks and user objectives	3.50	VF
2. Functional correctness - Degree to which a product or system provides the correct	3.62	VF



results with the needed degree of

precision.

3. Functional appropriateness - Degree to

which the functions facilitate the

3.75

VF

accomplishment of specified tasks and

objectives.

Weighted Mean	3.62	VF
---------------	------	----

Legend:

VF – Very Functional (3.50 – 4.0)

MF – Moderately Functional (1.50 – 2.49)

F – Functional (2.50 – 3.49)

PF – Poorly Functional (1.0 – 1.49)

Table 7 focuses on the functional sustainability of the system, assessing whether it meets its intended functions effectively. It is evaluated based on three key aspects, functional completeness, correctness, and appropriateness. Functional completeness received a rating of 3.50, indicating that the system includes all necessary functions to support user objectives without major missing features. Functional correctness scored 3.62, suggesting that the system provides accurate and reliable outputs, ensuring that users can depend on it for precise information. Functional appropriateness received the highest rating of 3.75, highlighting that the system's functions are well-designed to facilitate task accomplishment efficiently. The overall weighted mean of 3.62 classifies the system as 'Very Functional' (VF), confirming that it effectively meets user needs by providing a complete, correct, and appropriate set of features.

**Table 8****Performance Efficiency**

B. Performance Efficiency	MEAN	VERBAL INTERPRETATION
1. Time behaviour - Degree to which the response and processing times and throughput rates of a system, when performing its functions, meet requirements.	3.75	VE
2. Resource utilization - Degree to which the amounts and types of resources used by a system, when performing its functions, meet requirements.	3.62	VE
3. Capacity - Degree to which the maximum limits of a product or system parameter meet requirements.	3.75	VE
Weighted Mean	3.70	VE
Legend:		
VE – Very Efficient (3.50 – 4.0)	ME – Moderately Efficient (1.50 – 2.49)	
E – Efficient (2.50 – 3.49)	PE – Poorly Efficient (1.0 – 1.49)	

Table 8 evaluates the performance efficiency of the system, determining how well it operates in terms of speed, resource utilization, and capacity. Time behavior, which measures response and processing speed, received a rating of 3.75, indicating that the system processes tasks quickly without unnecessary delays. Resource utilization scored 3.62, suggesting that the



system effectively manages computing resources without excessive consumption. Capacity, also rated at 3.75, demonstrates that the system can handle a high workload without compromising performance. The weighted mean of 3.70 categorizes the system as 'Very Efficient' (VE), meaning that it not only operates swiftly but also optimally utilizes resources and maintains stability even under significant usage demands.

Table 9**Usability**

C. Usability	MEAN	VERBAL INTERPRETATION
1. Appropriateness recognizability - Degree to which users can recognize whether a system is appropriate for their needs.	3.62	VU
2. Learnability - Degree to which a system can be used by specified users to achieve specified goals of learning to use the product or system with effectiveness, efficiency, freedom from risk and satisfaction in a specified context of use.	3.62	VU
3. Operability - Degree to which a system has attributes that make it easy to operate and control.	3.50	VU
4. User error protection - Degree to which a system protects users against making errors.	3.62	VU



5. User interface aesthetics - Degree to

which a user interface enables pleasing 3.62 VU
and satisfying interaction for the user.

6. Accessibility - Degree to which a system

can be used by people with the widest
range of characteristics and capabilities to 3.75 VU
achieve a specified goal in a specified
context of use.

Weighted Mean	3.62	VU
---------------	-------------	----

Legend:

VU – Very Usable (3.50 – 4.0) **MU** – Moderately Usable (1.50 – 2.49)

U – Usable (2.50 – 3.49) **PU** – Poorly Usable (1.0 – 1.49)

Table 9 examines the usability of the system, focusing on how user-friendly and accessible it is. Usability is assessed based on six factors: appropriateness, recognizability, learnability, operability, user error protection, user interface aesthetics, and accessibility. Appropriateness, recognizability, learnability, and user error protection all received a rating of 3.62, indicating that users find the system relevant to their needs, easy to learn, and equipped with mechanisms to prevent or correct errors. Operability, rated at 3.50, confirms that the system is easy to control and navigate, ensuring smooth user interaction. User interface aesthetics, also at 3.62, suggests that the design is visually appealing and enhances user satisfaction. Accessibility, with the highest rating of 3.75, highlights that the system is inclusive and can accommodate users with different abilities and needs. The weighted mean of 3.62 classifies the system as 'Very Usable' (VU), meaning that



users find it easy to interact with, navigate, and operate while benefiting from its well-structured design and accessibility features.

Table 10

Summary Table of the Over-all Mean and Grand Distribution of the Acceptability Level

Acceptability Level of the System in terms of:	Over-all Mean	Rating
Functional Suitability	3.62	SA
Performance Efficiency	3.70	SA
Usability	3.62	SA
Grand Mean	3.65	SA

Legend:

SA – Strongly Acceptable (3.50 – 4.0)

U – Unacceptable (1.50 – 2.49)

A – Acceptable (2.50 – 3.49)

SU – Strongly Unacceptable (1.0 – 1.49)

Table 10 provides a summary of the overall acceptability of the system, combining the results from functional suitability, performance efficiency, and usability. The system received a mean rating of 3.62 for functional suitability, confirming that it is ‘Very Functional’ (VF) and capable of effectively supporting user objectives. Performance efficiency was rated the highest, with a mean of 3.70, classifying it as ‘Very Efficient’ (VE), ensuring that the system processes tasks quickly and utilizes resources effectively. Usability also received a mean of 3.62, categorizing it as ‘Very Usable’ (VU), which means users find it easy to learn, operate, and navigate. With a grand mean of 3.65, the system is classified as ‘Strongly Acceptable’ (SA), signifying that it successfully meets the needs and expectations of its users.



CHAPTER 5

SUMMARY, CONCLUSION, AND RECOMMENDATION

This chapter presents a summary of the research's key findings, derives conclusions from the results, and suggests recommendations for future enhancements or further studies. It highlights the project's overall accomplishments, considers its implications, and offers practical insights into improving the system and expanding its potential applications.

5.1. Summary of Findings

The Web-Based Archival and Document Mapping System for Buenavista's Municipal Social Welfare Services was evaluated using the ISO 25010 Software Product Quality Standards to assess its functionality, performance efficiency, and usability. The assessment aimed to determine how well the system met user requirements and aligned with industry standards to support the operations of the Buenavista Municipal Social Welfare and Development department.

The results indicate that the system is highly functional, efficient, and user-friendly. In terms of functional suitability, it received a "Very Functional" (VF) rating with a mean score of 3.62, confirming that it effectively fulfills its intended functions. It provides comprehensive and accurate features that support users in completing tasks efficiently. For performance efficiency, the system was rated "Very Efficient" (VE) with a mean score of 3.70, demonstrating its capability to process tasks quickly, utilize resources effectively, and maintain stability even under high usage. While the system ensures smooth operations, ongoing monitoring and optimizations are recommended to maintain peak performance.

Overall, the system achieved a "Strongly Acceptable" (SA) rating, with a grand mean of 3.65, affirming its effectiveness in functionality, efficiency, and usability. Users found the system intuitive, accessible, and easy to navigate, leading to smooth and efficient interactions. While the system already meets user expectations, further refinements in operability, customization



options, and feature expansion could enhance its long-term effectiveness and adaptability.

Continuous improvements in system optimization and usability enhancements will ensure that it remains a reliable and valuable tool for the department's operations.

5.2. Conclusion

In conclusion, the Web-Based Archival and Document Mapping System for Buenavista's Municipal Social Welfare Services effectively met its intended objectives by providing a functional, efficient, and user-friendly platform for managing and accessing archival records. The system demonstrated strong functional suitability, ensuring that it met both stated and implied user requirements. Its high-performance efficiency allowed for quick data processing, optimal resource utilization, and stable operation even under heavy usage. Additionally, the system's usability ensured ease of navigation, accessibility, and intuitive interaction, making it a practical tool for the department. While minor improvements, such as enhanced customization and further usability refinements, could have optimized the user experience, the system successfully streamlined document management, improved accessibility, and enhanced operational efficiency. Overall, it served as a reliable and valuable tool that supported the department's archival and documentation processes effectively.

5.3. Recommendation/s

With these enhancements, the Web-Based Archival and Document Mapping System will evolve into a more efficient, reliable, and user-centric tool, capable of meeting the growing demands of its users while ensuring secure and well-organized record management. Improving database indexing and search algorithms will enable faster and more accurate document retrieval, ensuring seamless access even during peak usage. Providing comprehensive training materials will equip users with the necessary knowledge and skills to operate the system effectively, reducing errors and minimizing dependence on technical support. Streamlining the user interface with



intuitive navigation and error-prevention features will enhance usability, improve user satisfaction, and minimize workflow disruptions.

Moreover, advanced functionalities such as real-time document tracking and automated classification tools can further empower administrators to monitor changes, organize files systematically, and prevent data loss. Expanding access to additional stakeholders, such as department heads and municipal staff, will allow them to retrieve records directly, eliminating the need for manual requests and significantly improving overall operational efficiency. This added accessibility promotes greater transparency and ensures that relevant personnel can access the information they need promptly.

Based on the recommendation of the MSWD Buenavista, the system should incorporate a feature that enables departments to create and manage folders according to their designated personnel. This functionality will enhance the organization and accessibility of documents by allowing each department to establish a clear folder structure tailored to their operational needs. By assigning folders to specific personnel, departments can streamline the storage, tracking, and retrieval of records, ensuring accountability and efficiency in managing departmental files. This structured approach not only improves workflow but also supports data integrity and ease of access, which are essential for effective governance and service delivery.

To better align with institutional needs, refinements to document mapping and reporting capabilities are recommended, such as incorporating customized categorization and metadata tagging for easier organization. Upgrading the system's interface with a more visually appealing and user-friendly design will further enhance the overall user experience.

To future researchers, the researchers suggest enhancing storage capacity and data retrieval processes to optimize the Web-Based Archival and Document Mapping System for managing larger volumes of records efficiently.



ETHICAL STANDARD

Ethical considerations are fundamental to maintaining the integrity, transparency, and credibility of this study. To uphold these principles, the research prioritizes the confidentiality and security of sensitive records managed within the web-based archival and document mapping system. Access to these records is strictly limited to authorized personnel, ensuring protection against unauthorized access and misuse. Moreover, the system is designed to align with ethical standards of fairness, accountability, and respect for all stakeholders, adhering to institutional policies and legal frameworks governing Buenavista's Municipal Social Welfare Services.

A. Protection of Intellectual Property Rights (IPR)

This study involves the development of the Web-Based Archival and Document Mapping System for Buenavista's Municipal Social Welfare Services, which includes the creation of innovative technologies, processes, and products requiring patent protection. These advancements are designed to improve archival management and document mapping for municipal operations. Hard copies of materials, including reports, surveys, and system documentation, are integral to this project. These materials will be safeguarded through copyright protection to ensure their originality and proper attribution. Since no brand, logo, or symbol is associated with this project, trademark registration is not applicable.

B. Informed Consent

This research involves conducting interviews to assess the needs and requirements of the municipal office and identify opportunities for system improvements. Participants were asked about their current challenges and the support needed to enhance office operations. Prior to the interviews, each participant was provided with a letter of consent outlining the study's purpose, the interview process, and potential benefits or implications. Participants reviewed the letter and voluntarily agreed to take part in the study. The interviews were conducted verbally, focusing on



the specific system requirements of the office, the scope of the proposed system, common challenges faced by employees, and the potential of the system to benefit both the office and the community. This study does not involve any animal or live organism trials; all collected information was treated with strict confidentiality and used solely for research purposes.

C. Data Privacy and Confidentiality

The data collected in this study will be protected through the implementation of Role-Based Access Control (RBAC). This approach ensures that only authorized personnel can access specific files based on their assigned roles. The personnel will have access solely to the documents relevant to their responsibilities, and they will not be able to view or modify files outside their authorization. To maintain strict confidentiality, no anonymous data will be shared with anyone. The management of access rights ensures that all data remains secure and protected. Full access to all files will be exclusively granted to the admin of the system, who will oversee the entire system. This layered control guarantees that sensitive information is handled with the utmost security and accountability.

D. Voluntary Participation and Freedom to Withdraw

Participation in this study is completely voluntary, and all participants will be fully informed of their right to withdraw at any time without any adverse consequences. Clear details regarding the study's objectives, processes, and participant rights will be provided to ensure informed consent. The research will strictly follow ethical guidelines to safeguard and respect the rights of every participant involved.

E. Minimization of Harm and Risk Management

The researchers are dedicated in ensuring the well-being of all participants by proactively minimizing any potential harm and effectively managing possible risks. All questions, discussions, and interactions will be conducted in a respectful and thoughtful manner, eliminating the



possibility of emotional, psychological, or social discomfort. To further protect participants, all data collected will be treated with the utmost confidentiality. Sensitive information will remain secure and anonymous, ensuring that participants are not subjected to any negative consequences or risks as a result of their involvement in the study. The research team is committed to upholding ethical standards and prioritizing participant safety throughout the study.

F. Beneficence and Contribution to knowledge

The development of the Web-Based Archival and Document Mapping System for Buenavista's Municipal Social Welfare Services contributes positively to both the academic community and society by addressing the critical need for efficient document management in municipal operations. This system provides valuable insights into the application of technology in public service, serving as a model for innovation in local governance. For the academic community, the project offers a practical resource for research and learning, while for society, it ensures improved access to social welfare services, enhanced transparency, and better public trust.

The project findings and system capabilities will be shared with municipal stakeholders, fostering collaboration and empowering them to make informed decisions for future improvements. By streamlining operations, safeguarding data, and enhancing service delivery, the system promotes sustainable administrative practices that benefit the community and ensure long-term positive impacts on local governance and public welfare.

G. Justice and Fair Participant Selection

By developing a web-based system that effectively archives a variety of documents, such as administrative records, financial records, legal records, personnel records, and social services records, from the local area or community, this study will evaluate the employees of Buenavista's Municipal Social Welfare Services and potentially save a significant amount of time and effort while seeking and maintaining various records. Participants will be chosen based on clear and fair



inclusion and exclusion criteria, ensuring justice and fairness in the selection process. All human participants, regardless of background, will be included with appropriate consideration and without discrimination. The selection process will be designed to avoid any form of bias. No animals are included in this study. The documents will be managed responsibly, ensuring secure and organized handling for easy retrieval and maintenance.

H. Data Integrity and Accuracy

The system design emphasizes ensuring data integrity by implementing Role-Based Access Control (RBAC) to restrict access to authorized users only. Sensitive document integrity and security are therefore guaranteed. Furthermore, techniques such as organized database architecture assist in maintaining accuracy in document management and retrieval. Moreover, the research does not have any potential biases or error, however it does have a limitation which is only implemented in BUENAVISTA'S MUNICIPAL SOCIAL WELFARE SERVICES.

I. Transparency and Honesty in Reporting

The findings of this study will be clearly reported without any data manipulation, and any potential conflict of interest will be declared. Third-party materials will be properly cited to ensure transparency and respect for intellectual property rights. Additionally, this study has utilized AI tools, including ChatGPT and Grammarly, to enhance the research process, ensuring clarity, accuracy, and adherence to ethical standards.

J. Use of Patented or Copyrighted Materials

This research will not utilize any patented materials, as the focus of the study is on archival management and document mapping, areas that do not require the use of patented technologies. Regarding copyrighted materials, any content that falls under copyright protection, such as previous research or literature, will be credited. This approach ensures compliance with copyright laws and ethical use of intellectual property.



K. Ethical Considerations for Animal and Human Trials

The researchers are committed to ethical standards that prioritize the well-being of all living beings. The study does not indicate any involvement of animals and human trials. The study focuses more on developing a digital archival system to improve document management and retrieval efficiency for a municipal welfare department. So, it does not come under human or animal trials that involve ethical considerations.

L. Responsible Use of AI and Related Technologies

To enhance the documentation's quality, the researchers also employed Grammarly. ChatGPT is also used in the study to assist in giving ideas to enhance the content of the documentation. AI tools are being utilized to meet ethical standards, making sure that the recommendations they produce are in line with moral principles and are carefully examined for correctness and applicability. The researcher also reviewed and validated the study to maintain integrity and professionalism of the research.

M. Ethical Clearance and Institutional Approval

Ethical approval will be granted by the Saint Michael College of Caraga Institutional Review Board. Thus, the process ensures accordance with established research ethical guidelines while securing participants intellectual property rights, safety, and data confidentiality. To guarantee the ethical integrity and legality of the research project, the researcher will adhere closely to the board's guidelines.



REFERENCES

- [1] N. Syamia, A. Shadrina Lubis, N. Hera Zabni, and M. Ikhsan Rifki, “WEB-BASED DOCUMENT ARCHIVING INFORMATION SYSTEM IN COMMISSION C DPRD OF NORTH SUMATRA PROVINCE,” *SAINTEKBU: Journal of Science and Technology*, no. 01, p. 16, 2024.
- [2] B. A. III Rodriguez, V. T. Aquiatan, C. Jun Verallo, S. B. Agpad, R. C. de Loyola, and E. P. Jireh Bibangco, “System for the DILG Negros Occidental,” 2024.
- [3] M. L. D. L. J. T. C. R. K. A. R. M. L. P. C. P. C. E. Niño Derex C. Nagrama1, “Web-based Document Management System,” *International Journal of Science and Applied Information Technology*, vol. 13, no. 3, pp. 16–21, Jun. 2024, doi: 10.30534/ijasit/2024/031332024.
- [4] M. Churiyah, N. Muhamiroh, M. Arief, B. A. Dharma, and A. Basuki, “Improving Student Archival Management Competency by Using Codeigniter Web-Based e-Archive,” 2021.
- [5] R. M. Santiañez and B. M. Sollano, “Development and Evaluation of Local Area Network Based Archiving System,” *American Journal of Agricultural Science, Engineering and Technology*, vol. 5, no. 2, pp. 286–296, Nov. 2021, doi: 10.54536/ajaset.v5i2.107.
- [6] F. V Castro, E. Rose, C. Quilon, and A. C. Arruejo, “DocTrack: A Dynamic Document Tracking System for the University of Northern Philippines,” *Journal of Innovative Technology Convergence*, vol. 4, no. 2, pp. 23–34, 2022.
- [7] N. Soveizi, F. Turkmen, and D. Karastoyanova, “Security and privacy concerns in cloud-based scientific and business workflows: A systematic review,” *Future Generation Computer Systems*, vol. 148, pp. 184–200, Nov. 2023, doi: 10.1016/J.FUTURE.2023.05.015.
- [8] I. A. Justina, O. E. Abiodun, and O. O. M., “A Secured Cloud-Based Electronic Document Management System,” *International Journal of Innovative Research and Development*, Dec. 2022, doi: 10.24940/IJIRD/2022/V11/I12/DEC22010.



-
- [9] M. G. Subia, R. Corpuz, and R. R. Corpuz, “Archiving And Digitizing Of Customer Records Of Golden Rural Bank Of The Philippines, Inc,” *INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH*, vol. 9, p. 1, 2020, [Online]. Available: www.ijstr.org
- [10] S. Jordan, S. S. Zabukovšek, and I. Š. Klančnik, “Document Management System – A Way to Digital Transformation,” *Naše gospodarstvo/Our economy*, vol. 68, no. 2, pp. 43–54, Jun. 2022, doi: 10.2478/NGOE-2022-0010.
- [11] N. R. Zahara and T. A. Salim, “Preservation of Digital Archives: Systematic Literature Review,” Dec. 01, 2022, *Airlangga University Faculty of Vocational Studies*. doi: 10.20473/rlj.V8-I2.2022.285-297.
- [12] A. Y. A Bani Ahmad *et al.*, “International Journal of INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING Framework for Cloud Based Document Management System with Institutional Schema of Database,” 2023. [Online]. Available: www.ijisae.org
- [13] L. Xing, “Secure Official Document Management and intelligent Information Retrieval System based on recommendation algorithm,” *International Journal of Intelligent Networks*, vol. 5, pp. 110–119, Jan. 2024, doi: 10.1016/J.IJIN.2024.02.003.
- [14] U. M. Melendres and K. M. Aranda, “Development and Evaluation of a Web-Based Resident Information Management System,” *Journal of Computer, Software, and Program*, vol. 1, no. 1, pp. 14–22, Jun. 2024, doi: 10.69739/jcsp.v1i1.50.
- [15] M. V. Gamido, H. V. Gamido, and D. J. P. Macaspac, “Electronic document management system for local area network-based organizations,” *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 31, no. 2, pp. 1154–1163, Aug. 2023, doi: 10.11591/ijeecs.v31.i2.pp1154-1163.



-
- [16] C. L. Amethyst M. and C. Roderic P., “Electronic Document Archival System of STO. Niño National High School,” *International Journal of Scientific and Management Research*, vol. 05, no. 06, pp. 01–09, 2022, doi: 10.37502/ijsmr.2022.5601.
- [17] Gracelyn T Condez, “Digital document repository system of researchesutilizing search engine tools,” *South Asian Journal of Engineering and Technology*, vol. 12, no. 1, pp. 18–23, Mar. 2022, doi: 10.26524/sajet.2022.12.04.
- [18] D. A. Mallares, W. Custer, G. Alegata, D. V Madrigal, and J. M. Bual, “Admission and Registrar Office (ARO) Mobile Document Scanner with Archiving System.” [Online]. Available: www.techniumscience.com
- [19] E. A. Bacalso, L. Z. Vincent Pleto, R. P. Jerome Fernandez, and A. M. Jain, “Integration of Process Monitoring in Procurement Management System for Bureau of Fire Protection CALABARZON,” 2024.
- [20] P. Chen, “Design and Implementation of New Media Manuscript Retrieval System,” Scitepress, Mar. 2024, pp. 333–337. doi: 10.5220/0012282900003807.
- [21] Y. Zhu *et al.*, “Large Language Models for Information Retrieval: A Survey,” Aug. 2023, [Online]. Available: <http://arxiv.org/abs/2308.07107>
- [22] S. Kumar, S. Arun, K. Ray, B. Pati, U. Rajendra, and A. Editors, “Lecture Notes in Electrical Engineering 728 Proceedings of International Conference on Communication, Circuits, and Systems,” 2020. [Online]. Available: <http://www.springer.com/series/7818>
- [23] A. Ben Ayed, I. Biskri, and J.-G. Meunier, “An End-to-End Efficient Lucene-Based Framework of Document/Information Retrieval,” *International Journal of Information Retrieval Research*, vol. 12, no. 1, pp. 1–14, Oct. 2021, doi: 10.4018/ijirr.289950.
- [24] P. Mahalakshmi and N. S. Fathima, “An Art of Review on Conceptual based Information Retrieval,” *Webology*, vol. 18, pp. 22–31, 2021, doi: 10.14704/WEB/V18SI02/WEB18009.
-



-
- [25] U. Mohammed Bala and M. M. Muhammad, “Analysis and Design of File Tracking System,” *Journal of Applied Sciences, Information, and Computing*, vol. 1, no. 2, 2020, [Online]. Available: <https://jasic.kiu.ac.ug>
- [26] S. F. Salleh, H. Ujir, R. Sapawi, and H. F. Hashim, “Accreditation Document Tracking System Using Scrum Approach.,” *International Journal of Evaluation and Research in Education*, vol. 9, no. 1, pp. 153–161, Mar. 2020, doi: 10.11591/ijere.v9i1.20418.
- [27] N. Bernard and K. Balog, “A Systematic Review of Fairness, Accountability, Transparency and Ethics in Information Retrieval,” *ACM Comput Surv*, Dec. 2023, doi: 10.1145/3637211.
- [28] E. Avuçlu and S. Yalçın, “A Web-Based Advanced Law Firm Tracking System Application For Lawyers,” *International Scientific and Vocational Studies Journal*, vol. 8, no. 1, pp. 76–86, Jun. 2024, doi: 10.47897/bilmes.1495894.
- [29] S. Sofik and Z. Rahman, “Global Visualization and Knowledge Mapping in the Field of Information Retrieval (IR): A Bibliometrics Analysis,” *Qualitative and Quantitative Methods in Libraries (QQML)*, vol. 10, pp. 623–647, 2021, [Online]. Available: <https://orcid.org/0000-0002-3393-8690700050>.
- [30] C. Madubuike, C. E. Madubuike, M. C. Okoronkwo, G. E. Oko, and T. A. Oyeniran, “Document tracking system for Akanu Ibiam Federal Polytechnic Unwana, Afikpo: Design guidelines and model implementation DOCUMENT TRACKING SYSTEM FOR AKANU IBIAM FEDERAL POLYTECHNIC... Document tracking system for Akanu Ibiam Federal Polytechnic Unwana, Afikpo: Design guidelines and model implementation DOCUMENT TRACKING SYSTEM FOR AKANU IBIAM FEDERAL POLYTECHNIC UNWANA, AFIKPO: DESIGN GUIDELINES AND MODEL IMPLEMENTATION.” [Online]. Available: <https://www.researchgate.net/publication/369618739>



-
- [31] K. Myaing, M. Tun, T. Thi, and S. Nyunt, “SECURITY OF HEALTHCARE SYSTEM USING ROLE-BASED ACCESS CONTROL.”
 - [32] G. Liu, R. Zhang, B. Wan, S. Ji, and Y. Tian, “Extended role-based access control with context-based role filtering,” in *KSII Transactions on Internet and Information Systems*, Korean Society for Internet Information, 2020, pp. 1263–1279. doi: 10.3837/tiis.2020.03.019.
 - [33] M. A. De Carvalho Junior and P. Bandiera-Paiva, “Health Information System Role-Based Access Control Current Security Trends and Challenges,” 2018, *Hindawi Limited*. doi: 10.1155/2018/6510249.
 - [34] M. Uddin, S. Islam, and A. Al-Nemrat, “A Dynamic Access Control Model Using Authorising Workflow and Task-Role-Based Access Control,” *IEEE Access*, vol. 7, pp. 166676–166689, 2019, doi: 10.1109/ACCESS.2019.2947377.
 - [35] A. Chatterjee, Y. Pitroda, and M. Parmar, “Dynamic Role-Based Access Control for Decentralized Applications,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer Science and Business Media Deutschland GmbH, 2020, pp. 185–197. doi: 10.1007/978-3-030-59638-5_13.
 - [36] R. Ghazal, A. K. Malik, N. Qadeer, B. Raza, A. R. Shahid, and H. Alquhayz, “Intelligent Role-Based Access Control Model and Framework Using Semantic Business Roles in Multi-Domain Environments,” *IEEE Access*, vol. 8, pp. 12253–12267, 2020, doi: 10.1109/ACCESS.2020.2965333.



APPENDICES

A. System Sources Code

Database Connection

```

1  <?php
2  // database connection
3  $host = "localhost";
4  $username = "root";
5  $password = "";
6  $database = "admms";
7
8  $conn = new mysqli($host, $username, $password, $database);
9
10 if ($conn->connect_error) {
11     die(json_encode([
12         'success' => false,
13         'message' => "Connection failed: " . $conn->connect_error
14     ]));
15 }
16 ?>

```

chart.php

```

1  <?php
2  // chart functions (admin | dashboard)
3  require 'conn.php';
4
5  $fromDate = isset($_GET['fromDate']) ? $_GET['fromDate'] : null;
6  $toDate = isset($_GET['toDate']) ? $_GET['toDate'] : null;
7
8  // empty chart arrays (default)
9  $data = [
10     'departments' => [],
11     'files' => [],
12     'folders' => [],
13     'stackedFiles' => []
14 ];
15
16 $deptQuery = "SELECT department_name, folder_color FROM department";
17 $result = $conn->query($deptQuery);
18 while ($row = $result->fetch_assoc()) {
19     $data['departments'][] = $row;
20 }
21
22 $fileQuery = "SELECT department, upload_date FROM files";
23 if ($fromDate && $toDate) {
24     $fileQuery .= " WHERE DATE(upload_date) BETWEEN '$fromDate' AND '$toDate'";
25 }
26 $result = $conn->query($fileQuery);
27 while ($row = $result->fetch_assoc()) {
28     $data['files'][] = $row;
29 }
30
31 $folderQuery = "SELECT department, created_at FROM folders";
32 if ($fromDate && $toDate) {
33     $folderQuery .= " WHERE DATE(created_at) BETWEEN '$fromDate' AND '$toDate'";
34 }
35 $result = $conn->query($folderQuery);
36 while ($row = $result->fetch_assoc()) {
37     $data['folders'][] = $row;
38 }
39
40 $stackedQuery = "SELECT department, DATE(upload_date) as upload_date, COUNT(*) as count FROM files";
41 if ($fromDate && $toDate) {
42     $stackedQuery .= " WHERE DATE(upload_date) BETWEEN '$fromDate' AND '$toDate'";
43 }
44 $stackedQuery .= " GROUP BY department, upload_date ORDER BY upload_date";
45 $result = $conn->query($stackedQuery);
46 while ($row = $result->fetch_assoc()) {
47     $data['stackedFiles'][] = $row;
48 }
49
50 header('Content-Type: application/json');
51 echo json_encode($data);
52

```



check_duplicate.php

```

1  <?php
2  session_start();
3  require_once 'conn.php';
4
5  header('Content-Type: application/json');
6
7  try {
8      $data = json_decode(file_get_contents('php://input'), true);
9
10     if (!isset($data['type']) || !isset($data['name'])) {
11         throw new Exception('Missing required parameters');
12     }
13
14     $type = $data['type'];
15     $name = $data['name'];
16
17     if ($type === 'folder') {
18         $parent_folder_id = $data['parent_folder_id'] ?? null;
19
20         if ($parent_folder_id) {
21             $query = "SELECT COUNT(*) as count FROM folders
22                         WHERE folder_name = ? AND parent_folder = ?";
23             $stmt = $conn->prepare($query);
24             $stmt->bind_param("si", $name, $parent_folder_id);
25         } else {
26             $query = "SELECT COUNT(*) as count FROM folders
27                         WHERE folder_name = ? AND (parent_folder IS NULL OR parent_folder = 0)";
28             $stmt = $conn->prepare($query);
29             $stmt->bind_param("s", $name);
30         }
31
32     } else if ($type === 'file') {
33         $folder_id = $data['folder_id'];
34
35         if (!$folder_id) {
36             throw new Exception('Folder ID is required for file duplicate check');
37         }
38
39         $query = "SELECT COUNT(*) as count FROM files
40                         WHERE original_name = ? AND folder_id = ?";
41         $stmt = $conn->prepare($query);
42         $stmt->bind_param("si", $name, $folder_id);
43
44     } else {
45         throw new Exception('Invalid type specified');
46     }
47
48     $stmt->execute();
49     $result = $stmt->get_result();
50     $row = $result->fetch_assoc();
51
52     echo json_encode([
53         'success' => true,
54         'exists' => ($row['count'] > 0)
55     ]);
56
57 } catch (Exception $e) {
58     echo json_encode([
59         'success' => false,
60         'message' => $e->getMessage()
61     ]);
62 }
63
64 $conn->close();
65 ?>
```

create.php

```

1  <?php
2  header('Content-Type: application/json');
3
4  include 'conn.php';
5  session_start();
6
7  if ($_SERVER['REQUEST_METHOD'] === 'POST') {
8
9      $action = $_POST['action'] ?? json_decode(file_get_contents('php://input'), true)['action'];
10
11     if ($action === 'create_account') {
12         $full_name = $conn->real_escape_string($_POST['full_name']);
13         $username = $conn->real_escape_string($_POST['username']);
14         $department = $conn->real_escape_string($_POST['department']);
15         $address = $conn->real_escape_string($_POST['address']);
16         $contact_info = $conn->real_escape_string($_POST['contact_info']);
17         $role = 'User';
18         $status = 'enabled';
19         $password = password_hash($conn->real_escape_string($_POST['password']), PASSWORD_BCRYPT);
20
21         $sql = "INSERT INTO user (full_name, username, department, address, contact_info, password, role, status, created_at)
22                 VALUES ('$full_name', '$username', '$department', '$address', '$contact_info', '$password', '$role', '$status', NOW())";
23
24         if ($conn->query($sql) === TRUE) {
25             echo json_encode([
26                 'success' => true,
27                 'message' => 'Account created successfully!'
28             ]);
29         } else {
30             echo json_encode([
31                 'success' => false,
32                 'message' => "Error: " . $conn->error
33             ]);
34         }
35     } elseif ($action === 'create_folder') {
36         date_default_timezone_set('Asia/Manila');
37         $currentDate = date('Y-m-d');
38     }
39
40 }
```



```
38
39     $folderName = $_POST['folderName'];
40     $year = $_POST['year'];
41     $department = $_POST['department'];
42
43     $stmt = $conn->prepare("INSERT INTO folders (folder_name, year, department, created_at) VALUES (?, ?, ?, ?)");
44     $stmt->bind_param("ssss", $folderName, $year, $department, $currentDate);
45
46     if ($stmt->execute()) {
47         echo json_encode([
48             'success' => true,
49             'message' => 'Folder created successfully!'
50         ]);
51     } else {
52         echo json_encode([
53             'success' => false,
54             'message' => "Error: " . $stmt->error
55         ]);
56     }
57
58     $stmt->close();
59 } elseif ($action === 'create_subfolder') {
60     $data = json_decode(file_get_contents('php://input'), true);
61
62     if (empty($data['folderName']) || empty($data['parentFolderId'])) {
63         echo json_encode(['success' => false, 'message' => "All fields are required."]);
64         exit;
65     }
66
67     $folderName = $data['folderName'];
68     $parentFolderId = $data['parentFolderId'];
69     $metadata = $data['metadata'] ?? [];
70     $userId = $_SESSION['user_id'];
71     $handleDuplicate = $data['handleDuplicate'] ?? false;
72
73     $conn->begin_transaction();
```

```
74
75     try {
76         $query = "SELECT year, department FROM folders WHERE folder_id = ?";
77         $stmt = $conn->prepare($query);
78         $stmt->bind_param('i', $parentFolderId);
79         $stmt->execute();
80         $result = $stmt->get_result();
81
82         if ($result->num_rows === 0) {
83             throw new Exception("Parent folder not found.");
84         }
85
86         $parentFolder = $result->fetch_assoc();
87         $year = $parentFolder['year'];
88         $department = $parentFolder['department'];
89         $createdAt = date('Y-m-d H:i:s');
90
91         if ($handleDuplicate) {
92             $baseNameQuery = "SELECT folder_name FROM folders WHERE parent_folder = ? AND folder_name LIKE ?";
93             $baseNameStmt = $conn->prepare($baseNameQuery);
94             $searchPattern = "%$folderName%";
95             $baseNameStmt->bind_param('is', $parentFolderId, $searchPattern);
96             $baseNameStmt->execute();
97             $baseNameResult = $baseNameStmt->get_result();
98
99             $existingNames = [];
100            while ($row = $baseNameResult->fetch_assoc()) {
101                $existingNames[] = $row['folder_name'];
102            }
103
104            $newName = $folderName;
105            $counter = 2;
106
107            while (in_array($newName, $existingNames)) {
108                $newName = $folderName . ' ' . $counter;
109                $counter++;
110            }
111        }
112    }
```

```
111
112     $folderName = $newName;
113 }
114
115 $query = "INSERT INTO folders (folder_name, year, department, created_at, parent_folder) VALUES (?, ?, ?, ?, ?)";
116 $stmt = $conn->prepare($query);
117 $stmt->bind_param('ssssi', $folderName, $year, $department, $createdAt, $parentFolderId);
118
119 if (!$stmt->execute()) {
120     throw new Exception("Failed to create folder: " . $stmt->error);
121 }
122
123 $newFolderId = $conn->insert_id;
124
125 if (empty($metadata)) {
126     $metadataQuery = "INSERT INTO metadata_title (folder_id, meta_data_title, input_type, user_id) VALUES (?, ?, ?, ?)";
127     $metadataStmt = $conn->prepare($metadataQuery);
128
129     foreach ($metadata as $meta) {
130         $metadataStmt->bind_param('isii', $newFolderId, $meta['title'], $meta['type'], $userId);
131         if (!$metadataStmt->execute()) {
132             throw new Exception("Failed to insert metadata: " . $metadataStmt->error);
133         }
134     }
135     $metadataStmt->close();
136 }
137
138 $conn->commit();
139 echo json_encode(["success" => true, "message" => "Folder created successfully."]);
140
141 } catch (Exception $e) {
142     $conn->rollback();
143     echo json_encode(["success" => false, "message" => $e->getMessage()]);
144 } finally {
145     $stmt->close();
146 }
147 } elseif ($action === 'add_metadata_title') {
148     try {
149         $data = json_decode(file_get_contents('php://input'), true);
```



```

149     $data = json_decode(file_get_contents('php://input'), true);
150
151     if (!isset($_SESSION['user_id'])) {
152         throw new Exception('User not authenticated');
153     }
154
155     if (!isset($data['folder_id']), !isset($data['meta_data_title']), !isset($data['input_type'])) {
156         throw new Exception('Missing required fields');
157     }
158
159     $folder_id = (int) $data['folder_id'];
160     $meta_data_title = $conn->real_escape_string($data['meta_data_title']);
161     $input_type = $conn->real_escape_string($data['input_type']);
162     $user_id = (int) $_SESSION['user_id'];
163
164     $query = "INSERT INTO metadata_title (folder_id, meta_data_title, input_type, user_id) VALUES (?, ?, ?, ?)";
165     $stmt = $conn->prepare($query);
166     $stmt->bind_param("isii", $folder_id, $meta_data_title, $input_type, $user_id);
167     $stmt->execute();
168     $stmt->close();
169
170     echo json_encode(['success' => true]);
171 } catch (Exception $e) {
172     echo json_encode(['success' => false, 'message' => $e->getMessage()]);
173 }
174
175 elseif ($action === 'add_department') {
176     try {
177         if (!isset($_SESSION['user_id']) || $_SESSION['role'] !== 'Admin') {
178             throw new Exception('Unauthorized access');
179         }
180
181         $department_name = trim($_POST['department_name'] ?? '');
182         $folder_color = trim($_POST['folder_color'] ?? '');
183
184         $folder_color = str_replace('#', ' ', $folder_color);
185
186         if (empty($department_name) || empty($folder_color)) {
187             throw new Exception('All fields are required');
188         }
189
190         $check_query = "SELECT id FROM department WHERE department_name = ?";
191         $check_stmt = $conn->prepare($check_query);
192         if (!$check_stmt) {
193             throw new Exception('Database prepare error: ' . $conn->error);
194         }
195
196         $check_stmt->bind_param("s", $department_name);
197         if (!$check_stmt->execute()) {
198             throw new Exception('Database execution error: ' . $check_stmt->error);
199         }
200
201         $result = $check_stmt->get_result();
202         if ($result->num_rows > 0) {
203             throw new Exception('Department name already exists');
204         }
205
206         $query = "INSERT INTO department (department_name, folder_color) VALUES (?, ?)";
207         $stmt = $conn->prepare($query);
208         if (!$stmt) {
209             throw new Exception('Database prepare error: ' . $conn->error);
210         }
211
212         $stmt->bind_param("ss", $department_name, $folder_color);
213         if (!$stmt->execute()) {
214             throw new Exception('Database execution error: ' . $stmt->error);
215         }
216
217         echo json_encode([
218             'success' => true,
219             'message' => 'Department added successfully'
220         ]);
221
222     } catch (Exception $e) {
223         echo json_encode([
224             'success' => false,
225             'message' => $e->getMessage()
226         ]);
227     } finally {
228         if (isset($stmt))
229             $stmt->close();
230         if (isset($check_stmt))
231             $check_stmt->close();
232     }
233
234 } else {
235     echo json_encode([
236         'success' => false,
237         'message' => 'Invalid action!'
238     ]);
239 }
240
241 $conn->close();

```



db_functions.php

```

1 <?php
2
3 function getFilePath($conn, $folder_id)
4 {
5     $path = [];
6
7     while ($folder_id) {
8         $folder_query = "SELECT folder_name, parent_folder FROM folders WHERE folder_id = ?";
9         $stmt = mysqli_prepare($conn, $folder_query);
10        mysqli_stmt_bind_param($stmt, 'i', $folder_id);
11        mysqli_stmt_execute($stmt);
12        $result = mysqli_stmt_get_result($stmt);
13
14        if ($row = mysqli_fetch_assoc($result)) {
15            array_unshift($path, $row['folder_name']);
16            $folder_id = $row['parent_folder'];
17        } else {
18            break;
19        }
20    }
21
22    return implode(' / ', $path);
23 }
24
25 function getMetadataTitles($conn)
26 {
27     $metadata_titles = [];
28     $title_query = "SELECT DISTINCT meta_data_title, metadata_id FROM metadata_title";
29     $title_result = mysqli_query($conn, $title_query);
30     while ($row = mysqli_fetch_assoc($title_result)) {
31         $metadata_titles[$row['metadata_id']] = $row['meta_data_title'];
32     }
33     return $metadata_titles;
34 }
35
36 function getDepartments($conn)
37 {
38     $departments = [];
39     $dept_query = "SELECT DISTINCT department_name FROM department WHERE department_name != 'admin'";
40     $dept_result = mysqli_query($conn, $dept_query);
41     while ($row = mysqli_fetch_assoc($dept_result)) {
42         $departments[] = $row['department_name'];
43     }
44     return $departments;
45 }
46
47 function getFiles($conn, $status, $user_id, $filters = [], $is_admin = false)
48 {
49     $file_query = "
50         SELECT f.file_id, f.original_name, f.upload_date, f.author, f.folder_id, f.user_id,
51             f.file_name, f.department, f.remarks, u.full_name
52         FROM files f
53         JOIN user u ON f.user_id = u.id
54         WHERE f.status = ?      Chat (CTRL + I) / Edit (CTRL + L)
55     ";
56
57     if (!$is_admin) {
58         $file_query .= " AND f.user_id = ?";
59     }
60
61     $where_clauses = [];
62     $params = [$status];
63
64     if (!$is_admin) {
65         $params[] = $user_id;
66     }
67
68     if (isset($filters['fromDate']) && !empty($filters['fromDate'])) {
69         $where_clauses[] = "f.upload_date >= ?";
70         $params[] = $filters['fromDate'];
71     }
72
73     if (isset($filters['toDate']) && !empty($filters['toDate'])) {
74         $where_clauses[] = "f.upload_date <= ?";
75         $params[] = $filters['toDate'];
76     }
77
78     if (isset($filters['department']) && !empty($filters['department'])) {
79         $where_clauses[] = "f.department = ?";
80         $params[] = $filters['department'];
81     }
82
83     if (!empty($where_clauses)) {
84         $file_query .= " AND ". implode(" AND ", $where_clauses);
85     }
86
87     $stmt = mysqli_prepare($conn, $file_query);
88     if (!empty($params)) {
89         $types = str_repeat('s', count($params));
90         mysqli_stmt_bind_param($stmt, $types, ...$params);
91     }
92     mysqli_stmt_execute($stmt);
93     return mysqli_stmt_get_result($stmt);
94 }
95
96 function getFileMetadata($conn, $file_id)
97 {
98     $metadata_query = "
99         SELECT mt.meta_data_title, md.meta_data_description
100        FROM metadata_description md
101        JOIN metadata_title mt ON md.title_id = mt.metadata_id
102        WHERE md.file_id = ?
103    ";
104    $stmt = mysqli_prepare($conn, $metadata_query);
105    mysqli_stmt_bind_param($stmt, 's', $file_id);
106    mysqli_stmt_execute($stmt);
107    return mysqli_stmt_get_result($stmt);
108 }

```



delete.php

```

1  <?php
2  error_reporting(E_ALL);
3  ini_set('log_errors', 1);
4  ini_set('error_log', 'php_errors.log');
5
6  require_once 'conn.php';
7
8  session_start();
9
10 header('Content-Type: application/json');
11
12 try {
13     $data = json_decode(file_get_contents('php://input'), true);
14
15     if (!isset($data['type'])) {
16         throw new Exception('Missing required field: type');
17     }
18
19     $type = $data['type'];
20
21     if ($type === 'metadata_title') {
22         // for deleting metadata title (admin | manage | folders (for subfolders only))
23         if (!isset($data['metadata_id'])) {
24             throw new Exception('Missing required field: metadata_id');
25         }
26
27         $metadata_id = (int) $data['metadata_id'];
28
29         $query = "DELETE FROM metadata_title WHERE metadata_id = ?";
30         $stmt = $conn->prepare($query);
31         $stmt->bind_param("i", $metadata_id);
32         $stmt->execute();
33         $stmt->close();
34
35         echo json_encode(['success' => true]);
36     } elseif ($type === 'folder_status') {
37         // for updating folder status (activate/deactivate) recursively
38         if (!isset($data['folder_id']) || !isset($data['status'])) {
39             throw new Exception('Missing required fields: folder_id or status');
40
41             throw new Exception('Missing required fields: folder_id or status');
42
43             $folder_id = $data['folder_id'];
44             $new_status = $data['status']; // Will be 'active' or 'inactive'
45
46             try {
47                 // Begin transaction
48                 $conn->begin_transaction();
49
50                 // Update folder status recursively
51                 $stats = updateFolderStatusRecursively($conn, $folder_id, $new_status);
52
53                 // Commit transaction
54                 $conn->commit();
55
56                 echo json_encode([
57                     'success' => true,
58                     'message' => "Folder and its subfolders updated to $new_status",
59                     'folders_updated' => $stats['folders_updated']
60                 ]);
61             } catch (Exception $e) {
62                 // Rollback transaction in case of error
63                 $conn->rollback();
64
65                 error_log("Error in delete.php: " . $e->getMessage());
66                 echo json_encode([
67                     'success' => false,
68                     'message' => 'An error occurred: ' . $e->getMessage()
69                 ]);
70             }
71         } elseif ($type === 'file') {
72             // handle file deletion -----
73             if (!isset($_SESSION['user_id'])) {
74                 throw new Exception('Unauthorized access');
75             }
76
77             if (!isset($data['file_id'])) {
78                 throw new Exception('File ID is required');
79             }
80
81             $fileId = $data['file_id'];
82             $userId = $_SESSION['user_id'];
83
84             // check ownership -----
85             $checkOwnership = $conn->prepare("SELECT file_name FROM files WHERE file_id = ? AND user_id = ?");
86             $checkOwnership->bind_param("ii", $fileId, $userId);
87             $checkOwnership->execute();
88             $result = $checkOwnership->get_result();
89
90             if ($result->num_rows === 0) {
91                 throw new Exception('Unauthorized to delete this file');
92             }
93
94             $fileData = $result->fetch_assoc();
95             $fileName = $fileData['file_name'];
96             $filePath = "../uploads/" . $fileName;
97
98             // start transaction -----
99             $conn->begin_transaction();
100
101             try {
102                 // delete file record -----
103                 $deleteFile = $conn->prepare("DELETE FROM files WHERE file_id = ? AND user_id = ?");
104                 $deleteFile->bind_param("ii", $fileId, $userId);
105                 $deleteFile->execute();
106
107                 // delete physical file -----
108                 if (file_exists($filePath)) {
109                     unlink($filePath);
110                 }
111
112                 // commit transaction -----
113                 $conn->commit();
114             }
115
116         }
117
118     }
119
120 }
```



```

113     echo json_encode(['success' => true]);
114 } catch (Exception $e) {
115     // rollback transaction on error
116     $conn->rollback();
117     throw new Exception('Database error occurred');
118 }
119 } else {
120     throw new Exception('Invalid type specified');
121 }
122 } catch (Exception $e) {
123     // rollback transaction if still in progress ----
124     if (isSet($conn) && $conn->in_transaction) {
125         $conn->rollback();
126     }
127 }
128 echo json_encode(['success' => false, 'message' => $e->getMessage()]);
129 }
130 }
131 $conn->close();
132 /**
133 * recursively update the status of a folder and its subfolders.
134 *
135 * @param mysqli $conn database conn.
136 * @param int $folder_id folder ID.
137 * @param string $status status to set ('active' / 'inactive').
138 * @return array stats about the folders updated.
139 */
140 function updateFolderStatusRecursively($conn, $folder_id, $status)
141 {
142     $stats = [
143         'folders_updated' => 0
144     ];
145 }
146
147
148 try {
149     // update the current folder's status
150     $update_folder = $conn->prepare("UPDATE folders SET status = ? WHERE folder_id = ?");
151     $update_folder->bind_param("si", $status, $folder_id);
152     $update_folder->execute();
153     $update_folder->close();
154     $stats['folders_updated']++;
155
156     // find and update subfolders recursively
157     $get_children = $conn->prepare("SELECT folder_id FROM folders WHERE parent_folder = ?");
158     $get_children->bind_param("i", $folder_id);
159     $get_children->execute();
160     $children_result = $get_children->get_result();
161
162     while ($child = $children_result->fetch_assoc()) {
163         $child_stats = updateFolderStatusRecursively($conn, $child['folder_id'], $status);
164         $stats['folders_updated'] += $child_stats['folders_updated'];
165     }
166     $get_children->close();
167
168     return $stats;
169 } catch (Exception $e) {
170     error_log("Error in updateFolderStatusRecursively: " . $e->getMessage());
171     throw $e; // re-throw to be caught in the calling function
172 }
173 }
174 ?>
```

folder_functions.php

```

1 <?php
2 // folder functions (used for: get_folders.php)
3 require_once 'conn.php';
4
5 function getFolderColor($department)
6 {
7     global $conn;
8
9     $stmt = $conn->prepare("SELECT folder_color FROM department WHERE LOWER(department_name) = LOWER(?)");
10    $stmt->bind_param("s", $department);
11    $stmt->execute();
12    $stmt->bind_result($folderColor);
13
14    if ($stmt->fetch()) {
15        $stmt->close();
16        return '#' . $folderColor;
17    } else {
18        $stmt->close();
19        return '#6c757d';
20    }
21 }
22
23 function getFileIcon($extension)
24 {
25     $icons = [
26         'pdf' => 'bi-file-pdf',
27         'doc' => 'bi-file-word',
28         'docx' => 'bi-file-word',
29         'xls' => 'bi-file-excel',
30         'xlsx' => 'bi-file-excel',
31         'ppt' => 'bi-file-ppt',
32         'pptx' => 'bi-file-ppt',
33         'txt' => 'bi-file-text',
34         'rtf' => 'bi-file-text'
35     ];
36
37     return isSet($icons[strtolower($extension)]) ? $icons[strtolower($extension)] : 'bi-file';
38 }
39 ?>
```



get_folders.php

```

1  <?php
2  session_start();
3
4
5  require_once 'conn.php';
6  require_once 'folder_functions.php';
7
8  $parent_folder_id = isset($_GET['folder_id']) ? intval($_GET['folder_id']) : null;
9  $user_department = $_SESSION['department'];
10 $role = $_SESSION['role'];
11
12 if ($user_department === 'admin') {
13     // for admin side folders
14     if ($parent_folder_id) {
15         $query = "SELECT * FROM folders WHERE parent_folder = ?";
16         $stmt = $conn->prepare($query);
17         $stmt->bind_param("i", $parent_folder_id);
18         $stmt->execute();
19         $folders_result = $stmt->get_result();
20
21         $files_query = "SELECT f.*, fld.row_no, fld.column_no, fld.status as folder_status
22         FROM files f
23         JOIN folders fld ON f.folder_id = fld.folder_id
24         WHERE fld.folder_id = ? AND f.status = 'approved'";
25         $files_stmt = $conn->prepare($files_query);
26         $files_stmt->bind_param("i", $parent_folder_id);
27         $files_stmt->execute();
28         $files_result = $files_stmt->get_result();
29
30         $parent_query = "SELECT folder_name FROM folders WHERE folder_id = ?";
31         $parent_stmt = $conn->prepare($parent_query);
32         $parent_stmt->bind_param("i", $parent_folder_id);
33         $parent_stmt->execute();
34         $parent_result = $parent_stmt->get_result();
35         $parent_folder = $parent_result->fetch_assoc();
36     } else {
37         $query = "SELECT * FROM folders WHERE (parent_folder IS NULL OR parent_folder = 0)";
38         $stmt = $conn->prepare($query);
39         $stmt->execute();
40
41         $folders_result = $stmt->get_result();
42
43     }
44
45     $years_query = "SELECT DISTINCT year FROM folders ORDER BY year DESC";
46     $years_stmt = $conn->prepare($years_query);
47     $years_stmt->execute();
48     $years_result = $years_stmt->get_result();
49 } else {
50     // for user folders
51     if ($parent_folder_id) {
52         $parent_details_query = "SELECT row_no, column_no FROM folders WHERE folder_id = ?";
53         $parent_details_stmt = $conn->prepare($parent_details_query);
54         $parent_details_stmt->bind_param("i", $parent_folder_id);
55         $parent_details_stmt->execute();
56         $parent_details_result = $parent_details_stmt->get_result();
57         $parent_details = $parent_details_result->fetch_assoc();
58         // $parent_row = $parent_details['row_no'] ?? 0;
59         // $parent_col = $parent_details['column_no'] ?? 0;
60
61         function getTopMostParentDetails($conn, $folder_id)
62         {
63             while ($folder_id) {
64                 $query = "SELECT parent_folder, row_no, column_no FROM folders WHERE folder_id = ?";
65                 $stmt = $conn->prepare($query);
66                 $stmt->bind_param("i", $folder_id);
67                 $stmt->execute();
68                 $result = $stmt->get_result();
69                 $folder = $result->fetch_assoc();
70
71                 if (!$folder || !$folder['parent_folder']) {
72                     return [
73                         'row_no' => $folder['row_no'] ?? 0,
74                         'column_no' => $folder['column_no'] ?? 0
75                     ];
76                 }
77
78             }
79             // move to the next parent in the hierarchy
80             $folder_id = $folder['parent_folder'];
81             return ['row_no' => 0, 'column_no' => 0]; // default data / values
82         }
83
84         $parent_details = getTopMostParentDetails($conn, $parent_folder_id);
85         $parent_row = $parent_details['row_no'];
86         $parent_col = $parent_details['column_no'];
87
88         $query = "SELECT * FROM folders WHERE parent_folder = ? AND department = ?";
89         $stmt = $conn->prepare($query);
90         $stmt->bind_param("is", $parent_folder_id, $user_department);
91         $stmt->execute();
92         $folders_result = $stmt->get_result();
93
94         $files_query = "SELECT f.*, fld.row_no, fld.column_no, fld.status as folder_status
95         FROM files f
96         JOIN folders fld ON f.folder_id = fld.folder_id
97         WHERE fld.folder_id = ? AND f.status = 'approved'";
98         $files_stmt = $conn->prepare($files_query);
99         $files_stmt->bind_param("i", $parent_folder_id);
100        $files_stmt->execute();
101        $files_result = $files_stmt->get_result();
102
103        $parent_query = "SELECT folder_name FROM folders WHERE folder_id = ?";
104        $parent_stmt = $conn->prepare($parent_query);
105        $parent_stmt->bind_param("i", $parent_folder_id);
106        $parent_stmt->execute();
107        $parent_result = $parent_stmt->get_result();
108        $parent_folder = $parent_result->fetch_assoc();
109    } else {
110        $query = "SELECT * FROM folders WHERE (parent_folder IS NULL OR parent_folder = 0) AND department = ?";
111        $stmt = $conn->prepare($query);
112        $stmt->bind_param("s", $user_department);
113        $stmt->execute();
114        $folders_result = $stmt->get_result();

```



```

114     $folders_result = $stmt->get_result();
115
116     $files_result = false;
117 }
118
119     $years_query = "SELECT DISTINCT year FROM folders WHERE department = ? ORDER BY year DESC";
120     $years_stmt = $conn->prepare($years_query);
121     $years_stmt->bind_param("s", $user_department);
122     $years_stmt->execute();
123     $years_result = $years_stmt->get_result();
124 }
125
126
127 ?>
```

get_metadata.php

```

1 <?php
2 session_start();
3 include 'conn.php';
4
5 if (isset($_GET['file_id'])) {
6     $file_id = $_GET['file_id'];
7     $metadata_query = "
8         SELECT mt.meta_data_title, md.meta_data_description, md.title_id, mt.input_type
9             FROM metadata_description md
10            JOIN metadata_title mt ON md.title_id = mt.metadata_id
11           WHERE md.file_id = ?
12     ";
13     $stmt = mysqli_prepare($conn, $metadata_query);
14     mysqli_stmt_bind_param($stmt, 'i', $file_id);
15     mysqli_stmt_execute($stmt);
16     $result = mysqli_stmt_get_result($stmt);
17
18     while ($metadata = mysqli_fetch_assoc($result)) {
19         $input_type = isset($metadata['input_type']) && in_array($metadata['input_type'],
20             ['text', 'number', 'date', 'time', 'month', 'year', 'email', 'tel', 'datetime-local']);
21         ?> $metadata['input_type'] : 'text';
22
23         echo "<div class='mb-3'>
24             <label for='metadata_{$metadata['title_id']}'>{$metadata['meta_data_title']}</label>
25             <input type='{$input_type}' class='form-control' id='metadata_{$metadata['title_id']}' name='metadata_descriptions[{$metadata['title_id']}]' value='{$metadata['meta_data_description']}'>
26         </div>";
27     }
28 }
29 ?>
```

get.php

```

1 <?php
2
3 header('Content-Type: application/json');
4 require_once 'conn.php';
5 session_start();
6
7 // allowed tables ----
8 $tables = ['user', 'metadata', 'metadata_titles'];
9
10 // ensure the requested table exists ----
11 if (!isset($_GET['table'])) || !in_array($_GET['table'], $tables)) {
12     echo json_encode(['error' => 'Invalid or missing table parameter']);
13     exit;
14 }
15
16 // new endpoint for fetch_metadata_title functionality (removed fetch_metadata_title.php) ----
17 if (isset($_GET['fetch_metadata_title']) && isset($_GET['folder_id'])) {
18     if (!isset($_SESSION['user_id']) || !isset($_SESSION['department'])) {
19         echo json_encode(['success' => false, 'message' => 'User not authenticated or department not set']);
20         exit;
21     }
22
23     $folder_id = $_GET['folder_id'];
24     $query = "SELECT metadata_id, meta_data_title FROM metadata_title WHERE folder_id = ?";
25     $stmt = $conn->prepare($query);
26     $stmt->bind_param("i", $folder_id);
27     $stmt->execute();
28     $result = $stmt->get_result();
29
30     $metadata_titles = [];
31     while ($row = $result->fetch_assoc()) {
32         $metadata_titles[] = $row;
33     }
34
35     echo json_encode([
36         'success' => true,
37         'metadata_titles' => $metadata_titles
38     ]);
39 }
```



```

39     $stmt->close();
40     $conn->close();
41     exit;
42 }
43
44 if ($_GET['table'] === 'metadata_titles') {
45     if (!isset($_GET['folder_id'])) {
46         echo json_encode(['error' => 'Folder ID not provided']);
47         exit;
48     }
49
50     $folder_id = intval($_GET['folder_id']);
51     $query = "SELECT * FROM metadata_title WHERE folder_id = ?";
52     $stmt = $conn->prepare($query);
53     $stmt->bind_param("i", $folder_id);
54     $stmt->execute();
55     $result = $stmt->get_result();
56
57     $metadata_titles = [];
58     while ($row = $result->fetch_assoc()) {
59         $metadata_titles[] = $row;
60     }
61
62     echo json_encode($metadata_titles);
63     $stmt->close();
64     $conn->close();
65     exit;
66 }
67
68 // check if metadata is requested ----
69 if ($_GET['table'] === 'metadata') {
70     if (!isset($_SESSION['id'])) {
71         echo json_encode(['success' => false, 'error' => 'Unauthorized']);
72         exit;
73     }
74 }
75
76
77 if (!isset($_GET['file_id'])) {
78     echo json_encode(['success' => false, 'error' => 'File ID not provided']);
79     exit;
80 }
81
82 $file_id = intval($_GET['file_id']);
83 $folder_id = isset($_GET['folder_id']) ? intval($_GET['folder_id']) : null;
84
85 $metadataTitles = [];
86 $metadataDescriptions = [];
87 $metadataInputTypes = [];
88
89 if ($folder_id) {
90     $query = "SELECT meta_data_title, input_type FROM metadata_title WHERE folder_id = ?";
91     $stmt = $conn->prepare($query);
92     $stmt->bind_param("i", $folder_id);
93     $stmt->execute();
94     $result = $stmt->get_result();
95
96     while ($row = $result->fetch_assoc()) {
97         $metadataTitles[] = $row['meta_data_title'];
98         $metadataInputTypes[] = $row['input_type'];
99     }
100
101 $query = "SELECT meta_data_description FROM metadata_description WHERE file_id = ?";
102 $stmt = $conn->prepare($query);
103 $stmt->bind_param("i", $file_id);
104 $stmt->execute();
105 $result = $stmt->get_result();
106
107 while ($row = $result->fetch_assoc()) {
108     $metadataDescriptions[] = $row['meta_data_description'];
109 }
110
111 echo json_encode([
112     'success' => true,
113     'metadataTitles' => $metadataTitles,
114     'metadataDescriptions' => $metadataDescriptions,
115     'metadataInputTypes' => $metadataInputTypes
116 ]);
117
118 $conn->close();
119 exit;
120 }
121
122 // handle requests for user table ----
123 $tableName = $_GET['table'];
124
125 // Fetching a single user
126 if (isset($_GET['id'])) {
127     $id = $conn->real_escape_string($_GET['id']);
128     $query = "SELECT id, full_name, username, address, department, contact_info FROM $tableName WHERE id = '$id'";
129     $result = $conn->query($query);
130
131     if ($result && $result->num_rows > 0) {
132         echo json_encode($result->fetch_assoc());
133     } else {
134         echo json_encode(['error' => 'User not found']);
135     }
136 }
137
138 // get user full name ----
139 elseif (isset($_GET['user_id'])) {
140     $user_id = intval($_GET['user_id']);
141     $query = "SELECT full_name FROM $tableName WHERE id = ?";
142     $stmt = $conn->prepare($query);
143     $stmt->bind_param("i", $user_id);
144     $stmt->execute();
145     $result = $stmt->get_result();

```



```

145     if ($user = $result->fetch_assoc()) {
146         echo json_encode(['success' => true, 'full_name' => $user['full_name']]);
147     } else {
148         echo json_encode(['success' => false, 'error' => 'User not found']);
149     }
150 }
151 // get all users except the admin ----
152 else {
153     $sql = "SELECT u.*, d.folder_color
154         FROM $tableName u
155         LEFT JOIN department d ON u.department = d.department_name
156         WHERE u.role != 'Admin'";
157
158     $result = $conn->query($sql);
159
160     $data = [];
161     if ($result && $result->num_rows > 0) {
162         while ($row = $result->fetch_assoc()) {
163             $data[] = $row;
164         }
165     }
166
167     echo json_encode($data);
168 }
169
170 $conn->close();
171 ?>
```

login.php

```

1  <?php
2  session_start();
3  header('Content-Type: application/json');
4
5  include 'conn.php';
6
7  if ($_SERVER['REQUEST_METHOD'] === 'POST') {
8      $username = $conn->real_escape_string($_POST['username']);
9      $password = $_POST['password'];
10
11     $sql = "SELECT * FROM user WHERE username = '$username'";
12     $result = $conn->query($sql);
13
14     if ($result->num_rows > 0) {
15         $user = $result->fetch_assoc();
16
17         if ($user['status'] === 'disabled') {
18             echo json_encode([
19                 'success' => false,
20                 'message' => 'Your account has been disabled. Please contact the administrator.'
21             ]);
22         } elseif (password_verify($password, $user['password'])) {
23             $SESSION['user_id'] = $user['id'];
24             $SESSION['username'] = $user['username'];
25             $SESSION['department'] = $user['department'];
26             $SESSION['address'] = $user['address'];
27             $SESSION['contact_info'] = $user['contact_info'];
28             $SESSION['role'] = $user['role'];
29             $SESSION['full_name'] = $user['full_name'];
30
31             $redirect = $user['role'] === 'Admin'
32                 ? 'pages/admin/dashboard.php'
33                 : 'pages/personnel/home.php';
34
35             echo json_encode([
36                 'success' => true,
37                 'message' => 'Login successful!',
38                 'redirect' => $redirect
39             ]);
40         } else {
41             echo json_encode([
42                 'success' => false,
43                 'message' => 'Invalid password';
44             ]);
45         }
46     } else {
47         echo json_encode([
48                 'success' => false,
49                 'message' => 'Username not found!';
50             ]);
51     }
52 }
53
54 $conn->close();
55 ?>
```

logout.php

```

1  <?php
2  session_start();
3
4  session_unset();
5
6  session_destroy();
7
8  header("Location: ../../");
9  exit;
10 ?>
```



reupload_file.php

```

1  <?php
2  session_start();
3  include 'conn.php';
4
5  // error reporting || debug ---- (apr 2, 2025)
6  error_reporting(E_ALL & ~E_NOTICE);
7
8  if (!isset($_SESSION['user_id'])) {
9      echo json_encode(['success' => false, 'message' => 'Unauthorized access']);
10     exit;
11 }
12
13 if ($_SERVER['REQUEST_METHOD'] !== 'POST' || !isset($_POST['reupload_file'])) {
14     echo json_encode(['success' => false, 'message' => 'Invalid request']);
15     exit;
16 }
17
18 $file_id = $_POST['file_id'] ?? null;
19 $file_name = $_POST['file_name'] ?? null;
20 $file_owner = $_POST['file_owner'] ?? null;
21
22 if (!$file_id || !$file_name || !$file_owner) {
23     echo json_encode(['success' => false, 'message' => 'Required fields are missing']);
24     exit;
25 }
26
27 $current_file_query = "SELECT file_name, folder_id FROM files WHERE file_id = ?";
28 $stmt = $conn->prepare($current_file_query);
29 $stmt->bind_param("i", $file_id);
30 $stmt->execute();
31 $result = $stmt->get_result();
32 $current_file = $result->fetch_assoc();
33
34 if (!$current_file) {
35     echo json_encode(['success' => false, 'message' => 'File not found']);
36     exit;
37 }
38
39 $new_filename = $current_file['file_name'];
40 $upload_success = true;
41
42 if (isset($_FILES['new_file']) && $_FILES['new_file']['error'] === UPLOAD_ERR_OK) {
43     $file = $_FILES['new_file'];
44
45     $file_type = mime_content_type($file['tmp_name']);
46     if ($file_type !== 'application/pdf') {
47         echo json_encode(['success' => false, 'message' => 'Only PDF files are allowed']);
48         exit;
49     }
50
51     $file_ext = pathinfo($file['name'], PATHINFO_EXTENSION);
52     $new_filename = uniqid() . '.' . $file_ext;
53     $upload_path = '../../../../../uploads/' . $new_filename;
54
55     if (!move_uploaded_file($file['tmp_name'], $upload_path)) {
56         $upload_success = false;
57     } else {
58         $old_file_path = '../../../../../uploads/' . $current_file['file_name'];
59         if (file_exists($old_file_path)) {
60             unlink($old_file_path);
61         }
62     }
63 }
64
65 if (!$upload_success) {
66     echo json_encode(['success' => false, 'message' => 'Failed to upload new file']);
67     exit;
68 }
69
70 $update_query = "UPDATE files SET
71     original_name = ?,
72     file_name = ?,
73     author = ?,
74     upload_date = NOW(),
75     status = 'pending',
76     remarks = ?,
77     WHERE file_id = ?";
78
79 $stmt = $conn->prepare($update_query);
80 $stmt->bind_param(
81     "sssl",
82     $file_name,
83     $new_filename,
84     $file_owner,
85     $file_id
86 );
87
88 if ($stmt->execute()) {
89     if (isset($_POST['metadata_descriptions']) && is_array($_POST['metadata_descriptions'])) {
90         $metadata_descriptions = $_POST['metadata_descriptions'];
91
92         foreach ($metadata_descriptions as $metadata_id => $description) {
93             if (is_array($description)) {
94                 $description = implode(' ', $description);
95             }
96
97             $update_metadata_query = "UPDATE metadata_description SET meta_data_description = ?
98                                     WHERE file_id = ? AND title_id = ?";
99             $meta_stmt = $conn->prepare($update_metadata_query);
100            $meta_stmt->bind_param('sii', $description, $file_id, $metadata_id);
101
102            if (!$meta_stmt->execute()) {
103                error_log("Failed to update metadata for file_id: $file_id, title_id: $metadata_id");
104            }
105        }
106    }
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
625
626
627
627
628
629
629
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
150
```



```
106 |     }
107 |
108 |
109 // clear output buffer | send JSON res.
110 ob_clean();
111 echo json_encode(['success' => true, 'message' => 'File reuploaded successfully and sent for approval']);
112 } else {
113     ob_clean();
114     echo json_encode(['success' => false, 'message' => 'Failed to update file record']);
115 }
116
117 $stmt->close();
118 $conn->close();
119 ?>
```

search.php

```
1 <?php
2 require_once 'conn.php';
3 session_start();
4
5 header('Content-Type: application/json');
6
7 $userDepartment = $_SESSION['department'];
8 $userId = $_SESSION['user_id'];
9 $yearFilter = isset($_GET['year']) && $_GET['year'] != 'all' ? $_GET['year'] : null;
10
11 // get department colors
12 $departmentColors = [];
13 $colorQuery = "SELECT department_name, CONCAT('#', folder_color) as color FROM department";
14 $colorResult = $conn->query($colorQuery);
15 while ($row = $colorResult->fetch_assoc()) {
16     $departmentColors[$row['department_name']] = $row['color'];
17 }
18
19 if ($userDepartment === 'admin') {
20     try {
21         $searchTerm = isset($_GET['term']) ? '%' . $_GET['term'] . '%' : '';
22
23         $response = [
24             'folders' => [],
25             'files' => [],
26             'departmentColors' => $departmentColors
27         ];
28
29         $folderQuery = "SELECT * FROM folders WHERE folder_name LIKE ?";
30         $params = array($searchTerm);
31         $types = "s";
32
33         if ($yearFilter) {
34             $folderQuery .= " AND year = ?";
35             $params[] = $yearFilter;
36             $types .= "s";
37         }
38     }
39 }
```

```
38
39     $stmt = $conn->prepare($folderQuery);
40     $stmt->bind_param($types, ...$params);
41     $stmt->execute();
42     $result = $stmt->get_result();
43     while ($row = $result->fetch_assoc()) {
44         $FolderPath = getFolderPath($conn, $row['folder_id']);
45         $row['FolderPath'] = $FolderPath;
46         $response['folders'][] = $row;
47     }
48
49     $fileQuery = "SELECT f.* , fo.row_no, fo.column_no FROM files f
50     INNER JOIN folders fo ON f.folder_id = fo.folder_id
51     WHERE (f.original_name LIKE ? OR f.author LIKE ?)
52     AND f.status = 'approved'";
53
54     $params = array($searchTerm, $searchTerm);
55     $types = "$s";
56
57     if ($yearfilter) {
58         $fileQuery .= " AND fo.year = ?";
59         $params[] = $yearFilter;
60         $types .= "s";
61     }
62
63     $stmt = $conn->prepare($fileQuery);
64     $stmt->bind_param($types, ...$params);
65     $stmt->execute();
66     $result = $stmt->get_result();
67     while ($row = $result->fetch_assoc()) {
68         $FolderPath = getFolderPath($conn, $row['folder_id']);
69         $row['FolderPath'] = $FolderPath;
70         $row['is_owner'] = ($row['user_id'] == $userId);
71         $response['files'][] = $row;
72     }
73 }
```



```

73
74     echo json_encode([
75         'success' => true,
76         'data' => $response,
77         'user_id' => $userId
78     ]);
79
80 } catch (Exception $e) {
81     echo json_encode([
82         'success' => false,
83         'error' => $e->getMessage()
84     ]);
85 }
86 } else {
87     try {
88         $searchTerm = isset($_GET['term']) ? '%' . $_GET['term'] . '%' : '';
89         $yearFilter = isset($_GET['year']) && $_GET['year'] !== 'all' ? $_GET['year'] : null;
90         $userDepartment = $_SESSION['department'];
91
92         $response = [
93             'folders' => [],
94             'files' => [],
95             'departmentColors' => $departmentColors
96         ];
97
98         // search folders with department and optional year filter
99         $folderQuery = "SELECT * FROM folders WHERE folder_name LIKE ? AND department = ?";
100        $params = array($searchTerm, $userDepartment);
101        $types = "ss";
102
103        if ($yearFilter) {
104            $folderQuery .= " AND year = ?";
105            $params[] = $yearFilter;
106            $types .= "s";
107        }
108
109        $stmt = $conn->prepare($folderQuery);
110        $stmt->bind_param($types, ...$params);
111        $stmt->execute();
112        $result = $stmt->get_result();
113        while ($row = $result->fetch_assoc()) {
114            $FolderPath = getFolderPath($conn, $row['folder_id']);
115            $row['folder_path'] = $FolderPath;
116            $response['folders'][] = $row;
117        }
118
119        // search files with department and optional year filter from parent folder
120        $fileQuery = "SELECT f.*, fo.row_no, fo.column_no FROM files f
121            INNER JOIN folders fo ON f.folder_id = fo.folder_id
122            WHERE (f.original_name LIKE ? OR f.author LIKE ?)
123            AND fo.department = ?
124            AND fo.status = 'approved'";
125
126        $params = array($searchTerm, $searchTerm, $userDepartment);
127        $types = "sss";
128
129        if ($yearFilter) {
130            $fileQuery .= " AND fo.year = ?";
131            $params[] = $yearFilter;
132            $types .= "s";
133        }
134
135        $stmt = $conn->prepare($fileQuery);
136        $stmt->bind_param($types, ...$params);
137        $stmt->execute();
138        $result = $stmt->get_result();
139        while ($row = $result->fetch_assoc()) {
140            $FolderPath = getFolderPath($conn, $row['folder_id']);
141            $row['folder_path'] = $FolderPath;
142            $row['is_owner'] = ($row['user_id'] == $userId);
143            $response['files'][] = $row;
144        }
145
146     echo json_encode([
147         'success' => true,
148         'data' => $response,
149         'user_id' => $userId
150     ]);
151
152 } catch (Exception $e) {
153     header('Content-Type: application/json');
154     echo json_encode([
155         'success' => false,
156         'error' => $e->getMessage()
157     ]);
158 }
159
160
161
162 function getFolderPath($conn, $folder_id)
163 {
164     $path = array();
165     $current_id = $folder_id;
166
167     while ($current_id) {
168         $query = "SELECT folder_id, folder_name, parent_folder FROM folders WHERE folder_id = ?";
169         $stmt = $conn->prepare($query);
170         $stmt->bind_param("i", $current_id);
171         $stmt->execute();
172         $result = $stmt->get_result();
173
174         if ($folder = $result->fetch_assoc()) {
175             array_unshift($path, $folder['folder_name']);
176             $current_id = $folder['parent_folder'];
177         } else {
178             break;
179         }
180     }
181
182 }

```



```

182     if (count($path) === 1 && $current_id === null) {
183         return '';
184     }
185     return implode(' / ', $path);
186 }
187
188 $conn->close();
189 ?>

```

update_folder.php

```

1 <?php
2 error_reporting(E_ALL);
3 ini_set('display_errors', 1);
4
5 include 'conn.php';
6
7 header('Content-Type: application/json');
8
9 try {
10     $data = json_decode(file_get_contents('php://input'), true);
11
12     if (!isset($data['folder_name'], $data['folder_id'])) {
13         throw new Exception("Way data doy!");
14     }
15
16     // function to also update years for subfolders (occurs when admin updates 'year' for parent folders)
17     function updateSubfoldersYear($conn, $parent_id, $year)
18     {
19         $query = "UPDATE folders SET year = ? WHERE parent_folder = ?";
20         $stmt = $conn->prepare($query);
21         $stmt->bind_param("si", $year, $parent_id);
22         $stmt->execute();
23         $stmt->close();
24
25         $query2 = "SELECT folder_id FROM folders WHERE parent_folder = ?";
26         $stmt2 = $conn->prepare($query2);
27         $stmt2->bind_param("i", $parent_id);
28         $stmt2->execute();
29         $result = $stmt2->get_result();
30
31         while ($row = $result->fetch_assoc()) {
32             updateSubfoldersYear($conn, $row['folder_id'], $year);
33         }
34         $stmt2->close();
35     }
36

```

```

36
37     $folder_name = $conn->real_escape_string($data['folder_name']);
38     $folder_id = (int) $data['folder_id'];
39     $year = isset($data['year']) ? $conn->real_escape_string($data['year']) : null;
40
41     if ($year !== null) {
42         $query = "UPDATE folders SET folder_name = ?, year = ? WHERE folder_id = ?";
43         $stmt = $conn->prepare($query);
44         $stmt->bind_param("ssi", $folder_name, $year, $folder_id);
45         $stmt->execute();
46         $stmt->close();
47
48         updateSubfoldersYear($conn, $folder_id, $year);
49     } else {
50         $query = "UPDATE folders SET folder_name = ? WHERE folder_id = ?";
51         $stmt = $conn->prepare($query);
52         $stmt->bind_param("si", $folder_name, $folder_id);
53         $stmt->execute();
54         $stmt->close();
55     }
56
57     $response = [
58         'success' => true,
59         'message' => 'Folder updated successfully'
60     ];
61 } catch (Exception $e) {
62     $response = [
63         'success' => false,
64         'message' => $e->getMessage()
65     ];
66 }
67 echo json_encode($response);
68 $conn->close();
69 ?>

```



update.php

```

1 <?php
2 require_once 'conn.php';
3 session_start();
4
5 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
6     $data = json_decode(file_get_contents('php://input'), true);
7
8     if (isset($data['action'])) {
9         if ($data['action'] === 'update_file') {
10             // file update (user modal edit file button)
11             $fileId = $data['file_id'];
12             $originalName = $data['original_name'];
13             $author = $data['author'];
14
15             $stmt = $conn->prepare("UPDATE files SET original_name = ?, author = ? WHERE file_id = ?");
16             $stmt->bind_param('si', $originalName, $author, $fileId);
17
18             if ($stmt->execute()) {
19                 echo json_encode(['success' => true]);
20             } else {
21                 echo json_encode(['success' => false, 'error' => $stmt->error]);
22             }
23
24             $stmt->close();
25         } elseif ($data['action'] === 'update_metadata') {
26             // metadata update (user modal edit metadata button)
27             $fileId = intval($data['fileid']);
28             $userId = $_SESSION['user_id'];
29             $metadataUpdates = $data['metadata'];
30
31             $response = ['success' => true, 'errors' => []];
32
33             foreach ($metadataUpdates as $update) {
34                 $title = $update['title'];
35                 $description = $update['data'];
36                 $originalData = $update['originalData'];
37
38                 // get the metadata_id from metadata_title
39                 $stmt = $conn->prepare("SELECT metadata_id FROM metadata_title WHERE meta_data_title = ?");
40                 $stmt->bind_param('s', $title);
41                 $stmt->execute();
42                 $result = $stmt->get_result();
43                 $titleRow = $result->fetch_assoc();
44
45                 if (!empty($titleRow)) {
46                     $response['success'] = false;
47                     $response['errors'][] = "Metadata title not found: $title";
48                     continue;
49                 }
50
51                 $titleId = $titleRow['metadata_id'];
52
53                 // check if a metadata description exists for this file and title id
54                 $stmt = $conn->prepare("SELECT COUNT(*) as count FROM metadata_description WHERE file_id = ? AND title_id = ?");
55                 $stmt->bind_param('ii', $fileId, $titleId);
56                 $stmt->execute();
57                 $result = $stmt->get_result();
58                 $row = $result->fetch_assoc();
59
60                 if ($row['count'] > 0) {
61                     // if exists, update it
62                     $stmt = $conn->prepare("UPDATE metadata_description SET meta_data_description = ? WHERE file_id = ? AND title_id = ?");
63                     $stmt->bind_param('ssi', $description, $fileId, $titleId);
64                 } else {
65                     // if not exists, insert new metadata description (if ever nga ang admin mag add nasad ug metadata title, so empty and desc
66                     $stmt = $conn->prepare("INSERT INTO metadata_description (file_id, meta_data_description, user_id, title_id) VALUES (?, ?, ?, ?");
67                     $stmt->bind_param('isii', $fileId, $description, $userId, $titleId);
68                 }
69
70                 if (!$stmt->execute()) {
71                     $response['success'] = false;
72                     $response['errors'][] = $stmt->error;
73                 }
74             }
75
76             echo json_encode($response);
77         } elseif ($data['action'] === 'update_metadata_title') {
78             // handle metadata title update (admin modal edit metadata button)
79             if (!isset($data['metadata_id']), !isset($data['meta_data_title'])) {
80                 echo json_encode(['success' => false, 'message' => 'Missing required fields']);
81                 return;
82             }
83
84             $metadataId = (int) $data['metadata_id'];
85             $metaDataTitle = $conn->real_escape_string($data['meta_data_title']);
86
87             $query = "UPDATE metadata_title SET meta_data_title = ? WHERE metadata_id = ?";
88             $stmt = $conn->prepare($query);
89             $stmt->bind_param('si', $metaDataTitle, $metadataId);
90             $stmt->execute();
91             $stmt->close();
92
93             echo json_encode(['success' => true]);
94         } elseif ($data['action'] === 'update_user') {
95             // handle user update (admin modal edit user button)
96             if (!isset($_SESSION['user_id']) || !$_SESSION['role'] === 'Admin') {
97                 echo json_encode(['error' => 'Unauthorized']);
98                 return;
99             }
100
101             $id = $conn->real_escape_string($data['id']);
102             $full_name = $conn->real_escape_string($data['full_name']);
103             $username = $conn->real_escape_string($data['username']);
104             $address = $conn->real_escape_string($data['address']);
105             $contact = $conn->real_escape_string($data['contact']);
106
107             $query = "UPDATE user SET
108                 full_name = '$full_name',
109                 username = '$username',
110                 address = '$address',
111                 contact_info = '$contact',
112                 WHERE id = '$id'";
113

```



```

113
114     if ($conn->query($query)) {
115         echo json_encode(['success' => true]);
116     } else {
117         echo json_encode(['error' => 'Update failed: ' . $conn->error]);
118     }
119 } elseif ($data['action'] === 'update_status') {
120     // handle status update (admin accounts list status toggle button)
121     if (!isset($_SESSION['user_id']) || $_SESSION['role'] !== 'Admin') {
122         echo json_encode(['error' => 'Unauthorized']);
123         return;
124     }
125
126     $id = $conn->real_escape_string($data['id']);
127     $status = $conn->real_escape_string($data['status']);
128
129     if (!in_array($status, ['enabled', 'disabled'])) {
130         echo json_encode(['error' => 'Invalid status']);
131         return;
132     }
133
134     $query = "UPDATE user SET status = '$status' WHERE id = '$id'";
135
136     if ($conn->query($query)) {
137         echo json_encode(['success' => true]);
138     } else {
139         echo json_encode(['error' => 'Status update failed: ' . $conn->error]);
140     }
141 } elseif ($data['action'] === 'update_folder_position') {
142     // folder position update (users folders - row/column)
143     if (!isset($data['folder_id']) || !isset($data['field']) || !isset($data['value'])) {
144         echo json_encode(['success' => false, 'message' => 'Missing required parameters']);
145         return;
146     }
147
148     $folder_id = intval($data['folder_id']);
149     $field = ($data['field'] === 'row') ? 'row_no' : 'column_no';
150     $value = intval($data['value']);
151
152     if ($value < 0) {
153         echo json_encode(['success' => false, 'message' => 'Invalid value']);
154         return;
155     }
156
157     try {
158         $stmt = $conn->prepare("UPDATE folders SET $field = ? WHERE folder_id = ?");
159         $stmt->bind_param('ii', $value, $folder_id);
160
161         if ($stmt->execute()) {
162             echo json_encode(['success' => true]);
163         } else {
164             echo json_encode(['success' => false, 'message' => 'Update failed']);
165         }
166
167         $stmt->close();
168     } catch (Exception $e) {
169         echo json_encode(['success' => false, 'message' => 'Database error']);
170     }
171 } elseif ($data['action'] === 'update_password') {
172     // user password update (admin accounts - change password button)
173     if (!isset($_SESSION['user_id']) || $_SESSION['role'] !== 'Admin') {
174         echo json_encode(['error' => 'Unauthorized']);
175         return;
176     }
177
178     if (!isset($data['id']), $data['password'])) {
179         echo json_encode(['error' => 'Missing required parameters']);
180         return;
181     }
182
183     $id = $conn->real_escape_string($data['id']);
184     $password = password_hash($conn->real_escape_string($data['password']), PASSWORD_BCRYPT);
185
186     $query = "UPDATE user SET password = ? WHERE id = ?";
187     $stmt = $conn->prepare($query);
188     $stmt->bind_param("si", $password, $id);
189
190     if ($stmt->execute()) {
191         echo json_encode(['success' => true]);
192     } else {
193         echo json_encode(['error' => 'Password update failed: ' . $stmt->error]);
194     }
195
196     $stmt->close();
197 } elseif ($data['action'] === 'update_file_status') {
198     $file_id = $data['file_id'];
199     $status = $data['status'];
200     $remarks = $data['remarks'] ?? '';
201
202     $query = "UPDATE files SET status = ?, remarks = ? WHERE file_name = ?";
203     $stmt = $conn->prepare($query);
204     $stmt->bind_param("sss", $status, $remarks, $file_id);
205
206     if ($stmt->execute()) {
207         echo json_encode(['success' => true]);
208     } else {
209         echo json_encode(['success' => false, 'error' => $stmt->error]);
210     }
211
212     $stmt->close();
213 } elseif ($data['action'] === 'update_profile') {
214     $userId = $_SESSION['user_id'];
215     $fullName = $data['full_name'];
216     $address = $data['address'];
217     $contactInfo = $data['contact_info'];
218     $password = $data['password'] ?? '';
219

```



```

219
220     try {
221         if (!empty($password)) {
222             $passwordHash = password_hash($password, PASSWORD_BCRYPT);
223             $sql = "UPDATE user SET full_name = ?, contact_info = ?, address = ?, password = ? WHERE id = ?";
224             $stmt = $conn->prepare($sql);
225             $stmt->bind_param("ssssi", $fullName, $contactInfo, $address, $passwordHash, $userId);
226         } else {
227             $sql = "UPDATE user SET full_name = ?, contact_info = ?, address = ? WHERE id = ?";
228             $stmt = $conn->prepare($sql);
229             $stmt->bind_param("sssi", $fullName, $contactInfo, $address, $userId);
230         }
231
232         if ($stmt->execute()) {
233             if ($stmt->affected_rows > 0) {
234                 $_SESSION['full_name'] = $fullName;
235                 $_SESSION['address'] = $address;
236                 $_SESSION['contact_info'] = $contactInfo;
237
238                 echo json_encode(['success' => true, 'message' => 'Profile updated successfully']);
239             } else {
240                 echo json_encode(['success' => true, 'message' => 'No changes detected']);
241             }
242         } else {
243             echo json_encode(['success' => false, 'error' => 'Database error: ' . $stmt->error]);
244         }
245
246         $stmt->close();
247     } catch (Exception $e) {
248         echo json_encode(['success' => false, 'error' => $e->getMessage()]);
249     }
250 } elseif ($data['action'] === 'update_department') {
251     $department_id = $data['department_id'];
252     $department_name = trim($data['department_name']);
253     $folder_color = trim($data['folder_color']);
254
255     $folder_color = str_replace('#', '', $folder_color);
256
257     if (empty($department_id) || empty($department_name) || empty($folder_color)) {
258         echo json_encode(['success' => false, 'message' => 'All fields are required']);
259         exit;
260     }
261
262     $check_query = "SELECT id FROM department WHERE department_name = ? AND id != ?";
263     $check_stmt = $conn->prepare($check_query);
264     $check_stmt->bind_param("si", $department_name, $department_id);
265     $check_stmt->execute();
266     $result = $check_stmt->get_result();
267
268     if ($result->num_rows > 0) {
269         echo json_encode(['success' => false, 'message' => 'Department name already exists']);
270         exit;
271     }
272
273     $query = "UPDATE department SET department_name = ?, folder_color = ? WHERE id = ?";
274     $stmt = $conn->prepare($query);
275     $stmt->bind_param("ssi", $department_name, $folder_color, $department_id);
276
277     if ($stmt->execute()) {
278         echo json_encode(['success' => true, 'message' => 'Department updated successfully']);
279     } else {
280         echo json_encode(['success' => false, 'message' => 'Error updating department']);
281     }
282
283     $stmt->close();
284 } elseif ($data['action'] === 'reupload_file') {
285     // Handle file reupload (change status to pending and clear remarks)
286     $file_id = $data['file_id'];
287
288     // Update the file status to 'pending' and clear remarks
289     $query = "UPDATE files SET status = 'pending', remarks = NULL WHERE file_id = ?";
290     $stmt = $conn->prepare($query);
291     $stmt->bind_param("s", $file_id);
292
293     if ($stmt->execute()) {
294         echo json_encode(['success' => true]);
295     } else {
296         echo json_encode(['success' => false, 'error' => $stmt->error]);
297     }
298
299     $stmt->close();
300 } else {
301     echo json_encode(['success' => false, 'error' => 'Invalid action']);
302 }
303 } else {
304     echo json_encode(['success' => false, 'error' => 'Action not specified']);
305 }
306
307 $conn->close();
308 }
309 ?>
```

upload_file.php

```
1<?php
2session_start();
3require_once 'conn.php';
4
5error_reporting(E_ALL);
6ini_set('display_errors', 0);
7ini_set('log_errors', 1);
8ini_set('error_log', 'upload_errors.log');
9
10header('Content-Type: application/json');
11
12try {
13    if (!isset($_SESSION['user_id']) || !isset($_SESSION['department'])) {
14        throw new Exception('User not authenticated or department not set');
15    }
16
17    if (!isset($_FILES['file'])) {
18        throw new Exception('No file uploaded');
19    }
20
21    if (!isset($_POST['author']) || empty(trim($_POST['author']))) {
22        throw new Exception('Author is required');
23    }
24
25    if (!isset($_POST['file_name']) || empty(trim($_POST['file_name']))) {
26        throw new Exception('File Name is required');
27    }
28
29    $file = $_FILES['file'];
30    $folder_id = $_POST['folder_id'];
31    $author = trim($_POST['author']);
32    $file_name = trim($_POST['file_name']);
33    $user_id = $_SESSION['user_id'];
34    $department = $_SESSION['department'];
35    $status = $_POST['status'] ?? 'pending';
36    $handle_duplicate = isset($_POST['handle_duplicate']) && $_POST['handle_duplicate'] === '1';
37
38    if ($file['error'] !== UPLOAD_ERR_OK) {
39        throw new Exception("File upload error: " . $file['error']);
40    }
41
42    $metadata_json = $_POST['metadata_json'] ?? null;
43    $metadata = ($metadata_json) ? json_decode($metadata_json, true) : [];
44
45    if (json_last_error() !== JSON_ERROR_NONE) {
46        throw new Exception('Invalid metadata format');
47    }
48
49    $metadata_ids = array_keys($metadata);
50    $valid_ids = [];
51
52    if (!empty($metadata_ids)) {
53        $placeholders = str_repeat('?', count($metadata_ids) - 1) . '?';
54
55        $validation_query = "SELECT metadata_id FROM metadata_title
56        WHERE metadata_id IN ($placeholders)
57        AND folder_id = ?";
58
59        $validation_stmt = $conn->prepare($validation_query);
60
61        $types = str_repeat('i', count($metadata_ids)) . 'i';
62        $args = array_merge($types, $metadata_ids, [$folder_id]);
63
64        call_user_func_array([$validation_stmt, 'bind_param'], $args);
65        $validation_stmt->execute();
66        $result = $validation_stmt->get_result();
67
68        while ($row = $result->fetch_assoc()) {
69            $valid_ids[] = $row['metadata_id'];
70        }
71        $validation_stmt->close();
72
73        $invalid_ids = array_diff($metadata_ids, $valid_ids);
74        if (!empty($invalid_ids)) {
75            throw new Exception("Invalid metadata IDs: " . implode(', ', $invalid_ids));
76        }
77    }
78
79    $conn->begin_transaction();
80
81    try {
82        date_default_timezone_set('Asia/Manila');
83        $upload_dir = './uploads/';
84        if (!file_exists($upload_dir)) {
85            mkdir($upload_dir, 0777, true);
86        }
87
88        if ($handle_duplicate) {
89            $check_query = "SELECT original_name FROM files WHERE folder_id = ? AND original_name LIKE ?";
90            $check_stmt = $conn->prepare($check_query);
91            $search_pattern = $file_name . '%';
92            $check_stmt->bind_param('is', $folder_id, $search_pattern);
93            $check_stmt->execute();
94            $check_result = $check_stmt->get_result();
95
96            $existing_names = [];
97            while ($row = $check_result->fetch_assoc()) {
98                $existing_names[] = $row['original_name'];
99            }
100
101            $new_name = $file_name;
102            $counter = 2;
103
104            while (in_array($new_name, $existing_names)) {
105                $new_name = $file_name . ' ' . $counter;
106                $counter++;
107            }
108
109            $file['name'] = $new_name;
110            $file['original_name'] = $file_name;
111            $file['status'] = $status;
112            $file['author'] = $author;
113            $file['department'] = $department;
114            $file['meta'] = $metadata;
115            $file['meta_id'] = $valid_ids[0];
116
117            $insert_query = "INSERT INTO files (file_name, original_name, folder_id, status, author, department, meta, meta_id) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
118            $insert_stmt = $conn->prepare($insert_query);
119            $insert_stmt->bind_param('ssssssss', $file['name'], $file['original_name'], $file['folder_id'], $file['status'], $file['author'], $file['department'], $file['meta'], $file['meta_id']);
120            $insert_stmt->execute();
121
122            $file['id'] = $conn->insert_id;
123
124            $update_query = "UPDATE metadata_title SET file_id = ? WHERE metadata_id = ?";
125            $update_stmt = $conn->prepare($update_query);
126            $update_stmt->bind_param('ii', $file['id'], $valid_ids[0]);
127            $update_stmt->execute();
128
129            $file['message'] = 'File uploaded successfully';
130            $file['status'] = 'success';
131
132            echo json_encode($file);
133        }
134    } catch (Exception $e) {
135        echo json_encode(['error' => $e->getMessage()]);
136    }
137}
```



```

108     $file_name = $new_name;
109 }
110
111 $file_extension = pathinfo($file['name'], PATHINFO_EXTENSION);
112 $unique_filename = uniqid() . '-' . time() . '-' . $file_extension;
113 $upload_path = $upload_dir . $unique_filename;
114
115 if (!move_uploaded_file($file['tmp_name'], $upload_path)) {
116     throw new Exception('Failed to move uploaded file');
117 }
118
119 $query = "INSERT INTO files (file_name, original_name, upload_date, user_id, folder_id, author, department, status)
120           VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
121 $stmt = $conn->prepare($query);
122 $current_date = date('Y-m-d H:i:s');
123
124 $stmt->bind_param(
125     "ssssss",
126     $unique_filename,
127     $file_name,
128     $current_date,
129     $user_id,
130     $folder_id,
131     $author,
132     $department,
133     $status
134 );
135
136 if (!$stmt->execute()) {
137     throw new Exception('Failed to insert file record: ' . $stmt->error);
138 }
139
140 $file_id = $stmt->insert_id;
141 $stmt->close();
142
143

```

```

143     if (!empty($valid_ids)) {
144         foreach ($valid_ids as $meta_id) {
145             if (isset($metadata[$meta_id]) && !empty(trim($metadata[$meta_id]))) {
146                 $meta_query = "INSERT INTO metadata_description (file_id, title_id, meta_data_description, user_id)
147                               VALUES (?, ?, ?, ?)";
148                 $meta_stmt = $conn->prepare($meta_query);
149                 $meta_description = trim($metadata[$meta_id]);
150                 $meta_stmt->bind_param("isi", $file_id, $meta_id, $meta_description, $user_id);
151
152                 if (!$meta_stmt->execute()) {
153                     throw new Exception('Failed to insert metadata: ' . $meta_stmt->error);
154                 }
155                 $meta_stmt->close();
156             }
157         }
158     }
159
160     $conn->commit();
161
162     echo json_encode([
163         'success' => true,
164         'message' => 'File uploaded successfully',
165         'file_name' => $unique_filename
166     ]);
167
168 } catch (Exception $e) {
169     $conn->rollback();
170     throw $e;
171 }
172
173

```

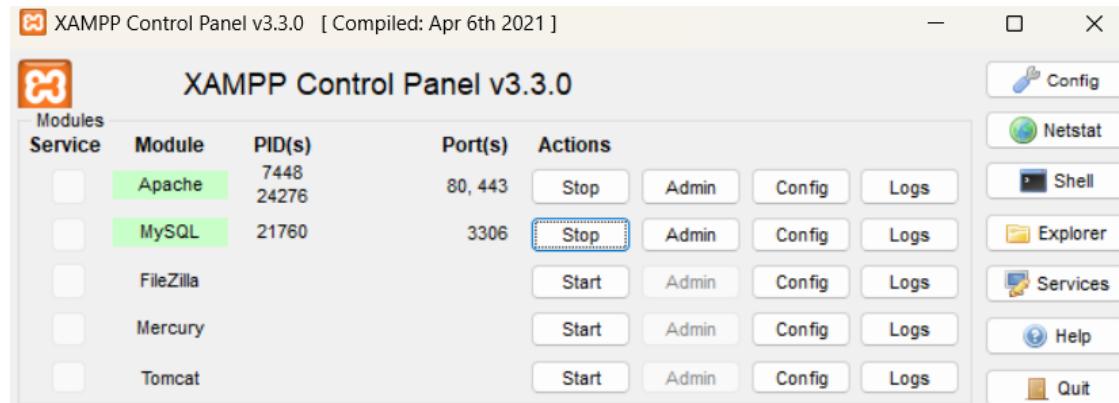
```

173 }
174 } catch (Exception $e) {
175     error_log('Upload error: ' . $e->getMessage());
176     echo json_encode([
177         'success' => false,
178         'message' => $e->getMessage()
179     ]);
180 }
181
182 $conn->close();
183 ?>

```



B. User's Manual



Make sure that XAMPP Apache and MySQL are up and running before clicking the desktop shortcut icon for ADMMS or opening your default browser and entering the following address: <http://localhost/>. The stop button indicates that MySQL and Apache are up and running. As a result, you can now use the system and database.

The image displays two screenshots of the ADMMS system. On the left is the login page, titled 'ADMMS', featuring two circular logos for 'Municipal Social Welfare and Development' and 'Archival Document Mapping Management System'. It includes fields for 'Username' and 'Password', and a 'Log In' button. Below the login form is the text 'Municipal Social Welfare and Development'. On the right is a purple-themed welcome screen with the text 'HELLO, WELCOME BACK!' in white. Below it, a yellow banner reads: 'Simplify your document management with our Archival Document Mapping Management System. Organize, map, and access your archival records effortlessly.' At the bottom of the welcome screen, a small note says 'Developed by: CCS 2024-2025 | Saint Michael College of Caraga (V0.1)'.

This is the log in page of both personnel and the admin. The personnel can only log in if they were given an account from the admin. It is a prevention of other user from accessing organization or admms.



The screenshot shows the Admin Panel interface for the Municipal Social Welfare and Development department. The main area displays a table of pending files. A red arrow labeled '1' points to the logo in the top left. Arrows labeled '2' point to the 'From:' and 'To:' date pickers. An arrow labeled '3' points to the 'Department' dropdown menu. An arrow labeled '4' points to the 'Filter' button. An arrow labeled '5' points to the 'Print' button. Below the table, a red arrow labeled '6' points to the 'View' button, and another red arrow labeled '7' points to the 'Approve' button. A third red arrow labeled '8' points to the 'Decline' button.

This is the admin UI for pendings you will encounter after logging in. The list of pending includes several key components: (1) a Date From filter to select the starting date, (2) a Date To filter to set the end date, (3) a Department Picker to choose the specific department for viewing related data, (4) a Filter button to apply the selected filters, (5) a Print button to print the filtered results or reports, (6) a View button to see detailed information of each request, (7) an Approve button to approve a request or document, and (8) a Decline button to reject it. These elements are designed to streamline navigation, filtering, and decision-making within the procurement dashboard.

The screenshot shows the Admin Panel interface for the Municipal Social Welfare and Development department. The main area displays a list of folders. A red arrow labeled '1' points to the 'Add Parent Folder' button in the sidebar.



This is the UI for adding parent folder. (1) The Add Parent Folder button under Manage let create a main folder to organize related subfolders and documents.

After redirecting to the Add Parent Folder section: The (1) Folder Name is where you enter the name of the folder, (2) Year is for specifying the folder's year, (3) Department is selected to assign the folder to a specific department, and (4) Create Folder button finalizes and adds the new parent folder.

After creating parent folder, (1) click the plus icon to expand and display options, including (2) New Folder to create a subfolder and (3) Upload File to add documents within the selected parent folder.



The screenshot shows a user interface for managing files. On the left is a sidebar with 'Archival Document Mapping Management System' and 'Admin Panel' sections. Under 'Manage', there are options for 'View Folders' and 'Add Parent Folder'. At the bottom is a 'Logout' button. The main area shows a 'Folders' list with a single item: 'Certificates 2024' with a note 'Missing location details'. A modal window titled 'Create New Folder' is open, containing fields for 'Folder Name' (1) and 'Add Metadata' (2). At the bottom of the modal are 'Cancel' (4) and 'Create' (3) buttons.

After clicking to the New Folder to create a subfolder, (1) input the Folder Name, (2) the admin can choice to add metadata, where they can specify the metadata name and input type, (3) click Create to finalize the subfolder, or (4) click Cancel to abort the process.

The screenshot shows a user interface for managing files. On the left is a sidebar with 'Archival Document Mapping Management System' and 'Admin Panel' sections. Under 'Manage', there are options for 'View Folders' and 'Add Parent Folder'. At the bottom is a 'Logout' button. The main area shows a 'Folders' list with a single item: 'Glass Apr 07, 2025 @ Location Row 1, Column 2'. A modal window titled 'Upload File' is open, containing fields for 'Select File' (1), 'File Name' (2), 'File Owner' (3), 'Date of Purchase' (4), 'Supplier' (5), and 'Amount' (6). At the bottom of the modal are 'Cancel' (7) and 'Upload' (8) buttons.

After tapping Upload File, a form will appear with the following fields: (1) Select File allows you to browse and choose a file from your device or computer, (2) File Name is where you can give the file a unique name for easy identification, (3) File Owner is where you enter the person for the file, (4,5,6) Metadata Description where you can add details from the admin or personnel metadata to better categorize the file, (7) Cancel lets you exit the upload process without saving



any changes, and (8) Upload will submit the file along with the entered metadata, completing the process. This ensures all relevant information is captured for proper tracking and organization.

A screenshot of the Archival Document Mapping Management System. The left sidebar shows navigation options: Dashboard, Reports, Uploaded Files (with a dropdown), Manage (with a dropdown), Accounts (with a dropdown), and a Logout button. The main area displays a table titled "Folders" under "Municipal Social Welfare and Development". The table has columns: #, File name, Upload date, File owner, Uploaded by, Date of Purchase, Supplier, Amount, and File. One row is shown: #1, Glass, Apr 07, 2025, Elmerto, Winston Lloyd B. Timogan, March 25, 2025, Jr Glass, 10,500. A red arrow points to the "File" icon in the last column of this row. A blue plus sign icon is in the bottom right corner of the main area.

#	File name	Upload date	File owner	Uploaded by	Date of Purchase	Supplier	Amount	File
1	Glass	Apr 07, 2025	Elmerto	Winston Lloyd B. Timogan	March 25, 2025	Jr Glass	10,500	

After uploading the file, (1) tapping the PDF file icon will open and display the uploaded PDF file, allowing you to view its contents directly within the system.



C. Letter of Permission

December 09, 2024

Ms. Maria Bernadita N. Quijano, RSW
MSWD Officer
Municipal Social Welfare and Development Office
Buenavista, Agusan del Norte, 8601

Dear Ma'am,

Greetings!

MSWD OFFICE
RECEIVED
SIGNER: *E. Quijano*
DATE: 12-9-2024
TIME: 1:28 PM
MUNICIPALITY OF BUENAVIDA, ADN

We hope this letter finds you well. We are the 3rd year students of BSIT of Saint Michael College of Caraga. We are writing to formally request permission and seek collaboration for us for our ongoing research study entitled "**WEB-BASED ARCHIVAL AND DOCUMENT MAPPING FOR BUENAVIDA'S MUNICIPAL SOCIAL WELFARE SERVICES**"

We are writing to formally request your permission and seek collaboration for the development and implementation of our system. Our study aims to improve efficiency and ensure centralized document management.

As part of this initiative, we would like to request [specific support needed, e.g., access to relevant data, policies, or feedback from your team]. Rest assured that we will adhere to any protocols or guidelines set by your office, and any information shared will be treated with utmost confidentiality and used strictly for academic purposes.

We believe this project will significantly enhance the efficiency of archival and document mapping within your organization, aligning with the institution's mission of fostering innovation through practical research. Additionally, the system's outcomes could serve as a benchmark for other municipalities seeking to adopt similar solutions.

Thank you for considering our request. We look forward to your positive response and are more than willing to discuss this project further at your convenience.

Sincerely,

Winston Lloyd B. Timogam
Winston Lloyd B. Timogam
Research Leader



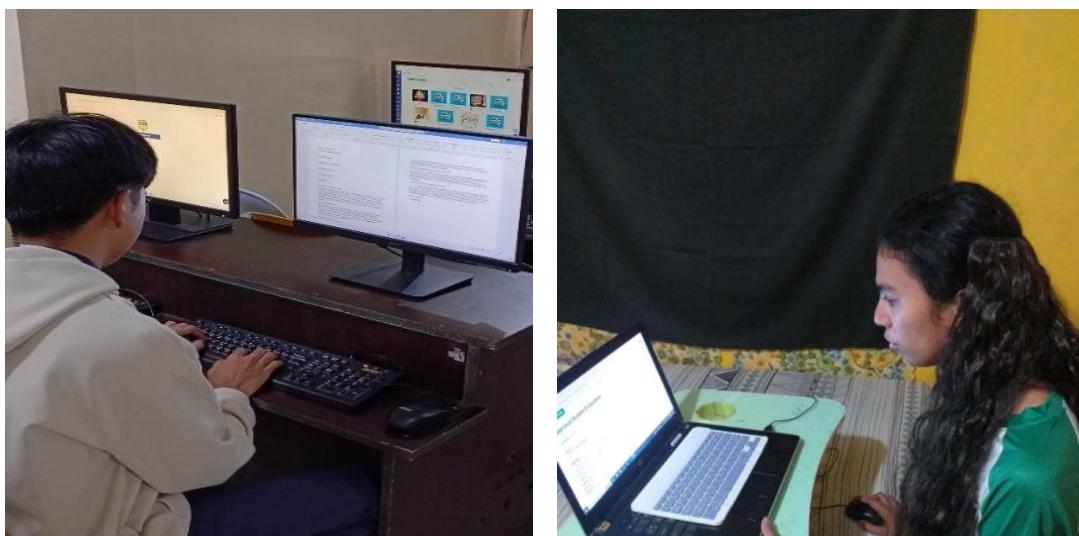
D. Documented Undertakings

Title Hearing



Researchers successfully passed one title during the title hearing.

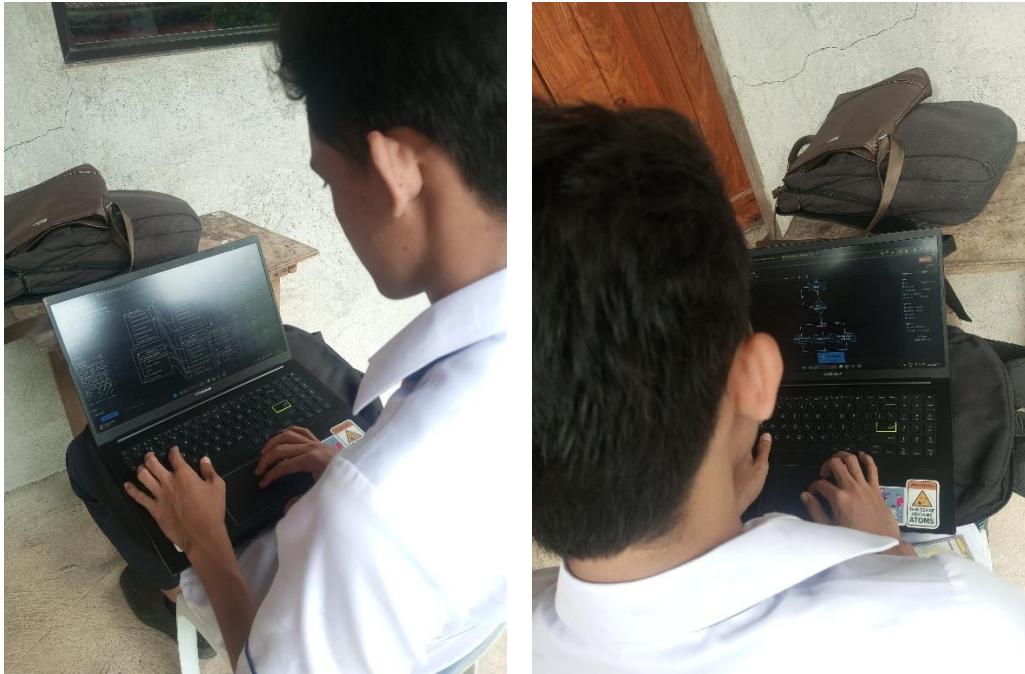
Chapter II Making



Researchers during the formulation of Chapter 2.



Diagrams Making



Researcher during the formulation of diagrams for Chapter 3.

Title Proposal



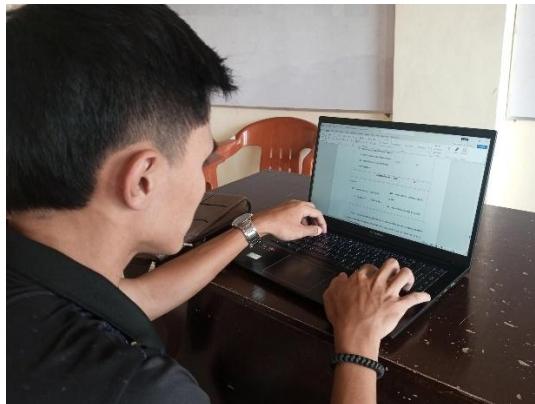
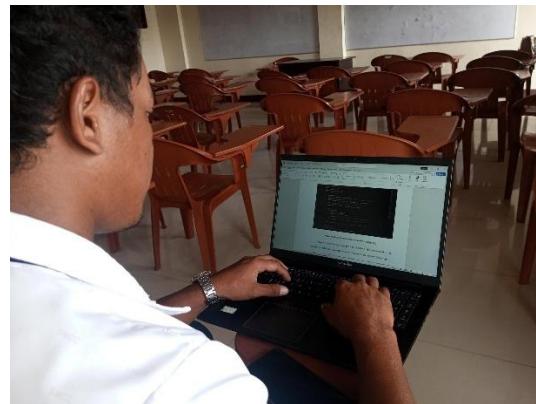
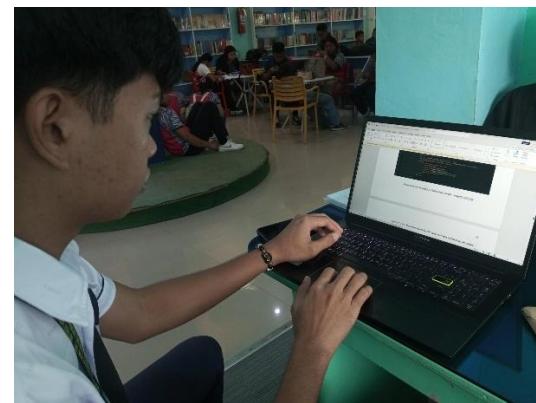
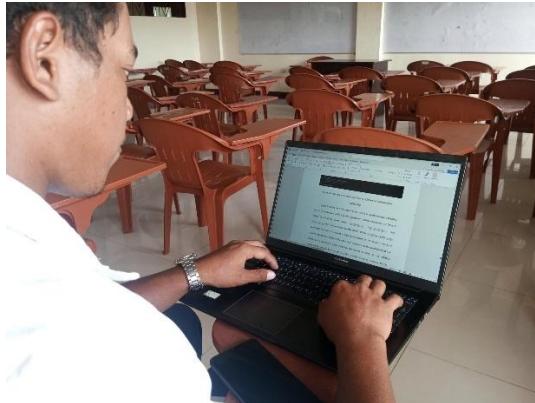
Researchers successfully defended Chapter 1 to 3 during the research proposal.

Developing a System



The Main Developer was responsible for designing, coding, and integrating the system's core features, managing database connections, creating technical diagrams, and ensuring the system met all functional requirements through testing and refinement.

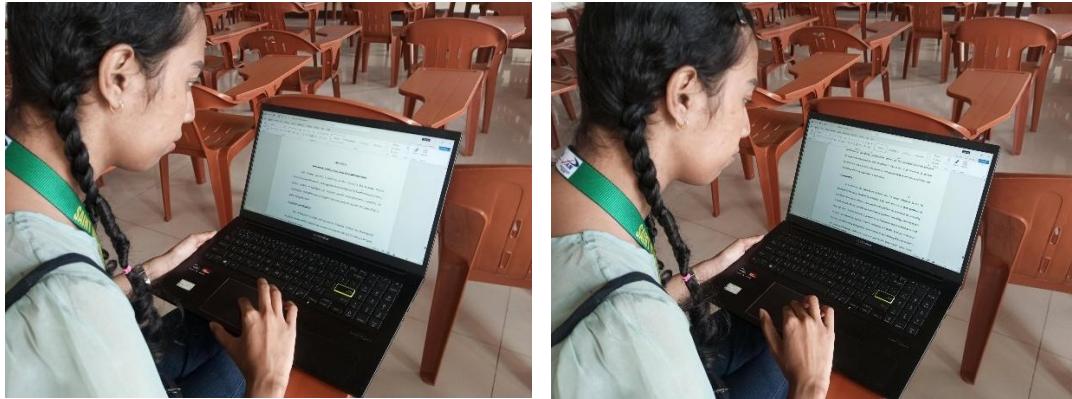
Chapter 4 Making



Researchers, during the formulation of Chapter 4, arranged the code, included the discussion, and tallied the results of the end users' evaluations. They focused on analyzing the code's functionality and provided insights based on the feedback collected from the users.

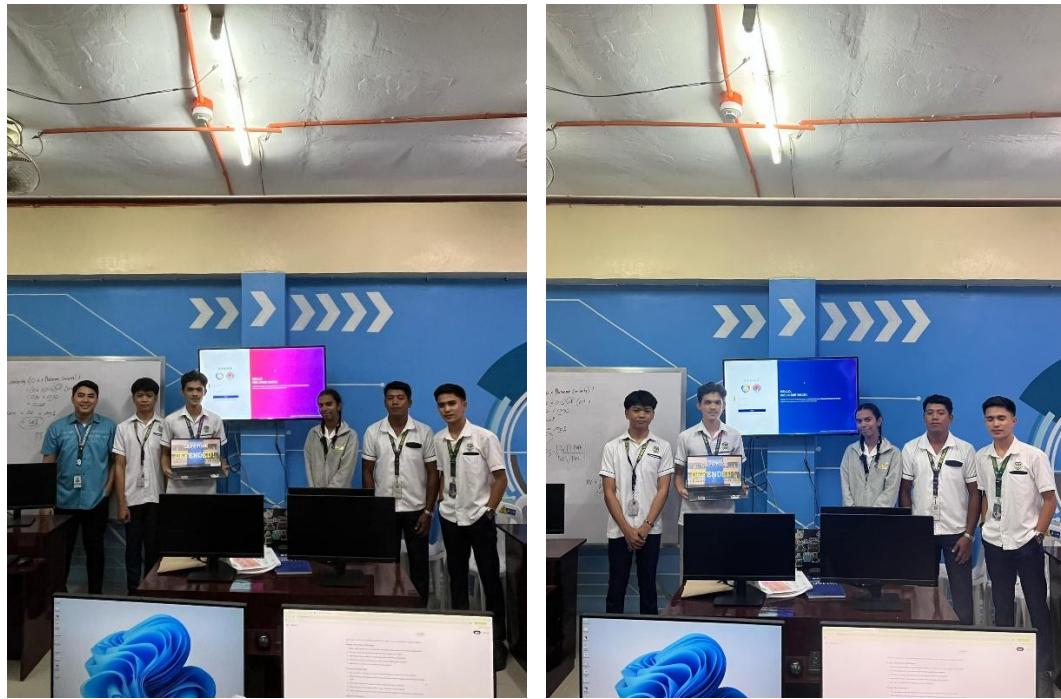


Chapter 5 Making



Researchers, in the development of Chapter 5, were tasked with synthesizing the study's findings, formulating conclusions, and providing well-founded recommendations. They critically analyzed the results, drawing key insights, and offered practical suggestions for future improvements or research directions, grounded in the study's outcomes.

Final Defense



Researchers successfully defended the final oral defense.



E. Certificate of Implementation



F. Research Locale



**G. Curriculum Vitae**

Name : Winston Lloyd B. Timogan

Address : D-3 Culit, Nasipit, Agusan Del Norte

Contact : 09553556487

Email : timogan001@gmail.com

**Personal Details**

Date of Birth : November 7, 2003

Gender : Male

Civil Status : Single

Age : 21

Citizenship : Filipino

Educational Background

Elementary : Culit Elementary School

Junior High School : Northwestern Agusan Colleges

Senior High School : Northwestern Agusan Colleges

Skills

- Programming

Membership School Organization

- Collegiate Computing Society (2024 - 2025)
- Michaelinian Security Guild (2024 - 2025)



G. Curriculum Vitae

Name : Juli-Ann S. Echalico

Address : P-1 Guinabsan, Buenavista, ADN

Contact : 09771379876

Email : akrawatanab20@gmail.com



Personal Details

Date of Birth : February 24, 2003

Gender : Female

Civil Status : Single

Age : 21

Citizenship : Filipino

Educational Background

Elementary : Tanutao Elementary School

Junior High School : Saint James High School

Senior High School : Saint James High School

Skills

- Programming

Membership School Organization

- N/A



G. Curriculum Vitae

Name : James Nolie S. Piel

Address : Purok-1, Brgy-4, Buenavista, ADN

Contact : 09635921813

Email : jamesnolie18@gmail.com



Personal Details

Date of Birth : April 12, 2002

Gender : Male

Civil Status : Single

Age : 22

Citizenship : Filipino

Educational Background

Elementary : Buenavista Central Elementary School

Junior High School : Saint James High School

Senior High School : Saint James High School

Skills

- Programming

Membership School Organization

- Michaelinian Cybersecurity Guild (2024 - 2025)



G. Curriculum Vitae

Name : John Carlo M. Bonotan

Address : Purok-4, Brgy-10 Buenavista, ADN

Contact : 09058394297

Email : bonotanjohncarlo@gmail.com



Personal Details

Date of Birth : March 1, 2003

Gender : Male

Civil Status : Single

Age : 21

Citizenship : Filipino

Educational Background

Elementary : Buenavista Elementary School

Junior High School : Saint James High School

Senior High School : Saint James High School

Skills

- Programming
- Video Editing

Membership School Organization

- N/A



G. Curriculum Vitae

Name : Elmerto G. Osigan Jr.

Address : Punta D-6 Nasipit, Agusan Del Norte

Contact : 09207373695

Email : elmertoosigan101@gmail.com



Personal Details

Date of Birth : April 14, 1999

Gender : Male

Civil Status : Single

Age : 25

Citizenship : Filipino

Educational Background

Elementary : Punta Elementary School

Junior High School : Northwestern Agusan Colleges

Senior High School : Northwestern Agusan Colleges

Skills

- Programming

Membership School Organization

- N/A



CERTIFICATE OF AUTHENTIC AUTHORSHIP

This is to certify that I/we, the undersigned author(s), have solely and honestly written the entire paper titled: "**Web-based Archival and Document Mapping for Buenavista's Municipal Social Welfare Services**" All contents, ideas, data, and analysis presented in this paper are the results of my/our own diligent work and intellectual effort. The paper has been crafted to the best of my/our abilities, in full compliance with academic integrity standards and ethical research practices.

I/We further declare that:

1. This paper is free from any form of plagiarism or copyright infringement.
2. All referenced materials have been properly cited, and due credit has been given to the original authors and sources.
3. The work does not contain any falsified or fabricated data or information.

I/We acknowledge that if any rules or laws related to copyright, plagiarism, or other related legal or ethical standards are violated, Saint Michael College of Caraga shall not be held accountable for my/our actions. Any legal, academic, or ethical consequences arising from such violations shall solely be my/our responsibility.

Signed this

Date: [Click here to enter text.](#)

Location: [Click here to enter text.](#)

Author(s):

WINSTON LLOYD B. TIMOGAN

Author

JULI-ANN S. ECHALICO

Author

JOHN CARLO M. BONOTAN

Author

JAMES NOLIE S. PIEL

Author

ELMERTO G. OSIGAN JR.

Author