SAINT MICHAEL COLLEGE OF CARAGA

6WEB-BASED ARCHIVAL AND DOCUMENT MAPPING FOR BUENAVISTA'S MUNICIPAL SOCIAL WELFARE SERVICES

A Capstone Project 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
JOHN CARLO M. BONOTAN
JULI-ANN S. ECHALICO
ELMERTO G. OSIGAN JR.
JAMES NOLIE S. PIEL

ADVISER: MARLON JUHN M. TIMOGAN

APRIL 2025

SAINT MICHAEL COLLEGE OF CARAGA
College Of Computing and Information Sciences

CERTIFICATE OF RESEARCH APPROVAL

This capstone project 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

COLLEGE OF COMPUTING AND INFORMATION SCIENCES

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 PUBLICATION:	Saint Michael College of Caraga
DATE	April 2025
PAGES	181

DEVELOPMENTAL RESEARCH

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, establish Role-Based Access Control through user authentication and permission settings, safeguard the confidentiality of private data by limiting access to authorized users, and 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

Based on ISO 25010 standards, the system received high functionality, performance efficiency, and usability ratings, reflected in a high total weighted mean of 3.65. This confirms 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, Role-Based Access Control

DEDICATION

This capstone research is dedicated with heartfelt appreciation to those who have contributed to our journey and success.

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

The researchers are deeply grateful to their adviser, Mr. Marlon Juhn Timogan, for your invaluable guidance, support, and dedication. Your insightful advice, constructive feedback, and encouragement were vital in shaping and completing this study. Your mentorship has left a lasting mark on our academic and personal growth.

This work also reflects our collaboration, hard work, and shared commitment. Each contribution, no matter how big or small, helped transform 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 the researchers from beginning to end. Thank you.

ACKNOWLEDGEMENT

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

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

The researchers also thank their instructors and panels members for their helpful insights and support, which were essential in refining our study.

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

The researchers are especially grateful to the Tertiary Education Subsidy for providing the financial assistance that allowed them to focus on their academic goals without additional burden.

The researchers especially thanks their friends, peers, and mentors for supporting them 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, the researchers dedicate this capstone to all of you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 Project Context

The Municipal Social Welfare Development is a local government agency charged with formulating a plan for administering and rendering social safety nets throughout their jurisdiction and implementing poverty alleviation programs to support 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 Socia 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 would 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 would 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. Archival procedures would help reduce human errors while 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 would facilitate effective service delivery in the future.

1.2 Objective of the Study

This study aims to design, develop, and implement a web-based archival and document mapping system for Buenavista's Municipal Social Welfare Services.

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 and ensure that access to specific 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 would provide comprehensive archival and document mapping services tailored to Buenavista's Municipal Social Welfare and Development Office. It is designed to streamline the management and organization of digital and physical records, ensuring secure storage and easy retrieval. This implementation would support the office's operational requirements, enhancing document accessibility, confidentiality, and overall efficiency in handling essential records.

The system would 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 would 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 is only accessible on smartphones, laptops, and computers 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 web-based platform is 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—This 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 assigned per category, which helps easily manage the documents.

Digital formats—These formats are used for storing information electronically. One common format is PDF for files.

Document Archiving – Storing documents in digital format within the system for quick and easy records retrieval overtime 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 primary 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 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 government branch that provides social protection to the less privileged and marginalized and encourages the disadvantaged 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 to these needs, supporting 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 users' actions, 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]. Traditional paper-based management is still standard, but technological advancements have shifted organizations to electronic systems. 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 strengthening 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 essential for document storage accessibility and efficiency because offices have already streamlined their workflows, ensuring documents can be easily retrieved and stored correctly.

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 significant advantages, but let us not undermine their barriers, 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]. Regarding organizational benefits, Jordan S. highlights that Document Management Systems (DMS) offer substantial 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 storing and accessing resident information compared to traditional management methods, making it an efficient solution for community organizations [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]. Technological advancements have made automated systems 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 extensive 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 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 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 called 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 much 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 the administration and review of access rights [32]. Junior M. et al. noted that 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

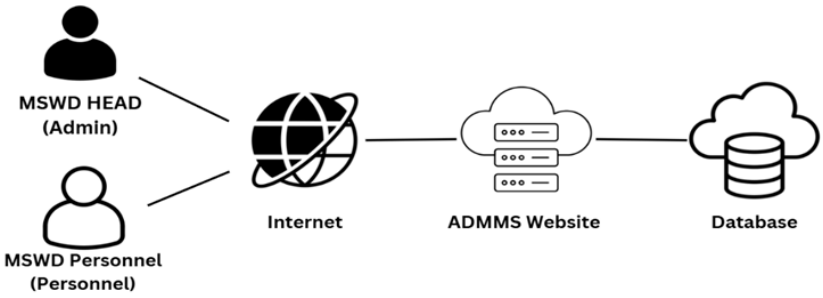


Figure 1. System Architecture

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 and create and view personnel accounts.

3.2 Conceptual Diagram

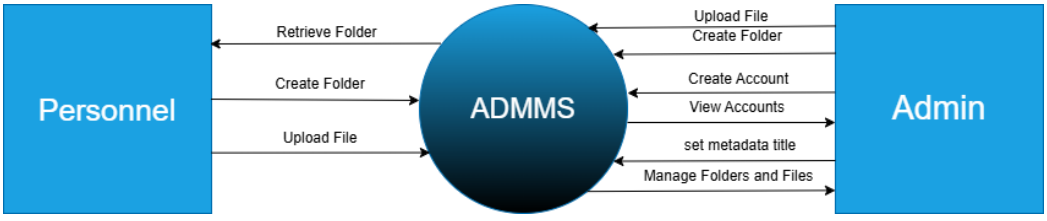


Figure 2. Conceptual Diagram (Level 0)

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 uploading files, creating accounts, and setting metadata for personnel. On the other hand, personnel can create folders and upload relevant files in the ADMMS.

3.3 Use Case Diagram

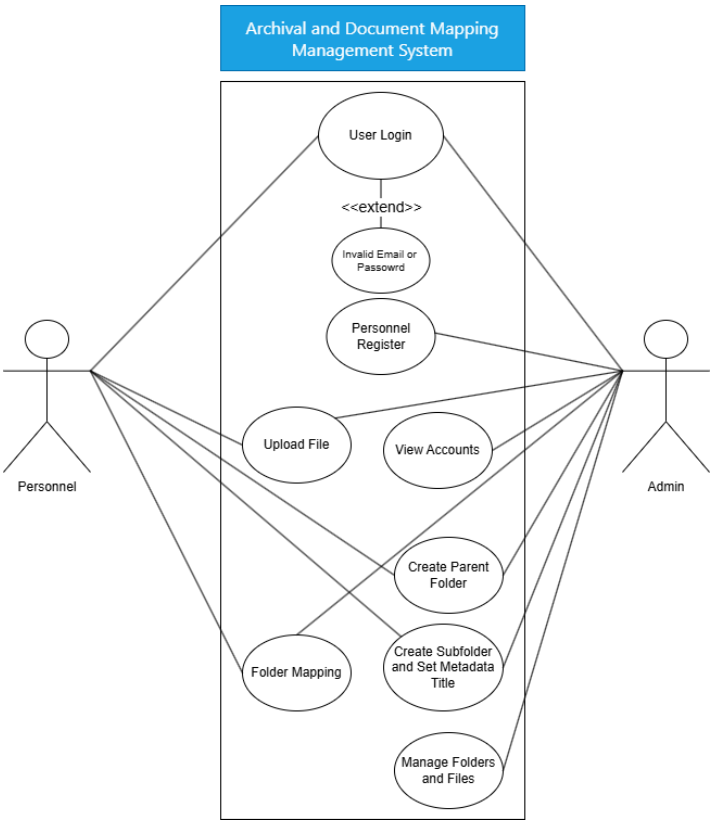


Figure 3. Use Case Diagram

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 uploading files, creating accounts, and setting metadata for personnel.

3.4 Activity Diagram

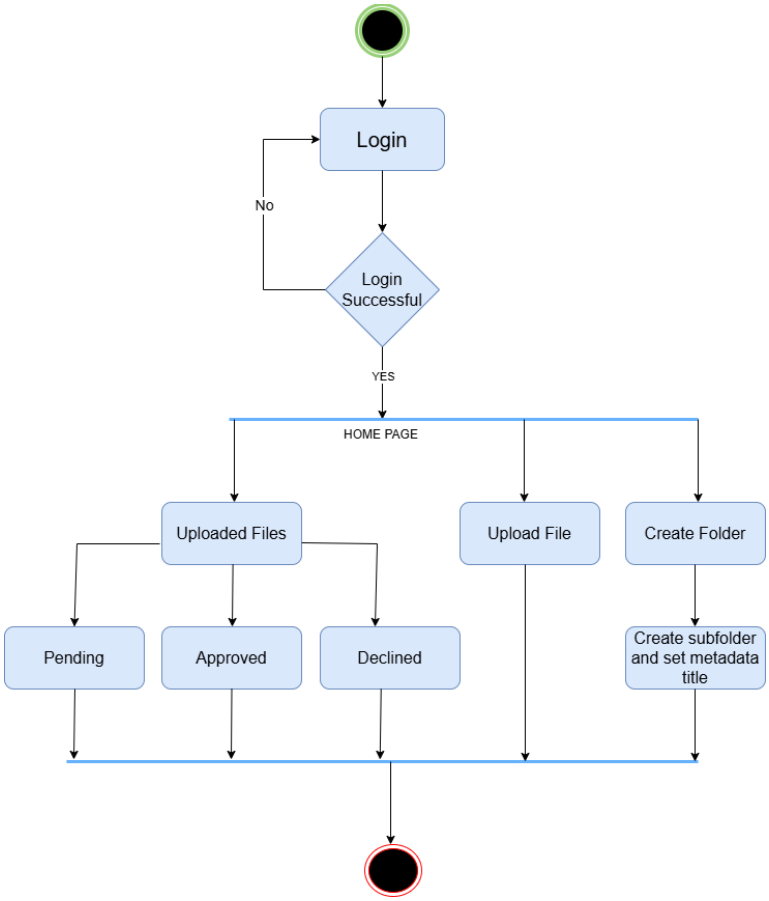


Figure 4. Activity Diagram – Personnel's

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, they can ask the administrator to create or give them one if they do not have an account yet. After successfully logging in, personnel are directed to the ADMMS homepage, where they can upload files and create folders. Uploaded files fall into three categories: pending, approved, and declined. The personnel may also log out of their accounts.

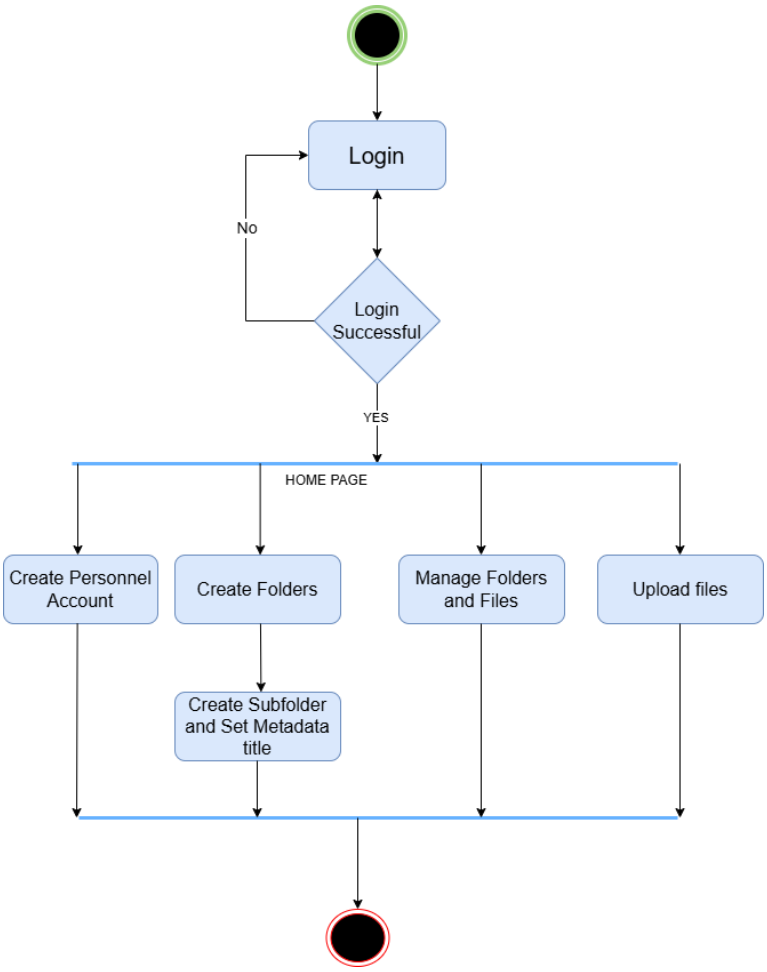


Figure 5. Activity Diagram – Admin

Figure 5 illustrates the series of activities available to the admin. After successful admin verification, users are forwarded to the homepage on the login page. The admin capabilities are viewing accounts, managing folders, creating accounts and folders, setting metadata for personnel, and uploading file.

3.5 Sequence Diagram

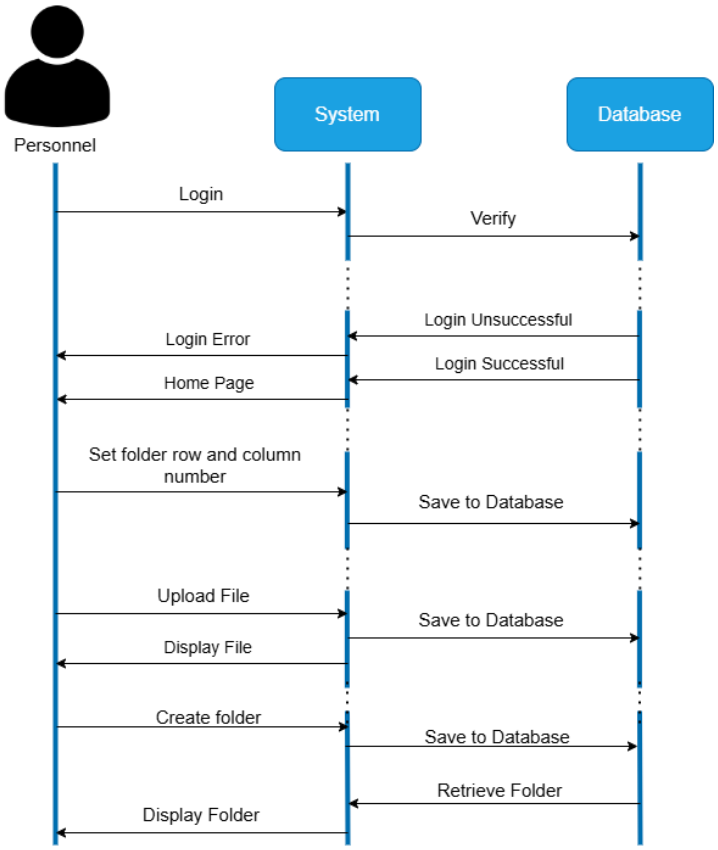


Figure 6. Sequence Diagram – Personnels

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 numbers in the folder and uploading files for database storage.

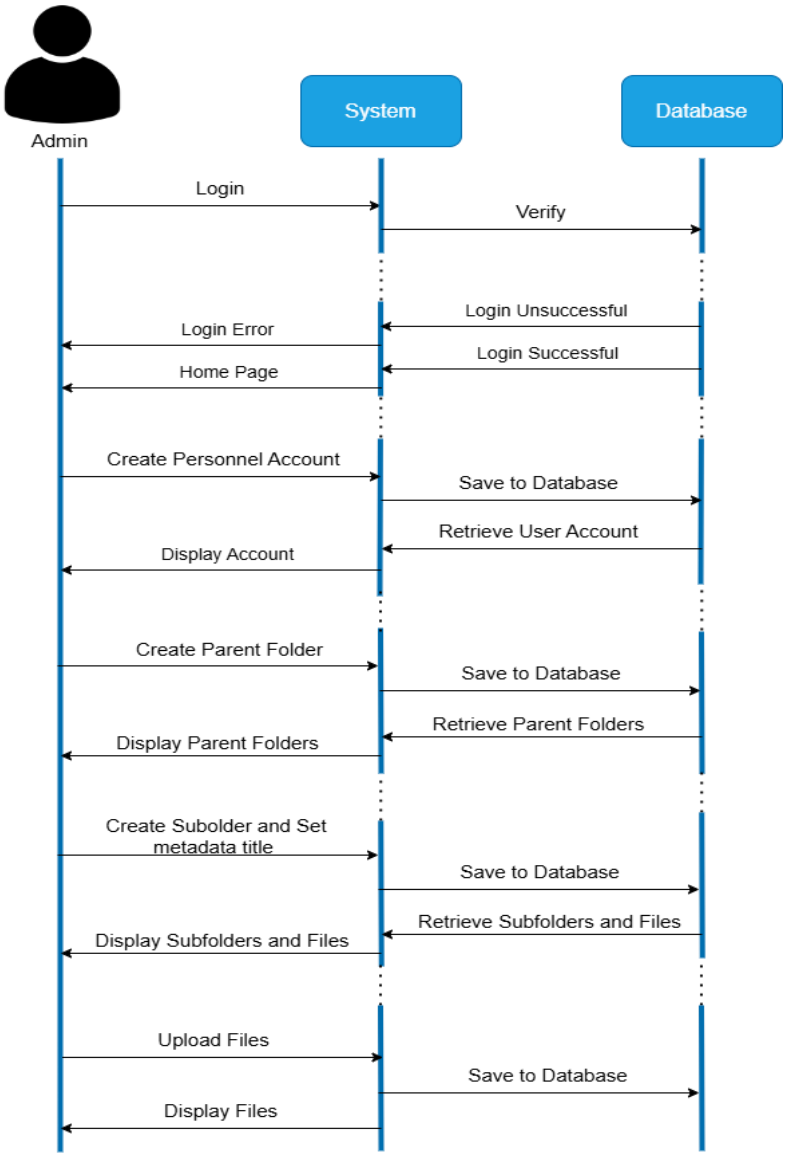


Figure 7. Sequence Diagram – Admin

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

3.6 ERD (Database Design)

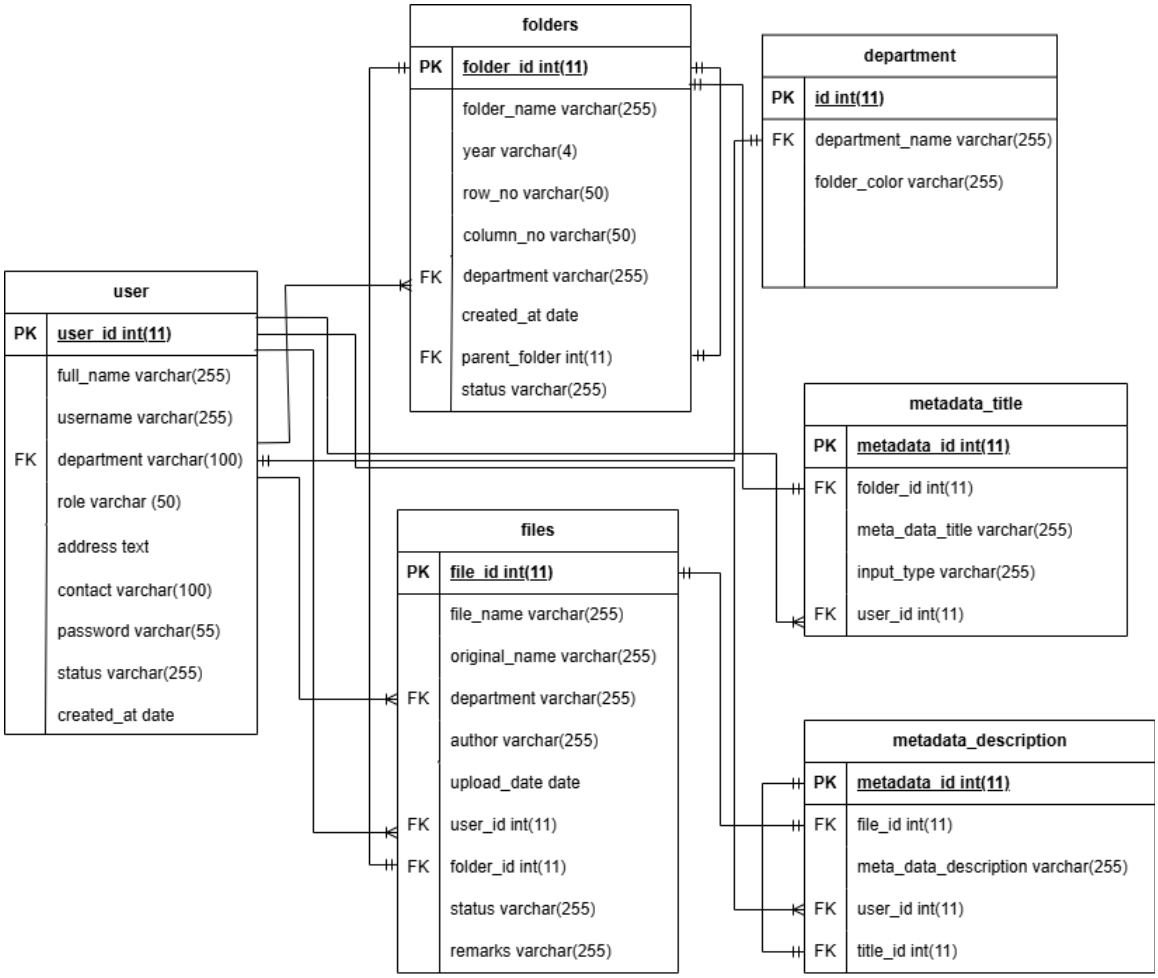


Figure 8. Entity Relationship Diagram – Database Design

Figure 8 presents a small schema for a Web-based Archival and Documents Mapping Management System (ADMMS). The ERD includes users, folders, departments, files, metadata titles, and descriptions. 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)

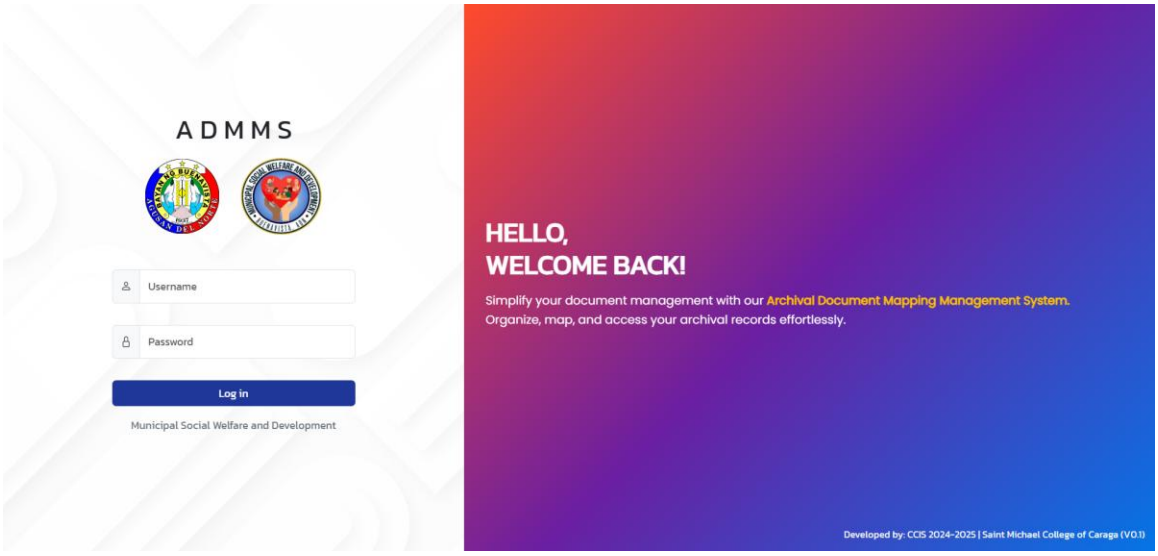


Figure 9. Personnel / Admin – Login Page

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

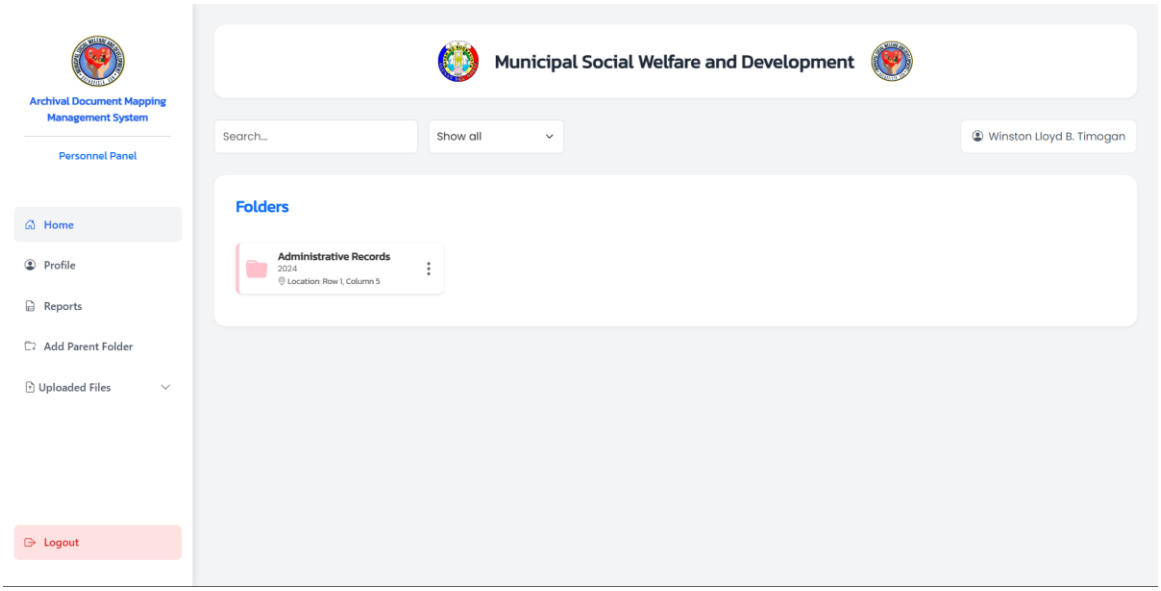


Figure 10. Personnel Side – Home Page (Personnel Interface Design)

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.

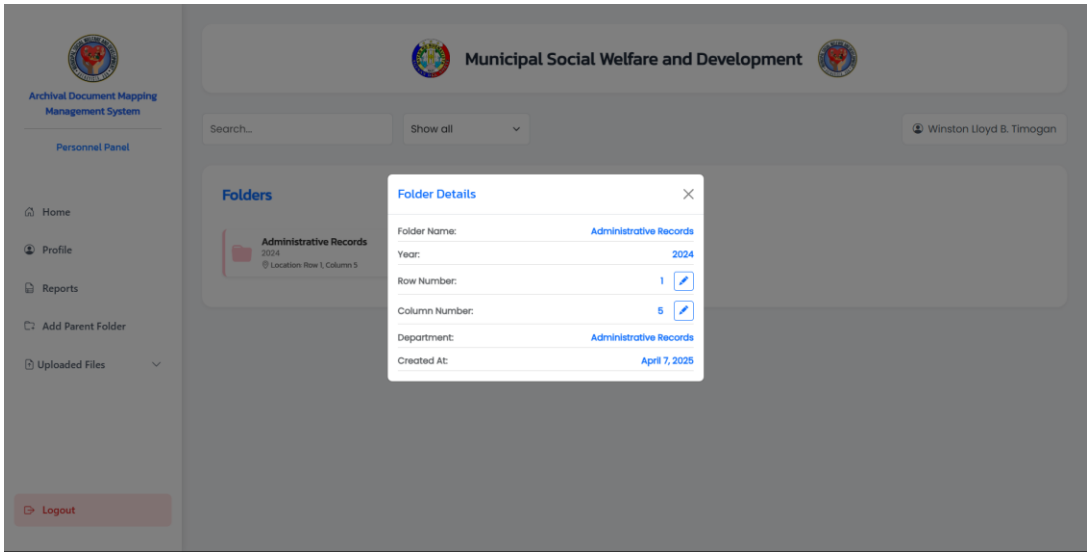


Figure 11. Personnel Side – Folder Details (Personnel Interface Design)

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.

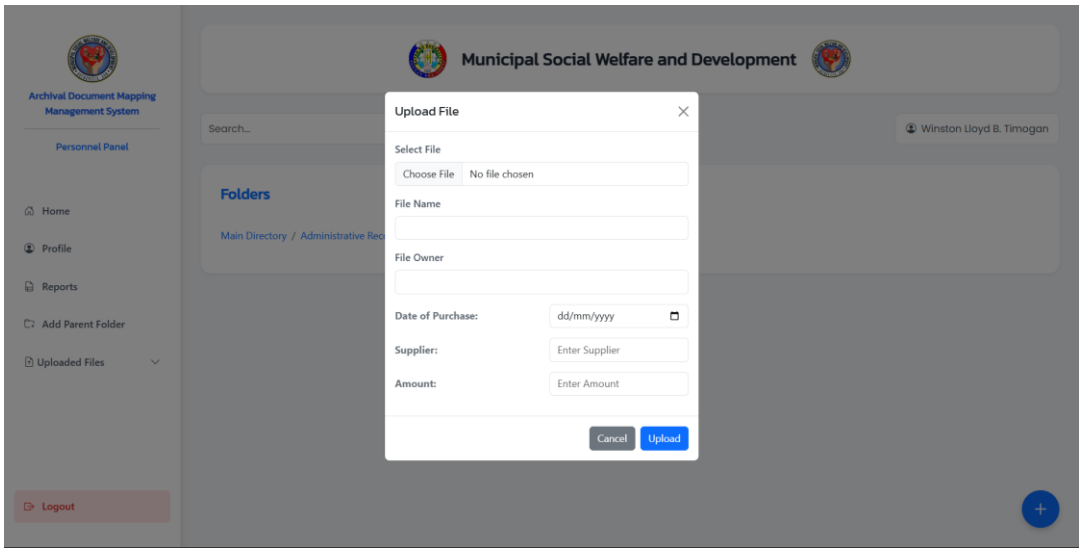


Figure 12. Personnel Side – Upload File (Personnel Interface Design)

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.

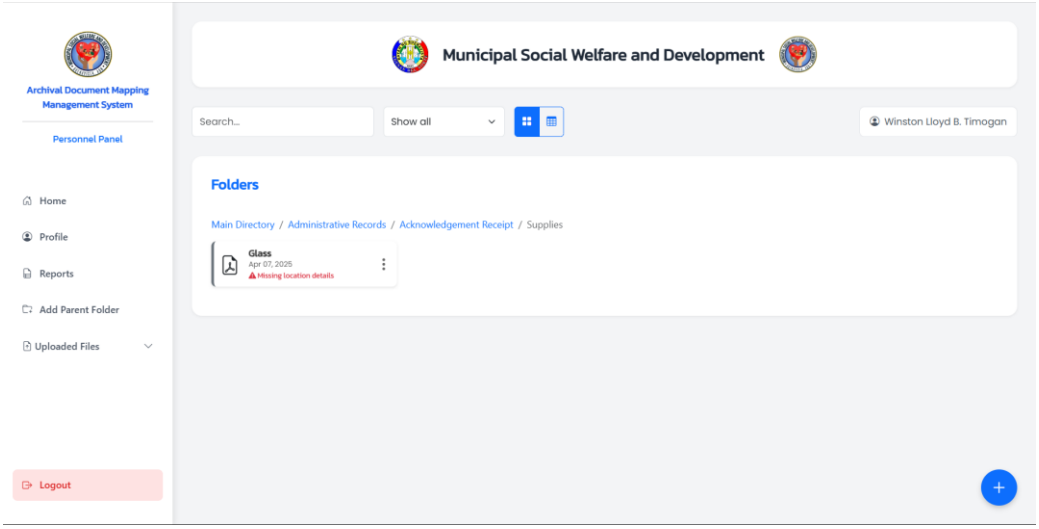


Figure 13. Personnel Side – Display File Card Style (Personnel Interface Design)

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.

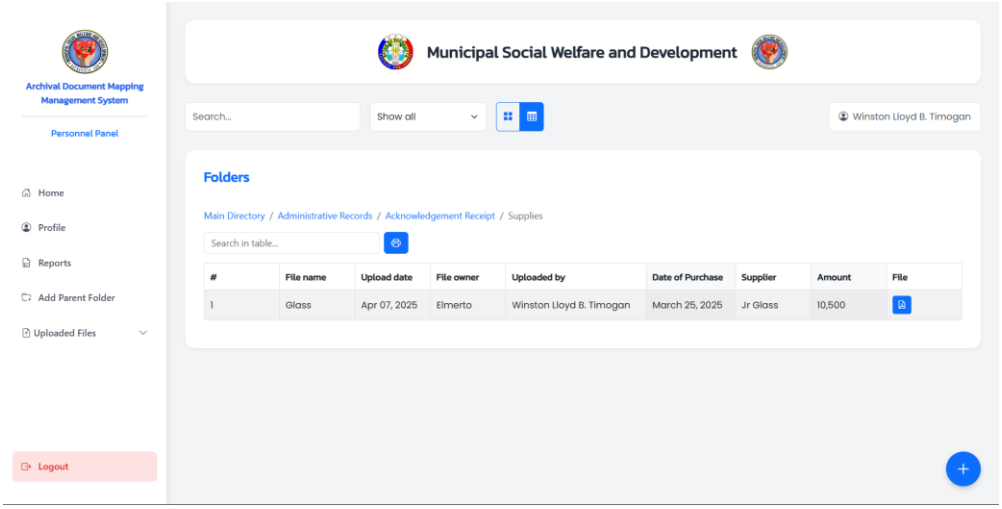


Figure 14. Personnel Side – Display File Table Style (Personnel Interface Design)

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 files. This layout ensures an organized and efficient file management experience for personnel.

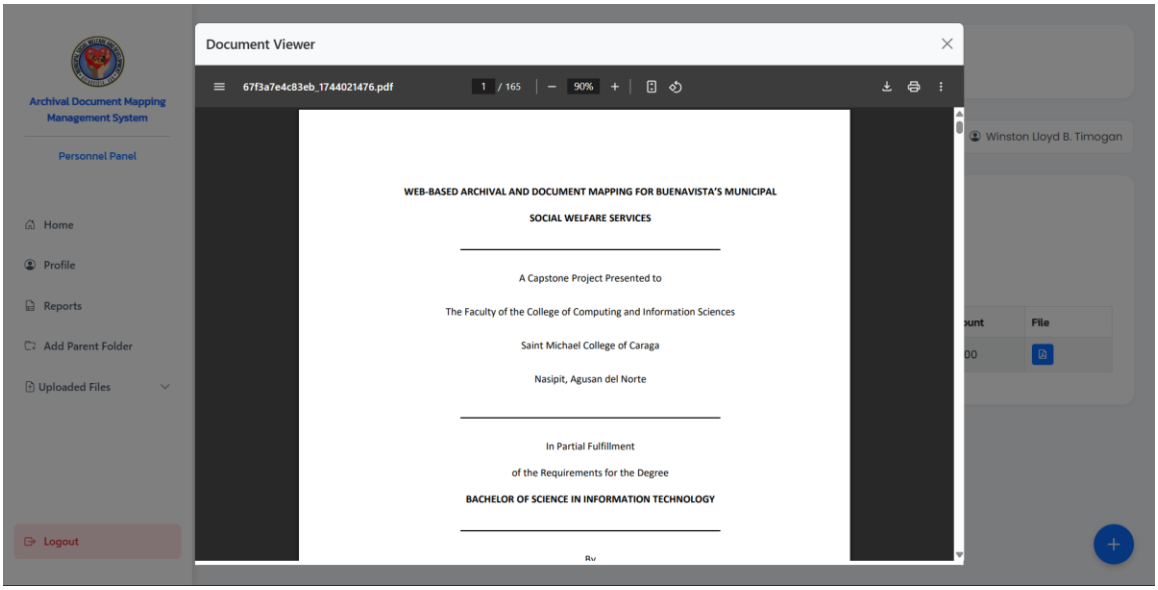


Figure 15. Personnel Side – View PDF File (Personnel Interface Design)

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 popup window. The modal includes options for zooming, scrolling, and closing the document, ensuring convenient access to file content while maintaining workflow efficiency.

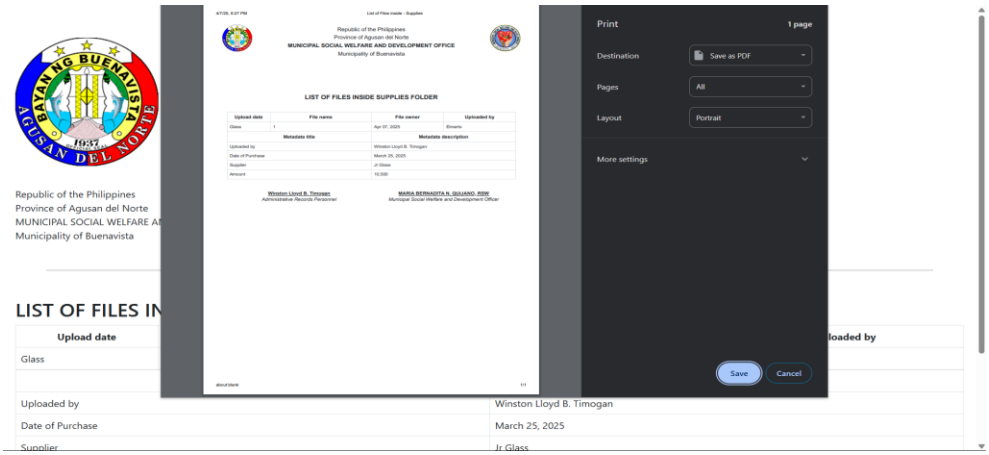


Figure 16. Personnel Side – Generate PDF and Print (Personnel Interface Design)

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

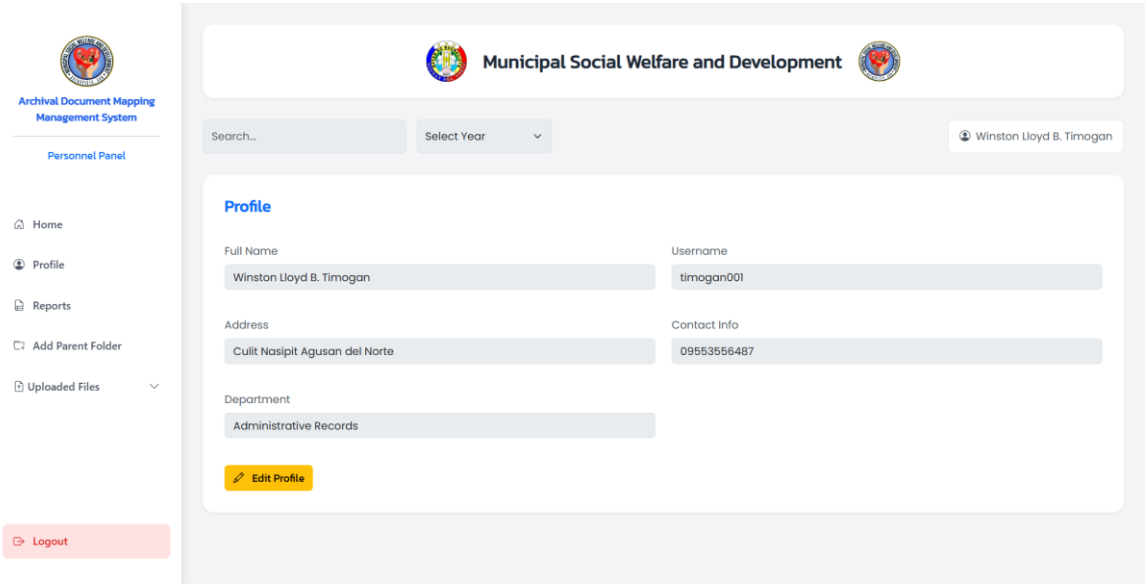


Figure 17. Personnel Side – Profile (Personnel Interface Design)

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.

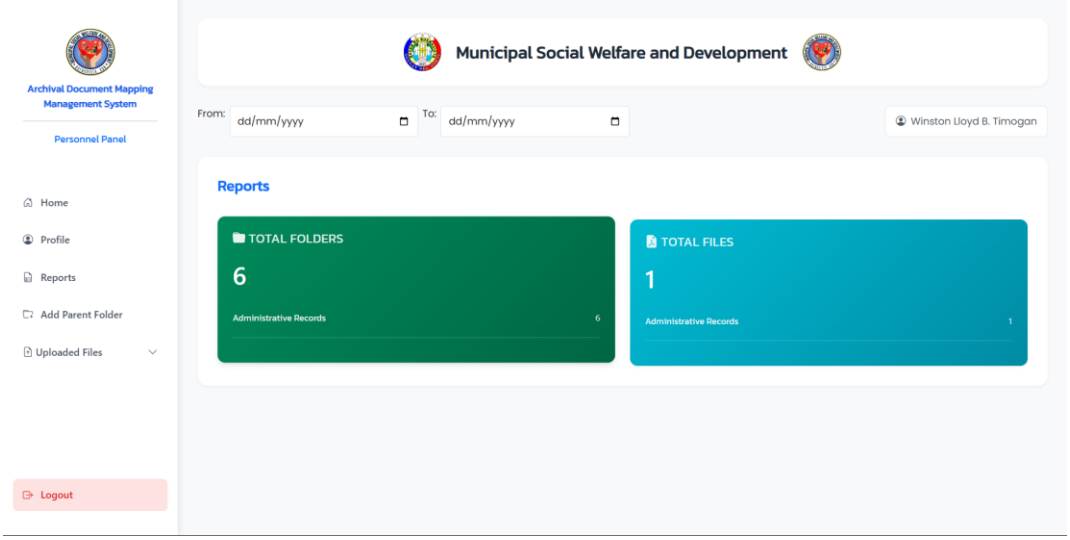


Figure 18. Personnel Side – Reports (Personnel Interface Design)

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 files managed by the personnel. The report view provides a clear and organized overview, ensuring efficient record management within the designated department.

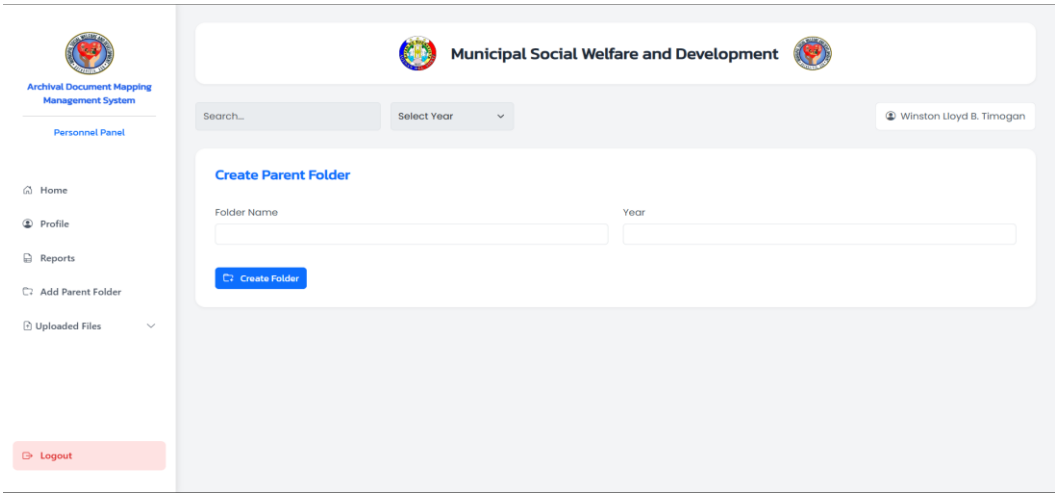


Figure 19. Personnel Side – Add Parent Folder (Personnel Interface Design)

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

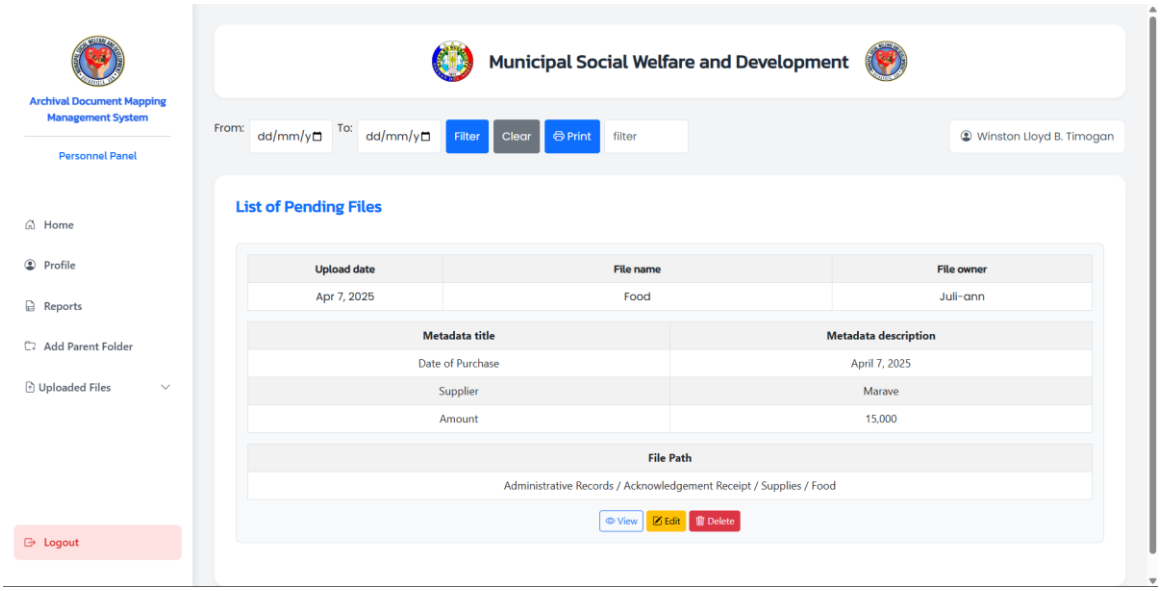


Figure 20. Personnel Side – List of Pending Files (Personnel Interface Design)

Figure 20 displays the personnel interface's List of Pending Files section, showing files awaiting approval. This Section presents a structured view of all pending documents, including upload date, file name, file owner, metadata title, and metadata description.

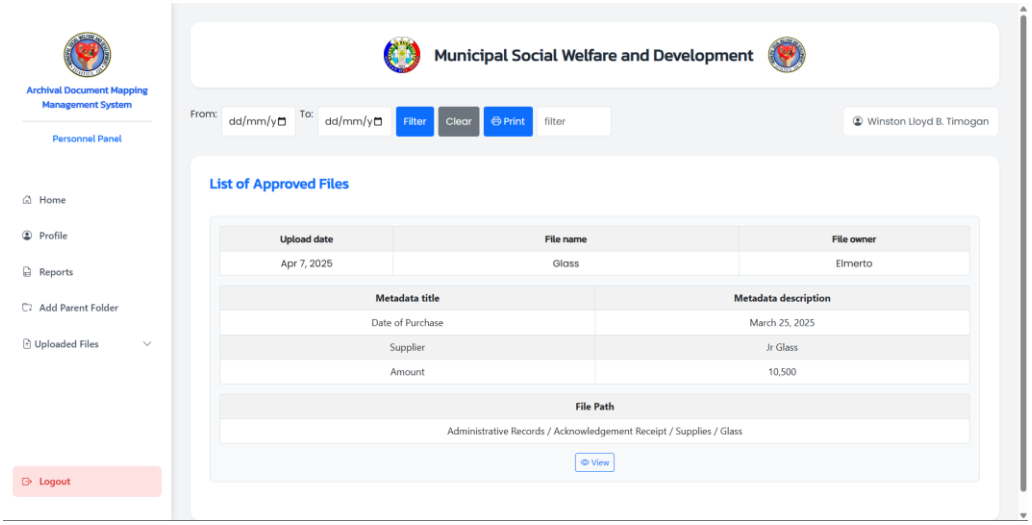


Figure 21. Personnel Side – List of Approved Files (Personnel Interface Design)

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 upload date, file name, file owner, metadata title, and metadata description. The interface enables personnel to access and manage their approved files for further use or reference.

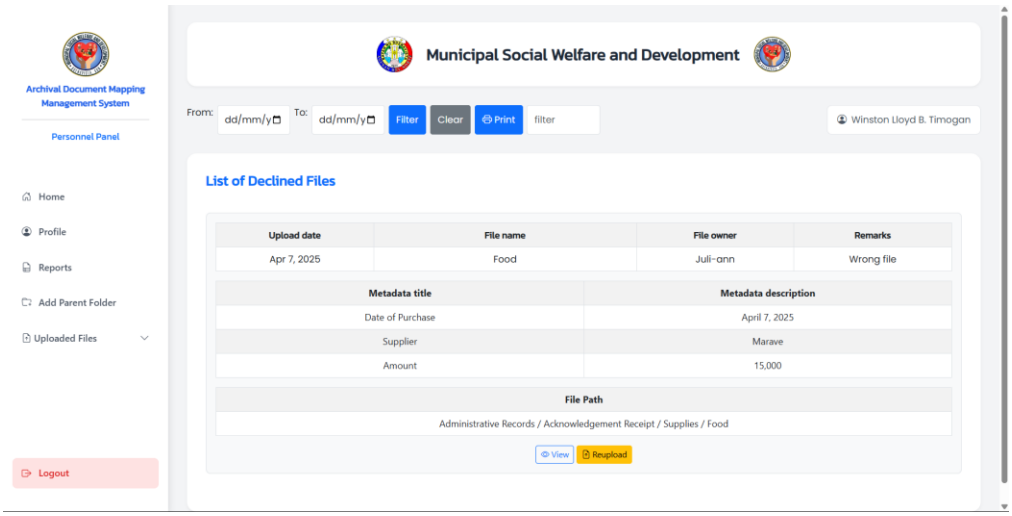


Figure 22. Personnel Side – List of Declined Files (Personnel Interface Design)

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 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.

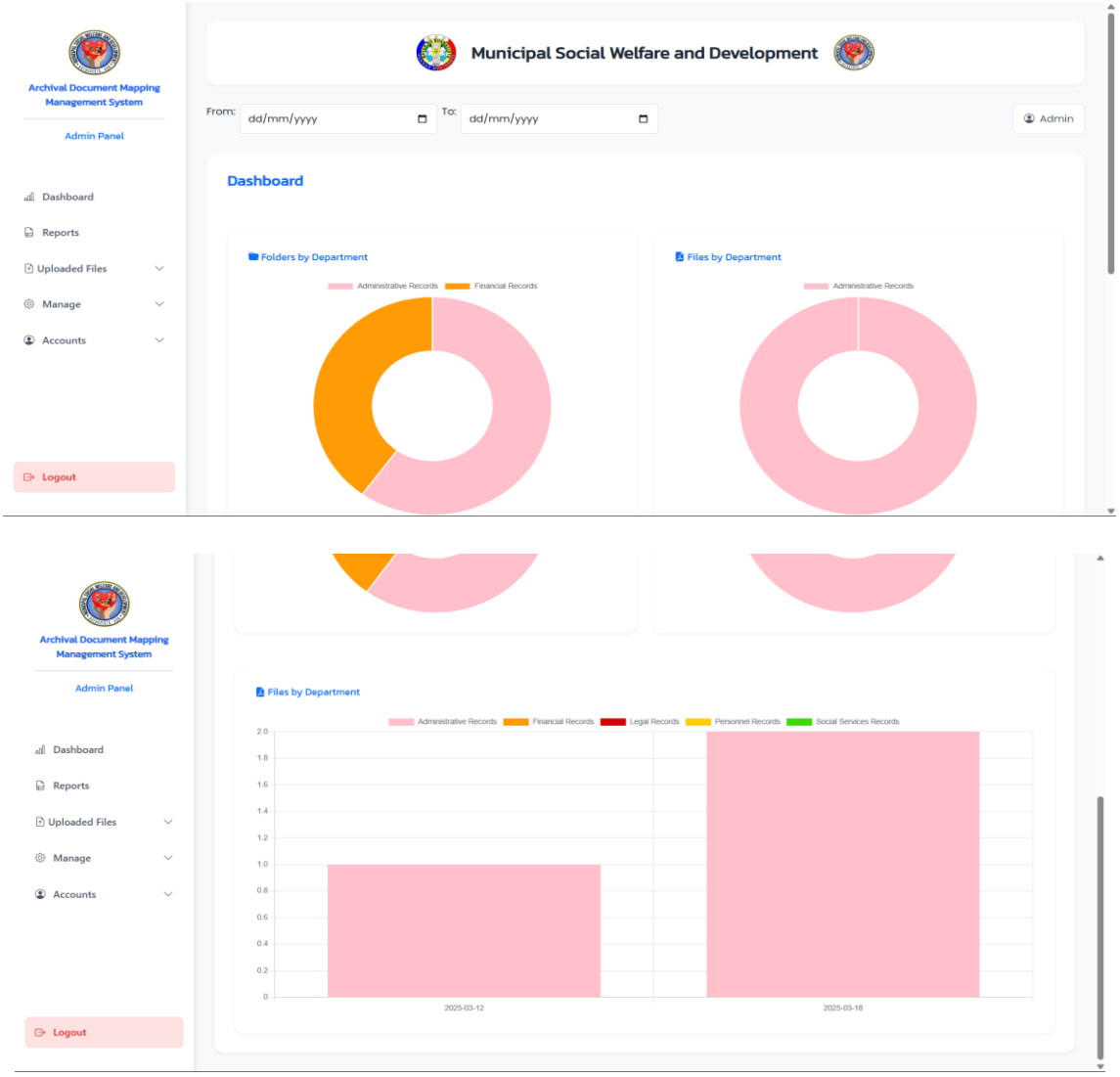


Figure 23. Admin Side – Dashboard (Admin Interface Design)

Figure 23 presents the Dashboard on the admin interface, displaying charts and graphs that visually represent system data. These graphical elements illustrate folders 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.

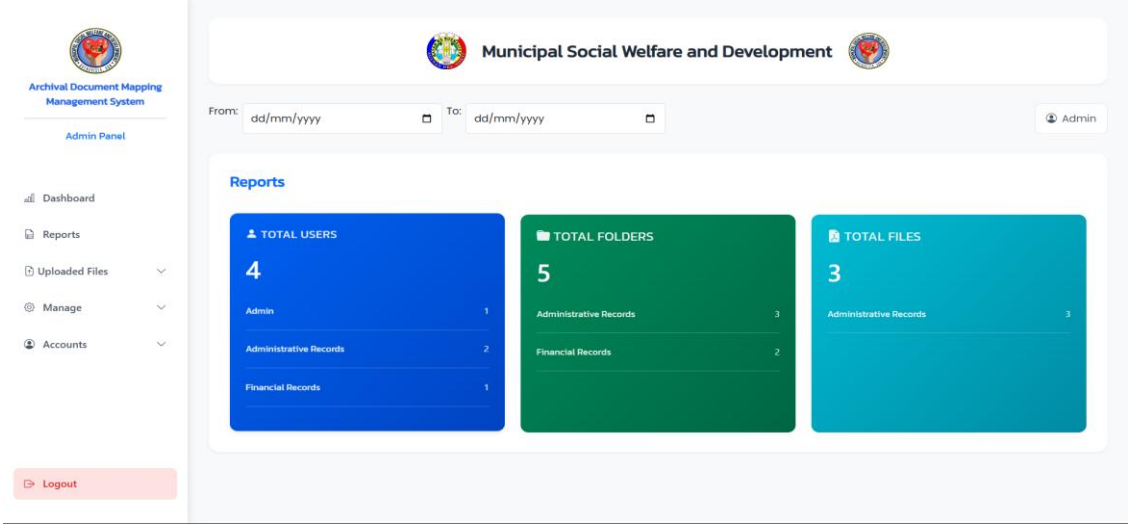


Figure 24. Admin Side – Reports (Admin Interface Design)

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 monitor system usage and supports efficient decision-making for document management.

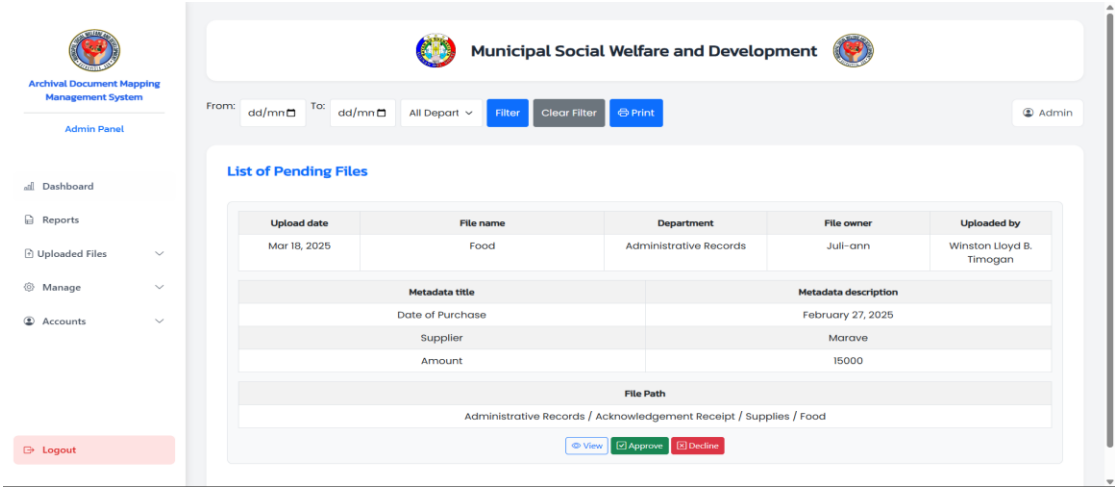


Figure 25. Admin Side – List of Pending Files (Admin Interface Design)

Figure 25 presents the List of Pending Files section on the admin interface, displaying all files 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.

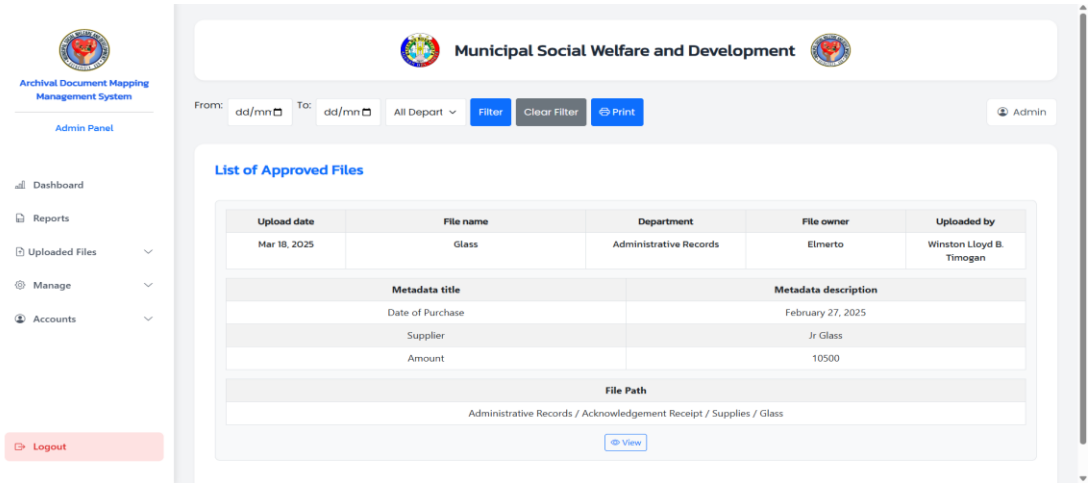


Figure 26. Admin Side – List of Approved Files (Admin Interface Design)

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.

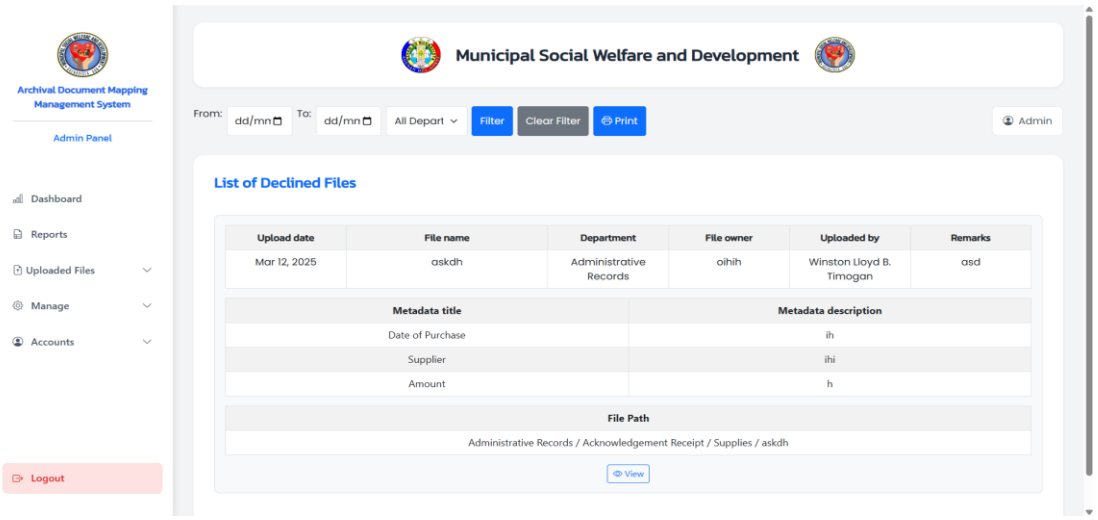


Figure 27. Admin Side – List of Declined Files (Admin Interface Design)

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.

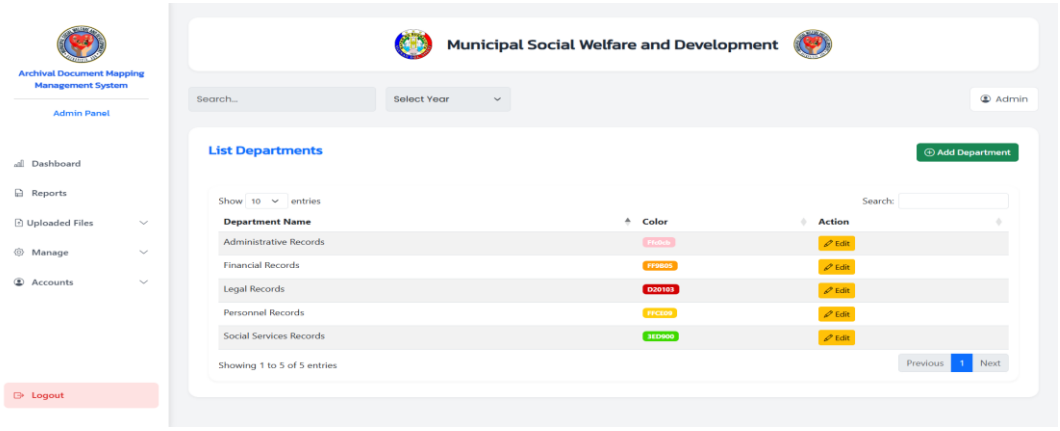


Figure 28. Admin Side – List Departments (Admin Interface Design)

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, ensuring an organized and customizable department management system.

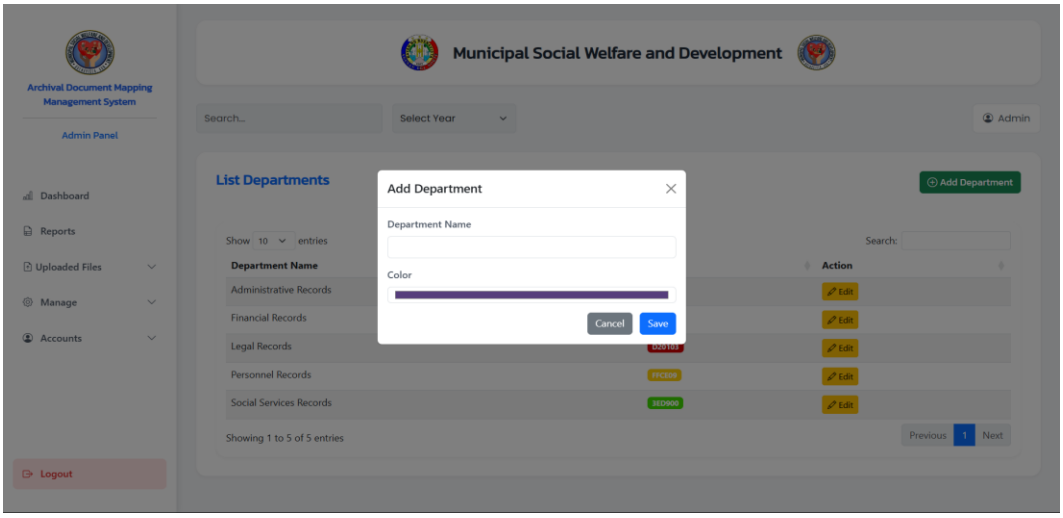


Figure 29. Admin Side – Add Department (Admin Interface Design)

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 names and assigned colors and an action button to save the new department. This feature ensures an organized and customizable department management system.

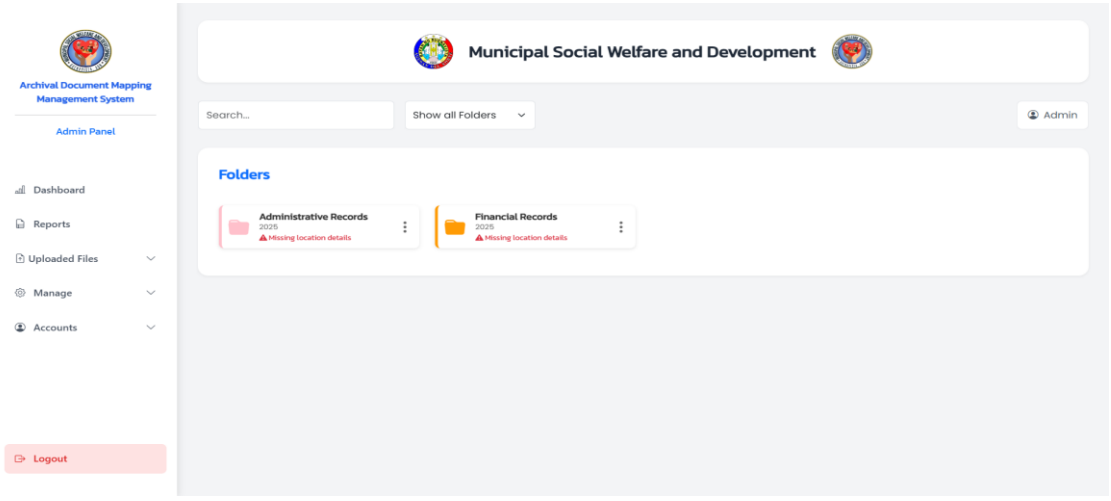


Figure 30. Admin Side – Display Folders (Admin Interface Design)

Figure 30 presents the Display Folders section on the admin interface, showing a structured view of all folders within the system. This section includes folder names, assigned departments, and resources for managing folders. Administrators can efficiently organize, view, and manage folders, ensuring proper document storage and accessibility.

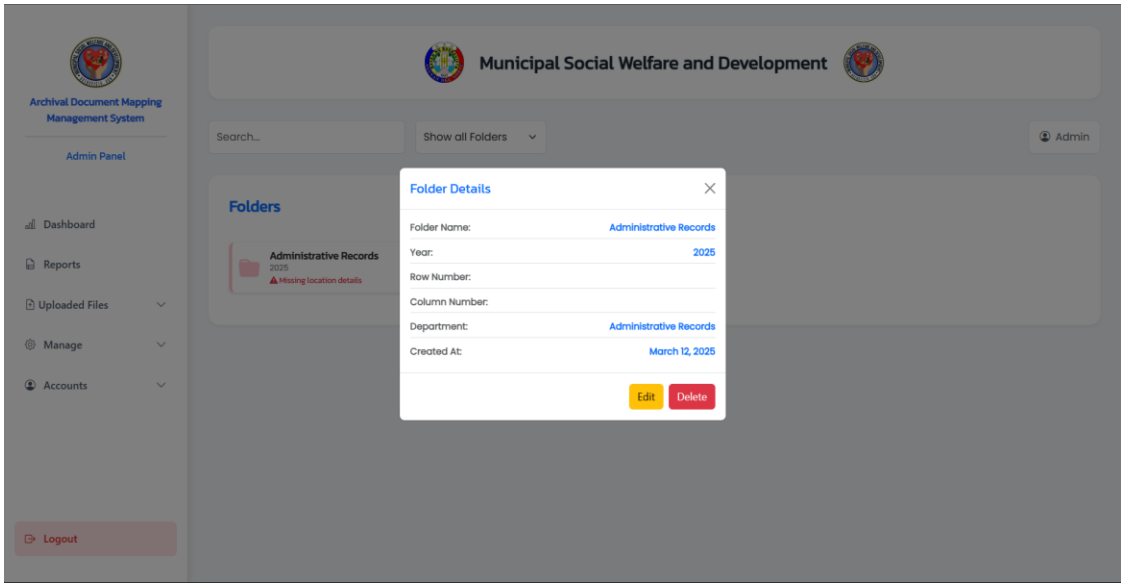


Figure 31. Admin Side – Folder Details (Admin Interface Design)

Figure 31 presents the Folder Details section on the admin interface, displaying specific information about a selected folder. This Section includes 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.

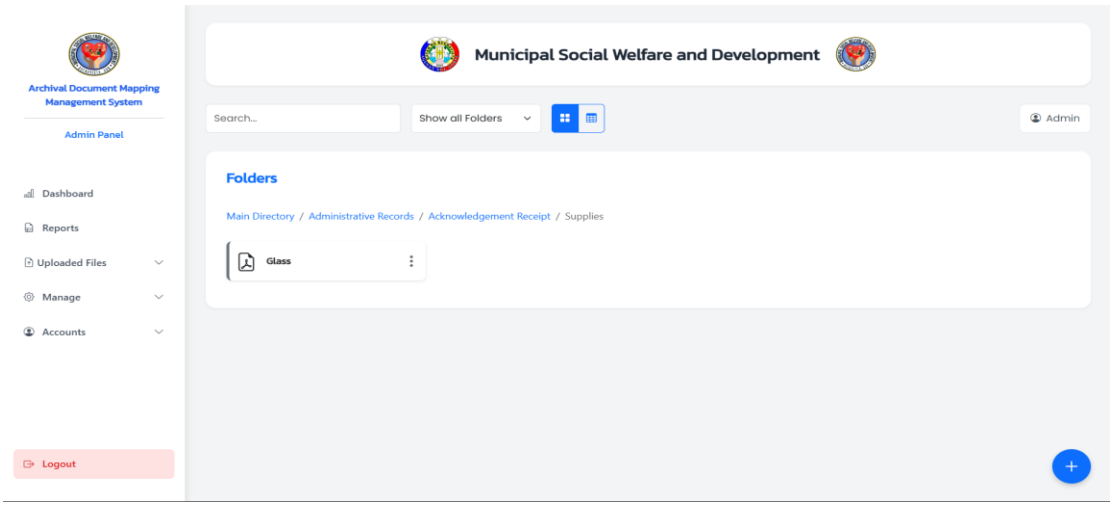


Figure 32. Admin Side – Display Files Card Style (Admin Interface Design)

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.

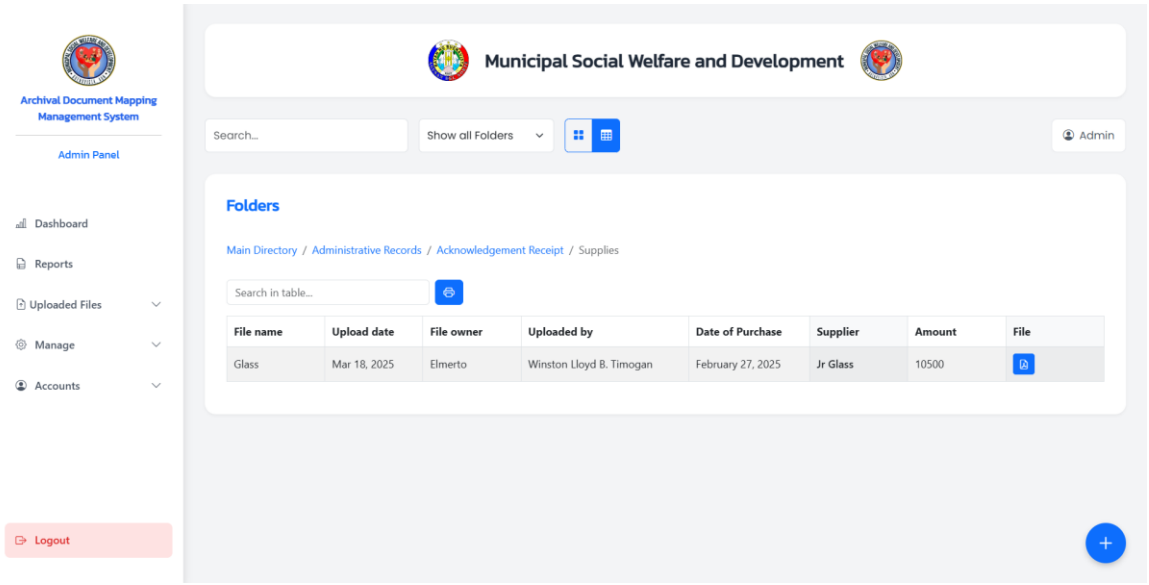


Figure 33. Admin Side – Display Files Table Style (Admin Interface Design)

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.

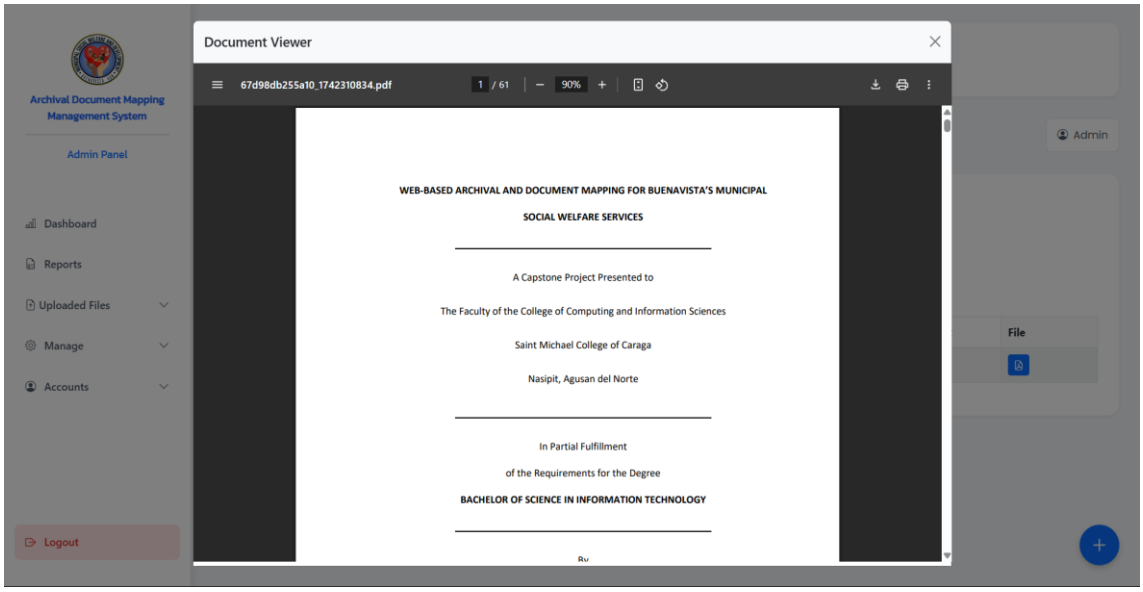


Figure 34. Admin Side – View PDF File (Admin Interface Design)

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.

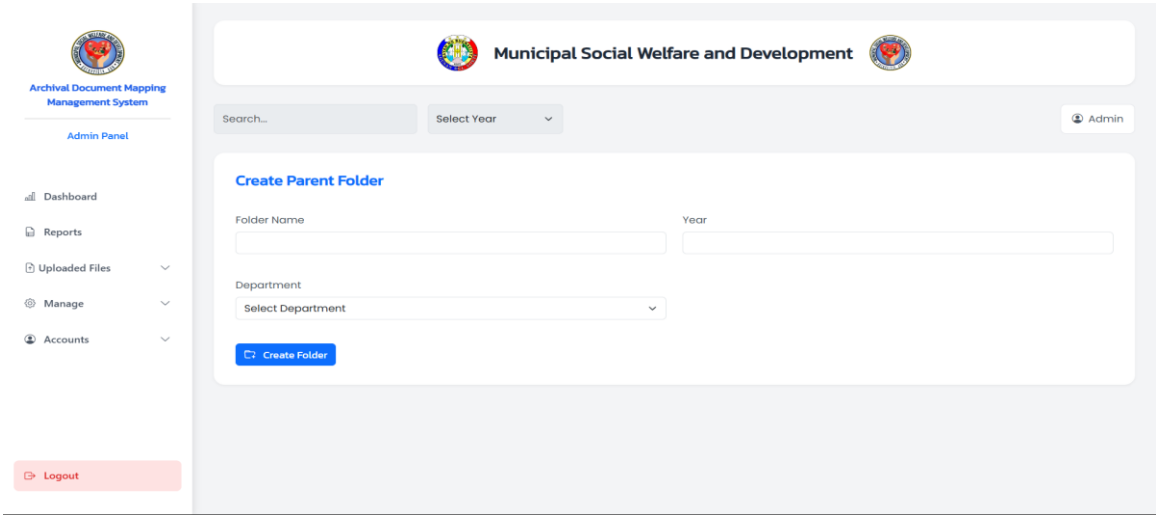


Figure 35. Admin Side – Create Parent Folder (Admin Interface Design)

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.

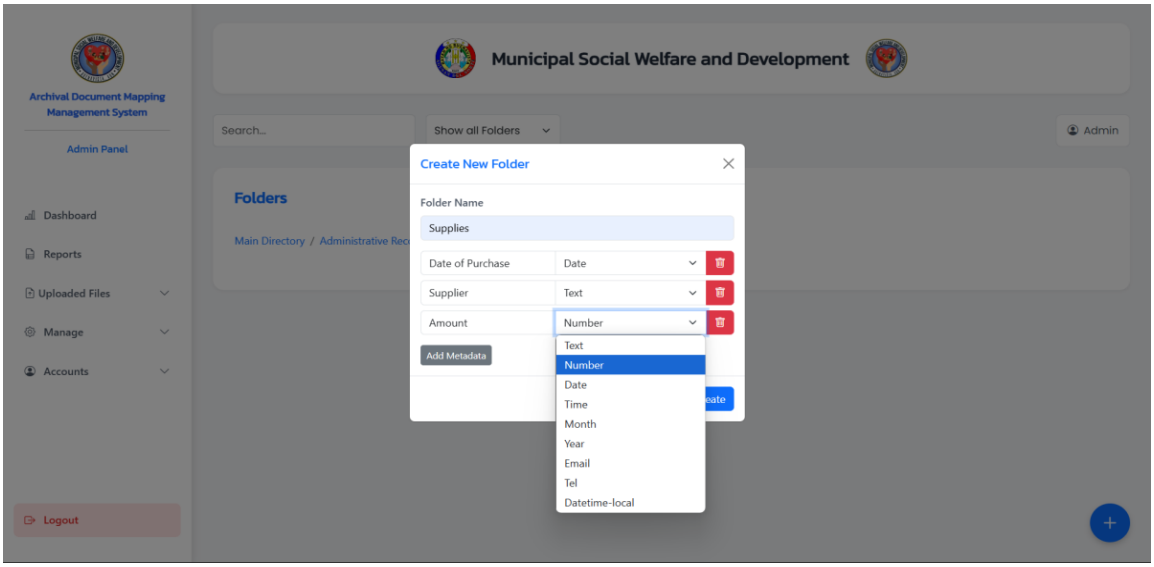


Figure 36. Admin Side – Create Subfolder and Set Metadata Title (Admin Interface Design)

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, and datetime-local, allowing for a more structured and flexible document management system.

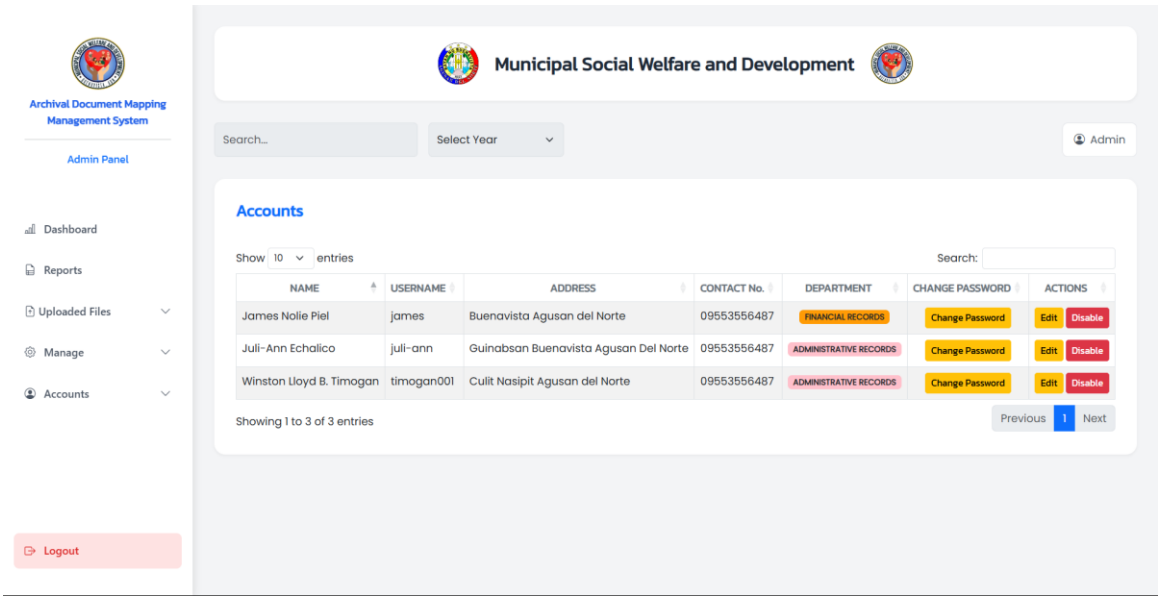


Figure 37. Admin Side – Display All Accounts (Admin Interface Design)

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.

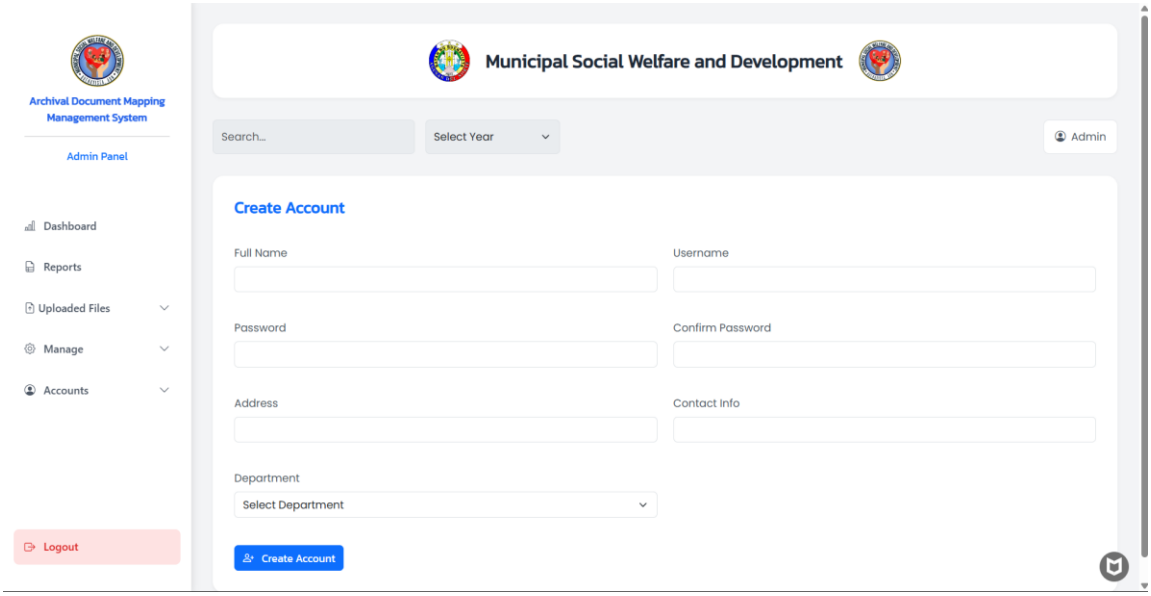


Figure 38. Admin Side – Register Personnel Account (Admin Interface Design)

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 full name, 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. 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.
Server	Apache	XAMPP is a web server that includes Apache and MySQL. It provides an easy way to set up a local web server environment. In ADMMS, XAMPP is used to manage and run the server environment for testing and development.

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.

3.10 Ethical Standard

ETHICAL STANDARD

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 creating 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 are 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 Social Welfare and Development (MSWD) office and identify opportunities for developing a system to support their operations. The study's participants were the personnel and staff members of the MSWD office, as they are the primary users and beneficiaries of the proposed system. The personnel and staff were asked about their current operational challenges and the support they need to improve their workflows. Before the interviews, each participant received a letter of consent that clearly outlined the purpose of the study, the nature of the interview process, and any potential benefits or implications. They were informed that their participation was voluntary, and they provided informed consent before proceeding. The interviews were conducted verbally and focused on identifying system requirements, defining the scope of the proposed solution, discussing common challenges in their current manual processes, and exploring how the system could enhance office operations and service delivery to the

community. This study did not involve any animal or live organism trials. All personal data collected from the MSWD personnel was handled with strict confidentiality and used exclusively for research purposes.

C. Data Privacy and Confidentiality

Implementing Role-Based Access Control (RBAC) protects the data collected in this study. This approach ensures only authorized personnel can access specific files based on their assigned roles. The personnel have access solely to the documents relevant to their responsibilities, and they cannot view or modify files outside their authorization. To maintain strict confidentiality, no anonymous data is shared with anyone. The management of access rights ensures that all data remains secure and protected. Full access to all files is exclusively granted to the admin of the system, who oversees 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 the study was voluntary, specifically for the MSWD personnel involved in interviews and feedback sessions during the development of the web-based system. They were clearly informed that their participation was optional and that they could withdraw from the study without any negative consequences. The research exclusively involved human participants, and all ethical standards were upheld to ensure their rights, privacy, and well-being were fully respected throughout the process.

E. Minimization of Harm and Risk Management

The researchers were dedicated to ensuring the well-being of all MSWD employees by proactively minimizing any potential harm and effectively managing possible risks. All questions, discussions, and interactions were conducted respectfully and thoughtfully, eliminating the possibility of emotional, psychological, or social discomfort. To further protect MSWD employees,

all data collected was treated with the utmost confidentiality. Sensitive information remained secure and anonymous, ensuring participants were not subjected to any negative consequences or risks due to their involvement in the study. The research team remained committed to upholding ethical standards and prioritizing the safety of all MSWD employees throughout the study.

F. Beneficence and Contribution to knowledge

Developing the Web-Based Archival and Document Mapping System for Buenavista's Municipal Social Welfare Services contributes positively to 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. The project offers a practical resource for the academic community for research and learning. At the same time, for society, it ensures improved access to social welfare services, enhanced transparency, and better public trust.

The project findings and system capabilities were 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 archived various documents such as administrative, financial, legal, personnel, and social services records from the local area or community, this study evaluated the employees of Buenavista's Municipal Social Welfare Services. It potentially saved significant time and effort in seeking and maintaining various records. MSWD personnel were selected based on clear and fair inclusion and exclusion criteria, ensuring justice

and fairness in the selection process. All participating employees, regardless of background, were included with appropriate consideration and without discrimination. The selection process was designed to avoid any form of bias. The documents were 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 errors. 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 were clearly reported without any data manipulation, and no potential conflict of interest was declared. Third-party materials were 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 did not utilize any patented materials, as the focus of the study is on archival management and document mapping, which do not require the use of patented technologies. Regarding copyrighted materials, any content that falls under copyright protection, such as previous research or literature, was 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 all living beings' well-being. The study does not indicate any involvement of animals or human trials. It focuses more on developing a digital archival system to improve the municipal welfare department's document management and retrieval efficiency. So, it does not involve human or animal trials that involve ethical considerations.

L. Responsible Use of AI and Other Related Technologies

The researchers also employed Grammarly to enhance the quality of the documentation. ChatGPT is also used in the study to provide ideas on how to enhance the content of the documentation. AI tools are being utilized to meet ethical standards, ensuring that the recommendations they produce align with moral principles and are carefully examined for correctness and applicability. The researcher also reviewed and validated the study to maintain its integrity and professionalism.

M. Ethical Clearance and Institutional Approval

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

CHAPTER IV

SOFTWARE DEVELOPMENT AND TESTING

This chapter covers the Web-Based Archival and Document Mapping System development and testing phases for Buenavista's Municipal Social Welfare Services, detailing the methodologies, frameworks, and tools utilized to build and optimize the system. It outlines the steps 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 #.

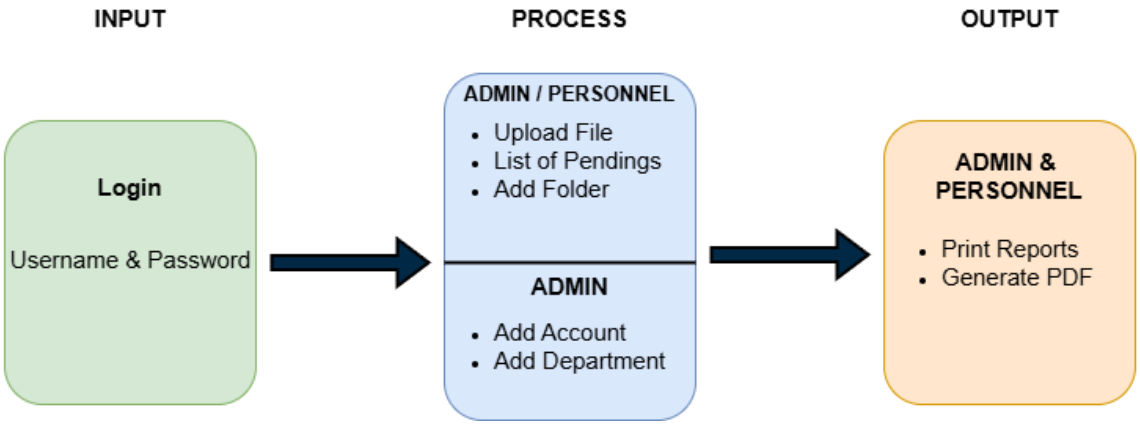


Figure 39. Input-Process-Output Diagram

Figure 39 shows the input-process-output diagram and 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 could 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 beneficiary records, financial assistance, and

case progress reports. 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.

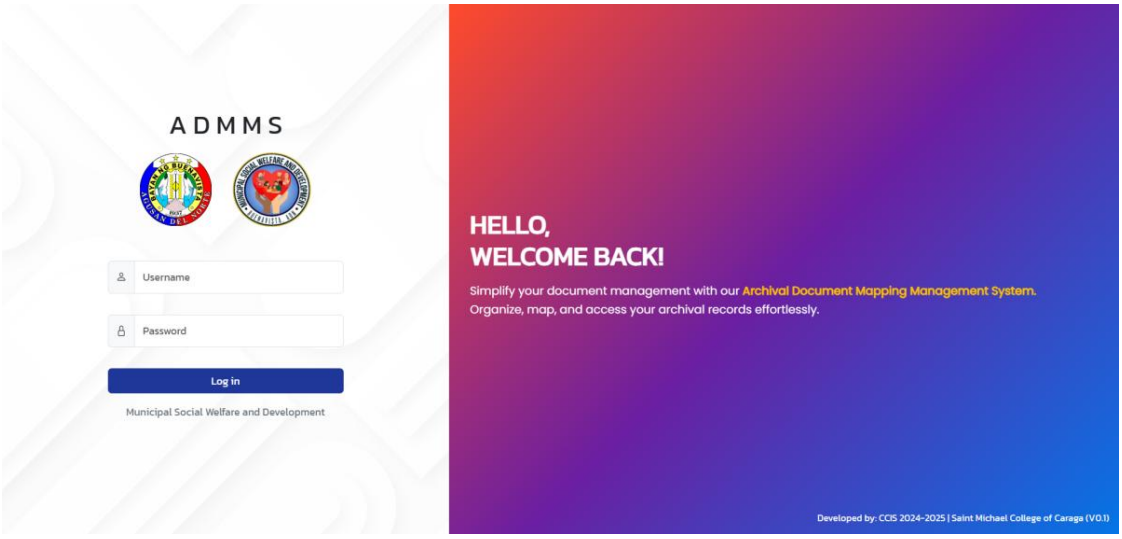


Figure 40. Admin/Personnel Login Page

Figure 40 shows the Admin and Personnel login page. This UI is the first thing they see when accessing the ADMMS. The username and password input is where they enter their account's assigned username and password.

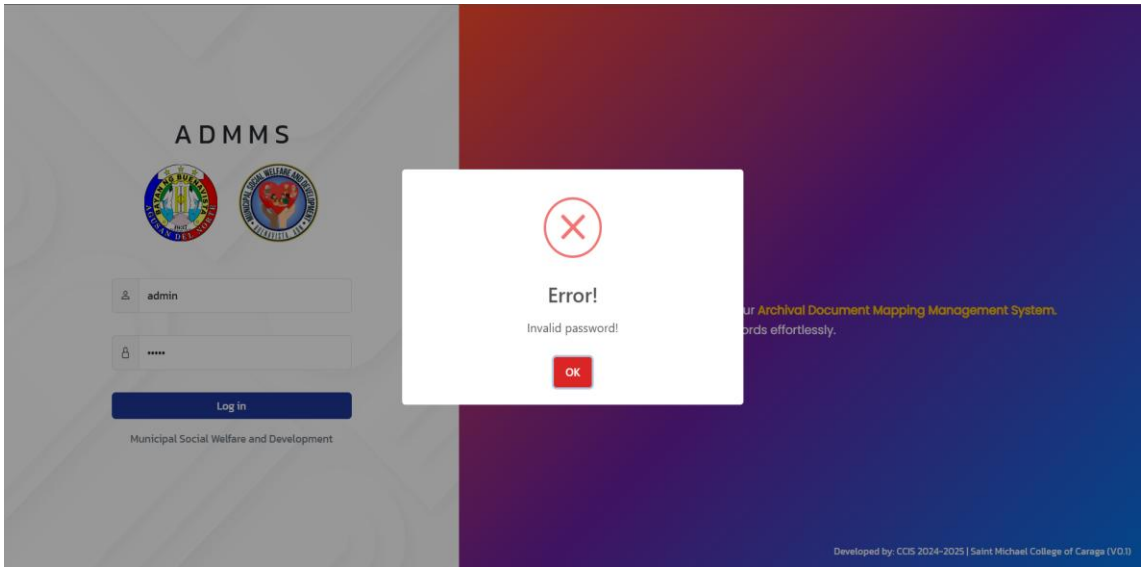


Figure 41 Admin/Personnel – Login Error Notification

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

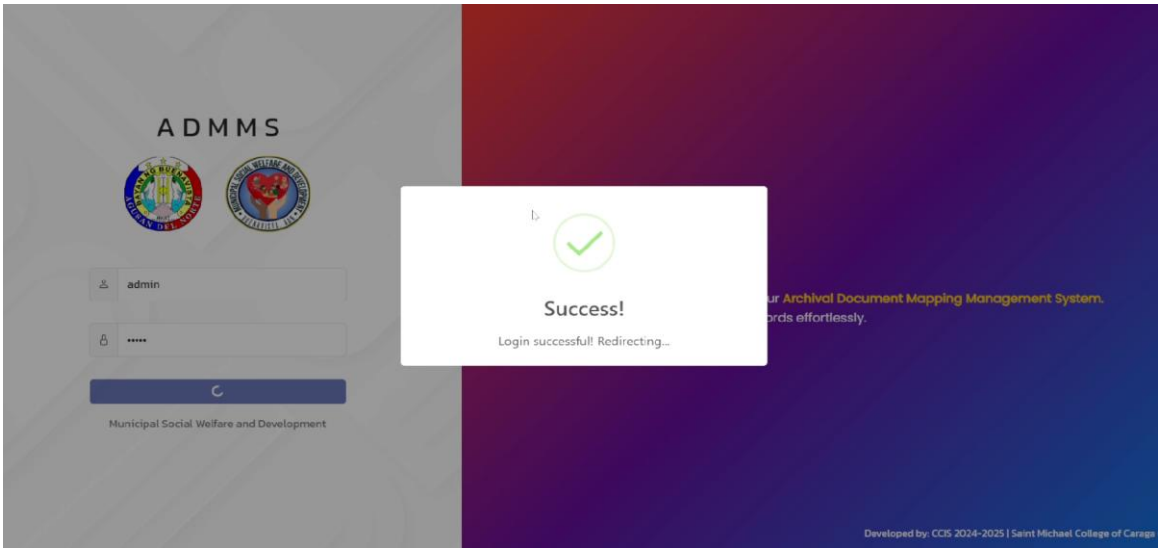


Figure 42. Admin/Personnel – Login Successful Notification

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

```
$(document).ready(function () {
    $('#loginForm').on('submit', function (e) {
        e.preventDefault();

        const button = $(this).find('button[type="submit"]');
        const loginText = button.find('.login-text');
        const spinner = button.find('.spinner-border');

        loginText.addClass('d-none');
        spinner.removeClass('d-none');
        button.prop('disabled', true);

        $.ajax({
            url: 'controllers/login.php',
            type: 'POST',
            data: $(this).serialize(),
            dataType: 'json',
            success: function (response) {
                if (response.success) {
                    Swal.fire({
                        title: 'Success!',
                        text: 'Login successful! Redirecting...',
                        icon: 'success',
                        timer: 1500,
                        showConfirmButton: false
                    }).then(() => {
                        window.location.href = response.redirect;
                    });
                } else {
                    Swal.fire({
                        title: 'Error!',
                        text: response.message || 'Invalid username or password!',
                        icon: 'error',
                        confirmButtonColor: '#dc2626'
                    });
```

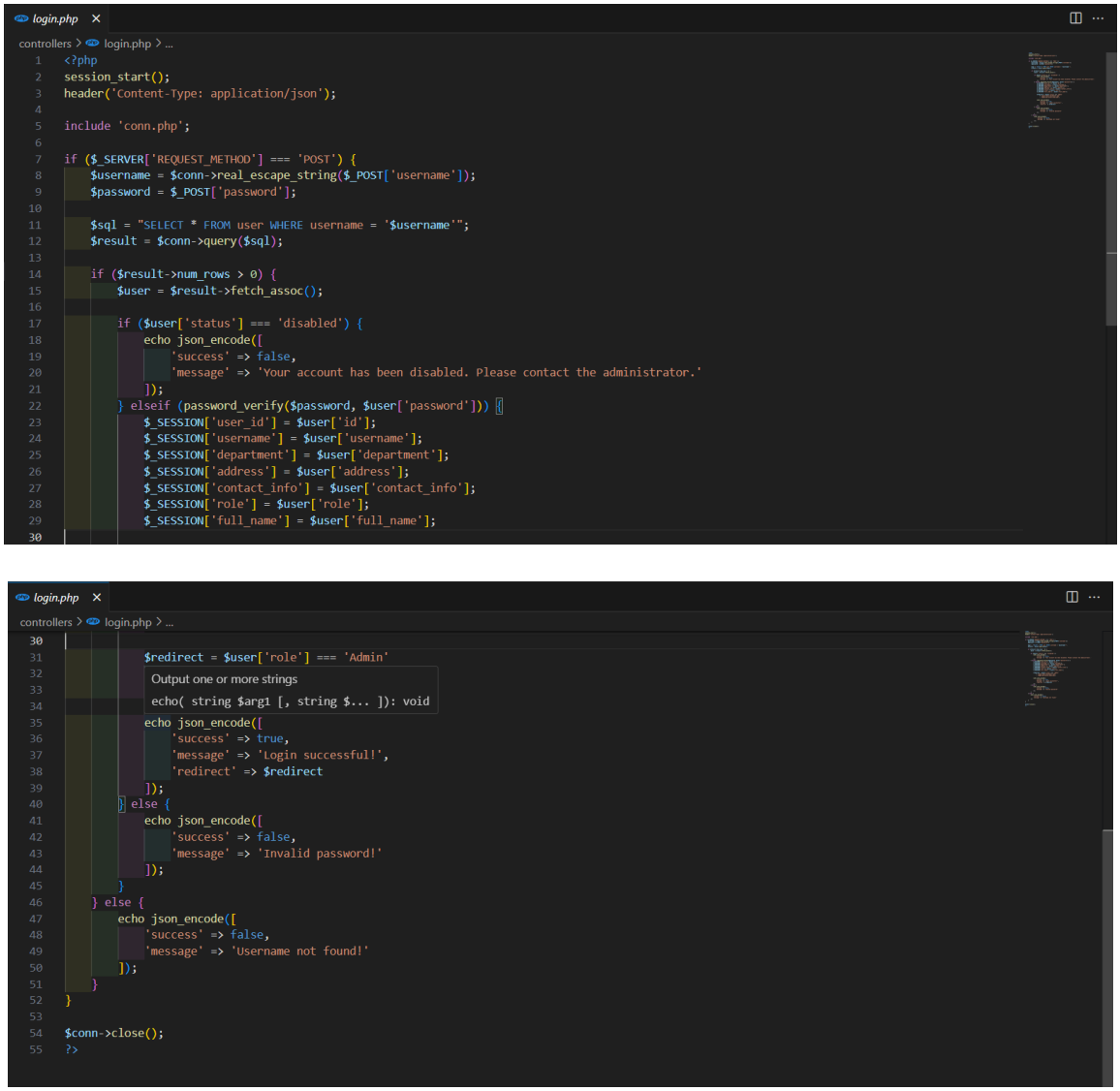
```
                    });
                    loginText.removeClass('d-none');
                    spinner.addClass('d-none');
                    button.prop('disabled', false);
                }
            },
            error: function (xhr) {
                let errorMessage = 'Something went wrong with the server!';
                if (xhr.responseText) {
                    errorMessage += `\n\nDetails: ${xhr.responseText}`;
                }

                Swal.fire({
                    title: 'Error!',
                    text: errorMessage,
                    icon: 'error',
                    confirmButtonColor: '#dc2626'
                });

                loginText.removeClass('d-none');
                spinner.addClass('d-none');
                button.prop('disabled', false);
            }
        });
    });
```

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 shows 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 `the 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.



```php
<?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'];

            $redirect = $user['role'] === 'Admin'
                Output one or more strings
                echo( string $arg1 [, string $... ]): void
            echo json_encode([
                'success' => true,
                'message' => 'Login successful!',
                'redirect' => $redirect
            ]);
        } else {
            echo json_encode([
                'success' => false,
                'message' => 'Invalid password!'
            ]);
        }
    } else {
        echo json_encode([
            'success' => false,
            'message' => 'Username not found!'
        ]);
    }
}

$conn->close();
?>
```

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 are 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.

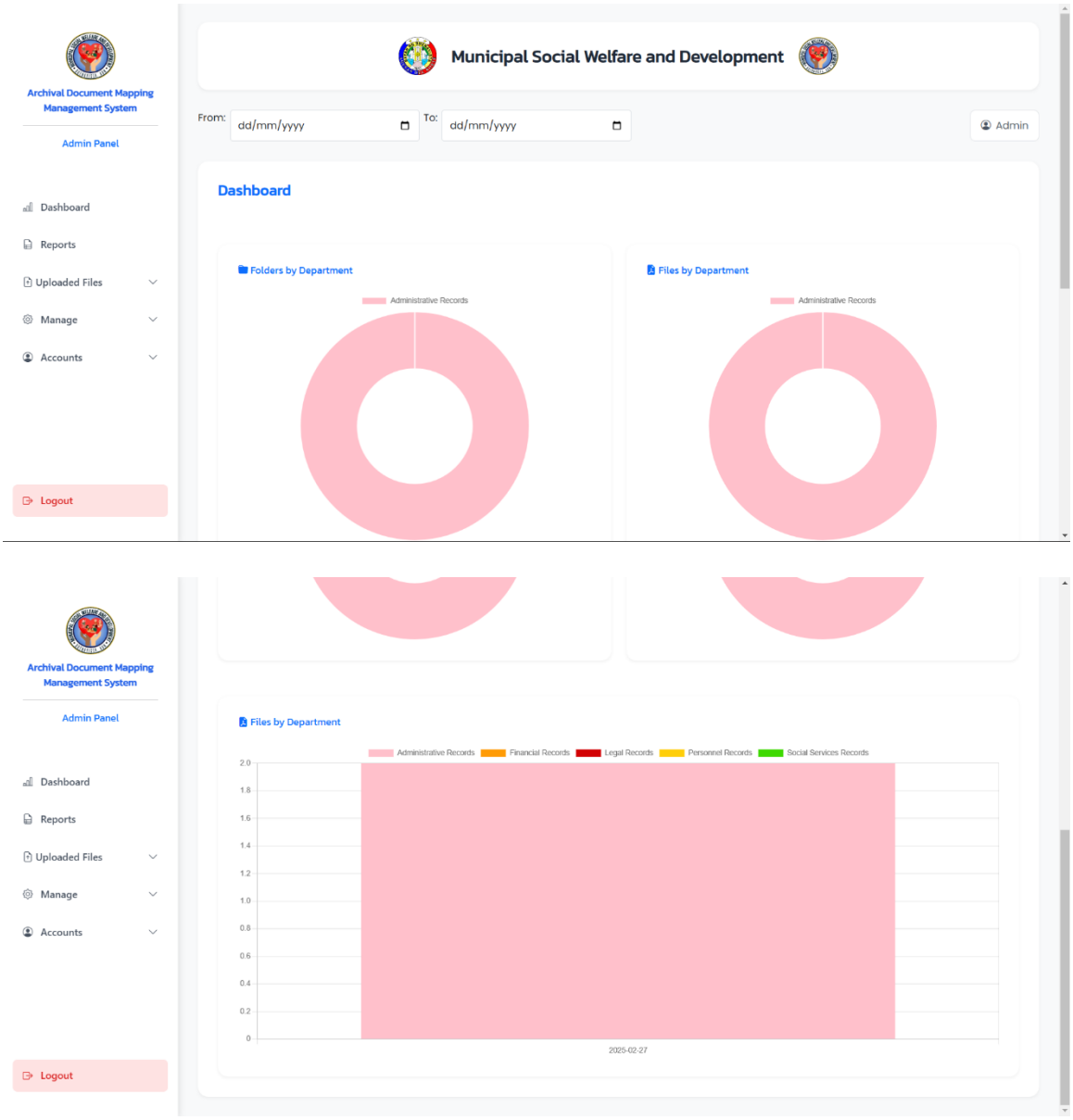


Figure 45. Admin – Dashboard

Figure 45 shows the Admin Panel Dashboard, which 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.

```
document.addEventListener('DOMContentLoaded', function () {

    let charts = {};

    function fetchChartData(fromDate = '', toDate = '') {
        fetch(`../../controllers/chart.php?fromDate=${fromDate}&toDate=${toDate}`)
            .then(response => response.json())
            .then(data => {
                renderDoughnutChart('foldersDoughnutChart', 'Folders by Department', data.folders, data.departments);
                renderDoughnutChart('filesDoughnutChart', 'Files by Department', data.files, data.departments);
                renderStackedBarChart('stackedBarChart', data.stackedFiles, data.departments);
            })
            .catch(error => console.error('Error fetching chart data:', error));
    }

    function destroyChart(chartId) {
        if (charts[chartId]) {
            charts[chartId].destroy();
            charts[chartId] = null;
        }
    }

    function renderDoughnutChart(canvasId, title, dataset, departments) {
        const ctx = document.getElementById(canvasId).getContext('2d');
        destroyChart(canvasId);

        const departmentColors = {};
        departments.forEach(dept => {
            departmentColors[dept.department_name] = `#${dept.folder_color}`;
        });

        const dataCount = {};
        dataset.forEach(item => {
            dataCount[item.department] = (dataCount[item.department] || 0) + 1;
        });

        charts[canvasId] = new Chart(ctx, {
            type: 'doughnut',
            data: {
                labels: Object.keys(dataCount),
                datasets: [{
                    data: Object.values(dataCount),
                    backgroundColor: Object.keys(dataCount).map(dept => departmentColors[dept] || '#ccc')
                }]
            },
            options: { responsive: true, plugins: { legend: { position: 'top' } } }
        });
    }

    function renderStackedBarChart(canvasId, dataset, departments) {
        const ctx = document.getElementById(canvasId).getContext('2d');
        destroyChart(canvasId);

        const departmentColors = {};
        departments.forEach(dept => {
            departmentColors[dept.department_name] = `#${dept.folder_color}`;
        });

        const groupedData = {};
        dataset.forEach(item => {
            if (!groupedData[item.upload_date]) groupedData[item.upload_date] = {};
            groupedData[item.upload_date][item.department] = item.count;
        });

        const labels = Object.keys(groupedData);
        const datasets = departments
            .filter(dept => dept.department_name !== 'Admin')
            .map(dept => ({
                label: dept.department_name,
                backgroundColor: `#${dept.folder_color}`,
                data: labels.map(date => groupedData[date][dept.department_name] || 0)
            }));
```

```
        charts[canvasId] = new Chart(ctx, {
            type: 'bar',
            data: { labels, datasets },
            options: { responsive: true, scales: { x: { stacked: true }, y: { stacked: true } } }
        });
    }

    function updateFilterButtons() {
        const fromDate = document.getElementById('fromDate').value;
        const toDate = document.getElementById('toDate').value;
        const filterButton = document.getElementById('filterButton');
        const clearFilterButton = document.getElementById('clearFilterButton');

        if (fromDate && toDate) {
            filterButton.style.display = 'inline-block';
            clearFilterButton.style.display = 'inline-block';
        } else {
            filterButton.style.display = 'none';
            clearFilterButton.style.display = 'none';
        }
    }
```

```
    document.getElementById('fromDate').addEventListener('change', updateFilterButtons);
    document.getElementById('toDate').addEventListener('change', updateFilterButtons);

    document.getElementById('filterButton').addEventListener('click', function () {
        const fromDate = document.getElementById('fromDate').value;
        const toDate = document.getElementById('toDate').value;
        fetchChartData(fromDate, toDate);
    });

    document.getElementById('clearFilterButton').addEventListener('click', function () {
        document.getElementById('fromDate').value = '';
        document.getElementById('toDate').value = '';
        updateFilterButtons();
        fetchChartData();
    });

    updateFilterButtons();
    fetchChartData();
});
```

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

Figure 46 shows the code for the `fetchChartData(fromDate, toDate)` function, which 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 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 destroyed adequately 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. Chart.js configures the bar chart with stacked bars on the x and y axes, allowing users to see total uploads while distinguishing departmental contributions. This function plays a key role in tracking trends in document submissions and identifying peak activity periods.

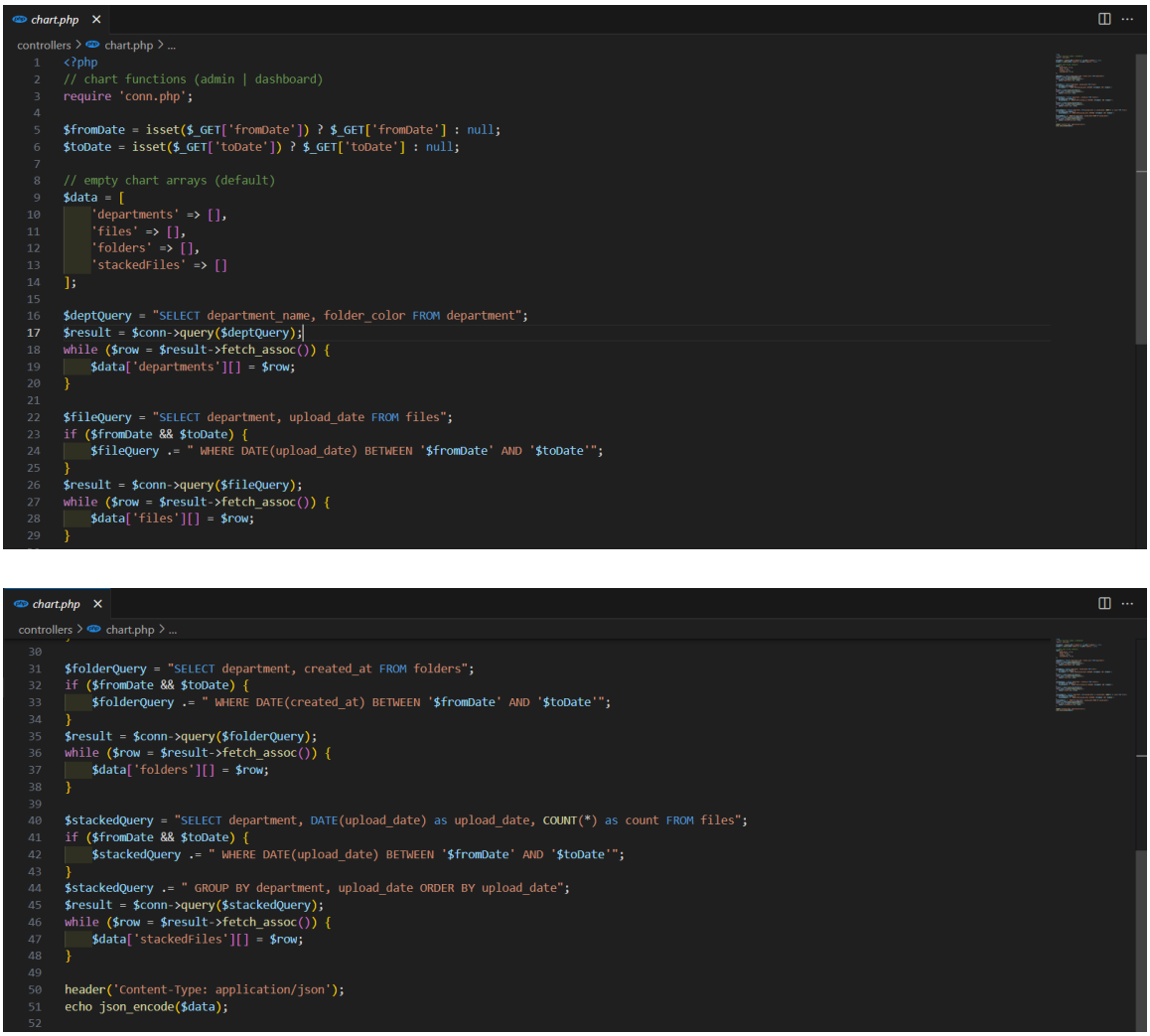
The `updateFilterButtons()` function manages the visibility of the filter buttons `(#filterButton and #clearFilterButton)` based on user input in the date fields. If both `fromDate and toDate` have values, the buttons are displayed; otherwise, they are hidden. This improves the user experience by ensuring 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 the `filter button` clicks, 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 users see relevant statistics immediately. This cohesive structure allows for seamless interaction and an intuitive user experience.

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.



```php
<?php
// chart functions (admin | dashboard)
require 'conn.php';

$fromDate = isset($_GET['fromDate']) ? $_GET['fromDate'] : null;
$toDate = isset($_GET['toDate']) ? $_GET['toDate'] : null;

// empty chart arrays (default)
$data = [
    'departments' => [],
    'files' => [],
    'folders' => [],
    'stackedFiles' => []
];

$deptQuery = "SELECT department_name, folder_color FROM department";
$result = $conn->query($deptQuery);
while ($row = $result->fetch_assoc()) {
    $data['departments'][] = $row;
}

$fileQuery = "SELECT department, upload_date FROM files";
if ($fromDate && $toDate) {
    $fileQuery .= " WHERE DATE(upload_date) BETWEEN '$fromDate' AND '$toDate'";
}
$result = $conn->query($fileQuery);
while ($row = $result->fetch_assoc()) {
    $data['files'][] = $row;
}
```



```php

$folderQuery = "SELECT department, created_at FROM folders";
if ($fromDate && $toDate) {
    $folderQuery .= " WHERE DATE(created_at) BETWEEN '$fromDate' AND '$toDate'";
}
$result = $conn->query($folderQuery);
while ($row = $result->fetch_assoc()) {
    $data['folders'][] = $row;
}

$stackedQuery = "SELECT department, DATE(upload_date) as upload_date, COUNT(*) as count FROM files";
if ($fromDate && $toDate) {
    $stackedQuery .= " WHERE DATE(upload_date) BETWEEN '$fromDate' AND '$toDate'";
}
$stackedQuery .= " GROUP BY department, upload_date ORDER BY upload_date";
$result = $conn->query($stackedQuery);
while ($row = $result->fetch_assoc()) {
    $data['stackedFiles'][] = $row;
}

header('Content-Type: application/json');
echo json_encode($data);
```

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

Figure 48 shows 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.

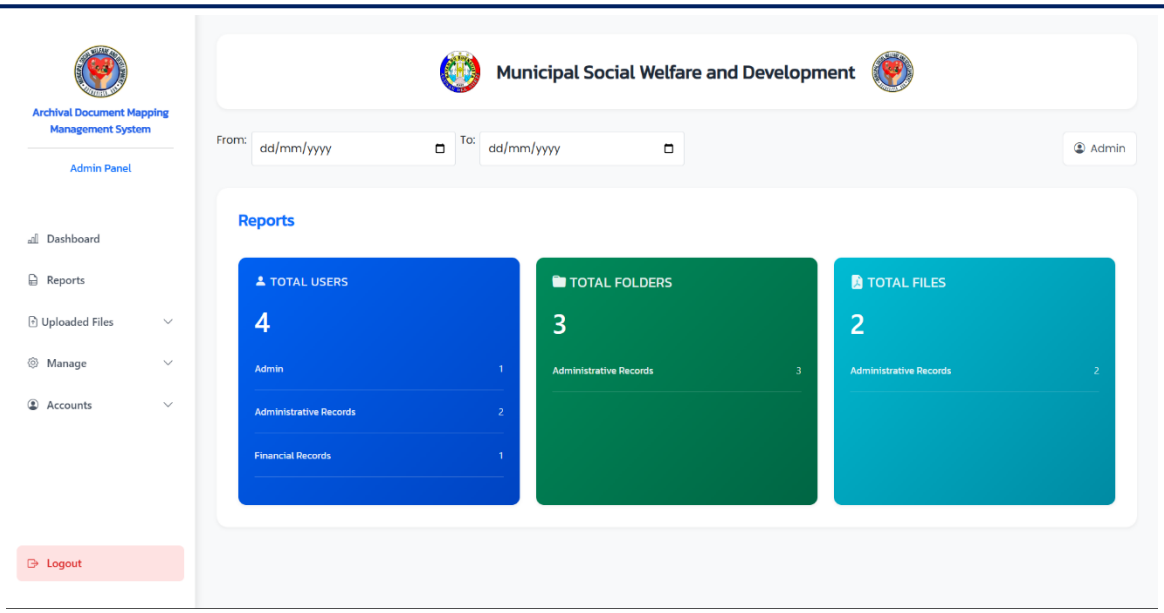


Figure 49. Admin – Reports

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.

```
$(document).ready(function () {
    function getQueryParams() {
        let params = {};
        window.location.search.substring(1).split('&').forEach(function (pair) {
            let [key, value] = pair.split('=');
            params[decodeURIComponent(key)] = decodeURIComponent(value);
        });
        return params;
    }

    function checkDates() {
        if ($('#fromDate').val() && $('#toDate').val()) {
            $('#filterButton').show();
            $('#clearFilterButton').show();
        } else {
            $('#filterButton').hide();
            $('#clearFilterButton').hide();
        }
    }

    let params = getQueryParams();
    if (params.fromDate) {
        $('#fromDate').val(params.fromDate);
    }
    if (params.toDate) {
        $('#toDate').val(params.toDate);
    }

    $('#fromDate, #toDate').on('change', function () {
        checkDates();
    });

    $('#filterButton').on('click', function () {
        let fromDate = $('#fromDate').val();
        let toDate = $('#toDate').val();
        window.location.href = `?fromDate=${fromDate}&toDate=${toDate}`;
    });

```

```
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 `the getQueryParams()` function. It 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 controls 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. It assigns them to the corresponding input fields, ensuring 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 a user-friendly experience for date-based searches.

reports.php

pages > admin > reports.php > ...

```php
<?php
session_start();
if (!isset($_SESSION['user_id'])) {
    header('Location: ../../index.php');
    exit;
}

if ($_SESSION['role'] === 'User') {
    header('Location: ../personnel/home.php');
    exit;
}

require_once '../../controllers/conn.php';

$userStats = [];
$folderStats = [];
$fileStats = [];

$fromDate = isset($_GET['fromDate']) ? $_GET['fromDate'] : null;
$toDate = isset($_GET['toDate']) ? $_GET['toDate'] : null;

$dateCondition = "";
if ($fromDate && $toDate) {
    $fileDate = "AND DATE(upload_date) BETWEEN '$fromDate' AND '$toDate'";
    $createdDate = "AND DATE(created_at) BETWEEN '$fromDate' AND '$toDate'";
} else {
    $fileDate = "";
    $createdDate = "";
}


$filesQuery = "SELECT department, COUNT(*) as count
               FROM files
               WHERE 1=1 $fileDate
               GROUP BY department";
$filesResult = mysqli_query($conn, $filesQuery);
```

```php
$filesResult = mysqli_query($conn, $filesQuery);

$totalFiles = 0;
while ($row = mysqli_fetch_assoc($filesResult)) {
    $fileStats[$row['department']] = $row['count'];
    $totalFiles += $row['count'];
}

$foldersQuery = "SELECT department, COUNT(*) as count
                 FROM folders
                 WHERE 1=1 $createdDate
                 GROUP BY department";
$foldersResult = mysqli_query($conn, $foldersQuery);

$totalFolders = 0;
while ($row = mysqli_fetch_assoc($foldersResult)) {
    $folderStats[$row['department']] = $row['count'];
    $totalFolders += $row['count'];
}

$usersQuery = "SELECT department, COUNT(*) as count
               FROM user
               WHERE 1=1 $createdDate
               GROUP BY department";
$usersResult = mysqli_query($conn, $usersQuery);

$totalUsers = 0;
while ($row = mysqli_fetch_assoc($usersResult)) {
    $userStats[$row['department']] = $row['count'];
    $totalUsers += $row['count'];
}

mysqli_close($conn);
?>
```

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

Figure 51 shows the "`reports.php`" script code, which generates 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 records summary. 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.

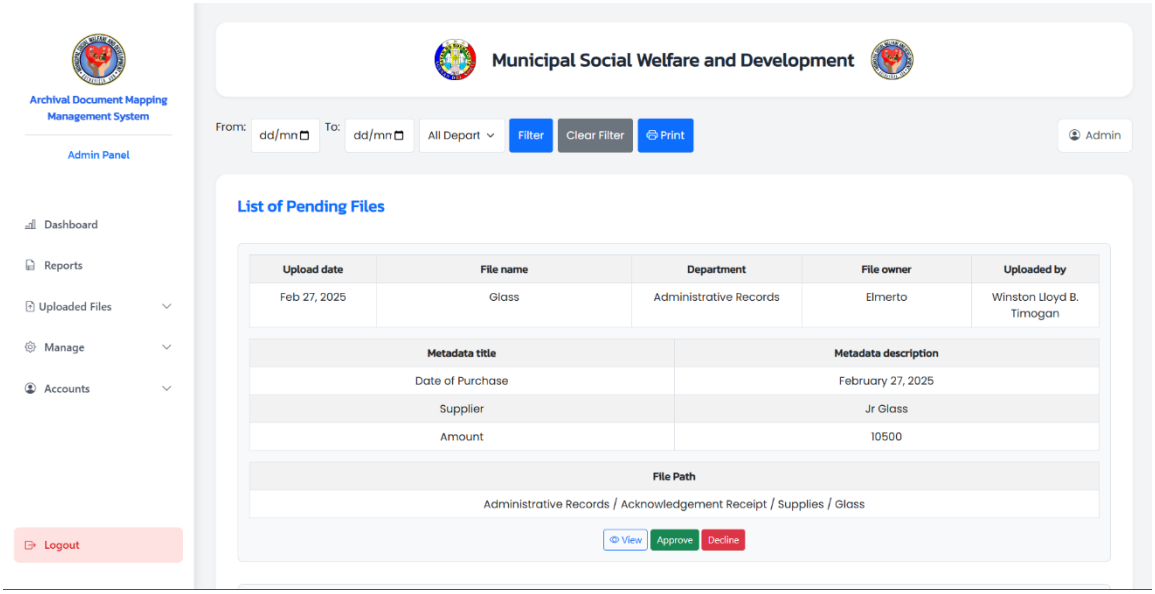


Figure 52. Admin - List of Pending files

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,

name, department, owner, uploader, and associated metadata such as title and description. They can accept or decline the file based on its relevance and compliance.

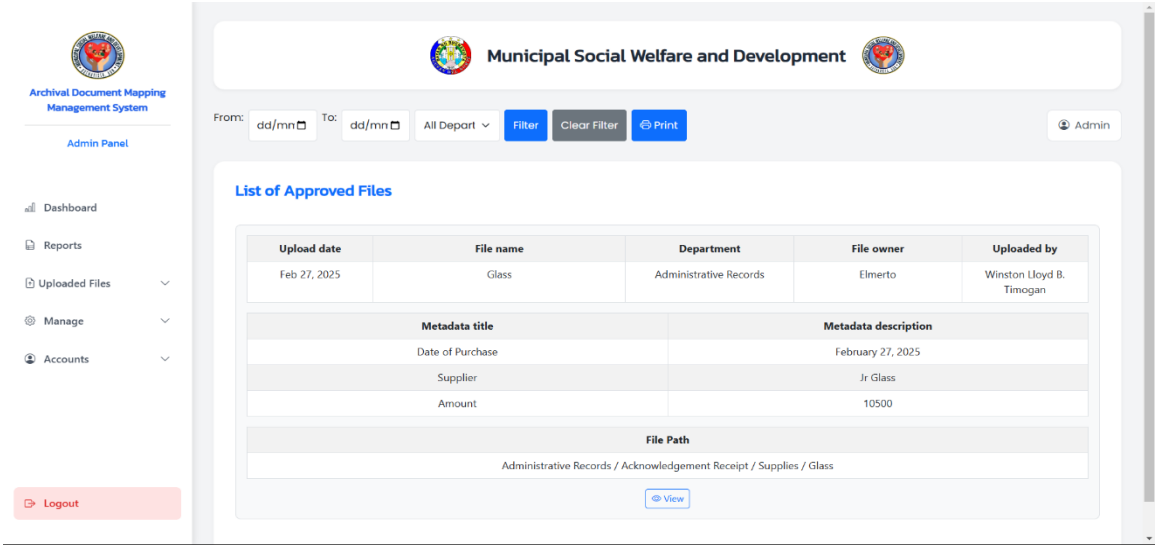


Figure 53. Admin - List of Approve Files

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

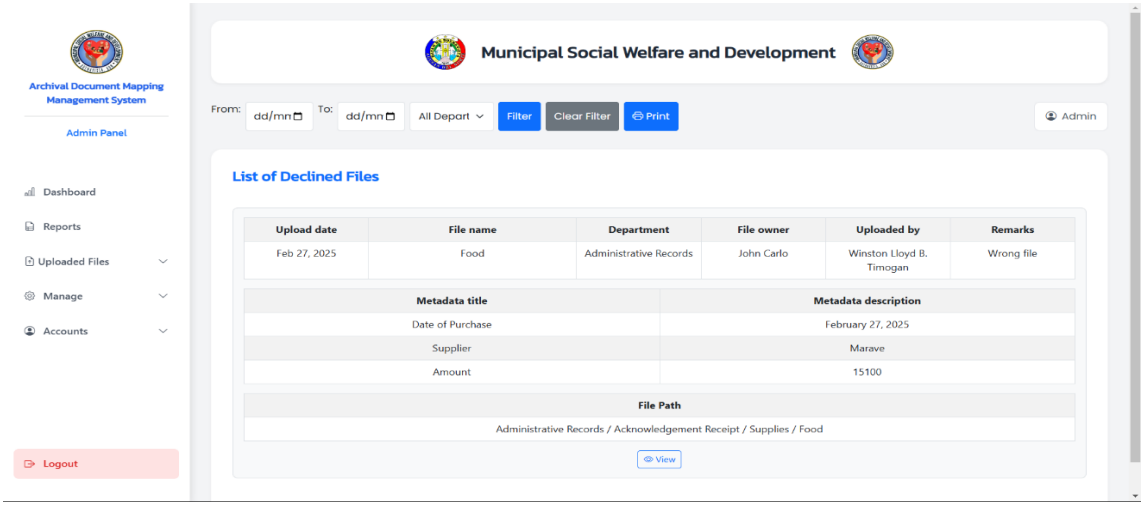


Figure 54. Admin - List of Decline Files

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

controllers > db_functions.php > getFilePath

```php
<?php

function getFilePath($conn, $folder_id)
{
    $path = [];

    while ($folder_id) {
        $folder_query = "SELECT folder_name, parent_folder FROM folders WHERE folder_id = ?";
        $stmt = mysqli_prepare($conn, $folder_query);
        mysqli_stmt_bind_param($stmt, 'i', $folder_id);
        mysqli_stmt_execute($stmt);
        $result = mysqli_stmt_get_result($stmt);

        if ($folder = mysqli_fetch_assoc($result)) {
            array_unshift($path, $folder['folder_name']);
            $folder_id = $folder['parent_folder'];
        } else {
            break;
        }
    }

    return implode(' / ', $path);
}
```

```php
function getMetadataTitles($conn)
{
    $metadata_titles = [];
    $title_query = "SELECT DISTINCT meta_data_title, metadata_id FROM metadata_title";
    $title_result = mysqli_query($conn, $title_query);
    while ($row = mysqli_fetch_assoc($title_result)) {
        $metadata_titles[$row['metadata_id']] = $row['meta_data_title'];
    }
    return $metadata_titles;
}
```

```php
function getFiles($conn, $status, $user_id, $filters = [], $is_admin = false)
{
    $file_query = "
        SELECT f.file_id, f.original_name, f.upload_date, f.author, f.folder_id, f.user_id,
               f.file_name, f.department, f.remarks, u.full_name
        FROM files f
        JOIN user u ON f.user_id = u.id
        WHERE f.status = ?
    ";

    if (!$is_admin) {
        $file_query .= " AND f.user_id = ?";
    }

    $where_clauses = [];
    $params = [$status];

    if (!$is_admin) {
        $params[] = $user_id;
    }

    if (isset($filters['fromDate']) && !empty($filters['fromDate'])) {
        $where_clauses[] = "f.upload_date >= ?";
        $params[] = $filters['fromDate'];
    }

    if (isset($filters['toDate']) && !empty($filters['toDate'])) {
        $where_clauses[] = "f.upload_date <= ?";
        $params[] = $filters['toDate'];
    }
```

```php
    if (isset($filters['department']) && !empty($filters['department'])) {
        $where_clauses[] = "f.department = ?";
        $params[] = $filters['department'];
    }

    if (!empty($where_clauses)) {
        $file_query .= " AND " . implode(" AND ", $where_clauses);
    }

    $stmt = mysqli_prepare($conn, $file_query);
    if (!empty($params)) {
        $types = str_repeat('s', count($params));
        mysqli_stmt_bind_param($stmt, $types, ...$params);
    }
    mysqli_stmt_execute($stmt);
    return mysqli_stmt_get_result($stmt);
}
```

```php
function getFileMetadata($conn, $file_id)
{
    $metadata_query = "
        SELECT mt.meta_data_title, md.meta_data_description
        FROM metadata_description md
        JOIN metadata_title mt ON md.title_id = mt.metadata_id
        WHERE md.file_id = ?
    ";
    $stmt = mysqli_prepare($conn, $metadata_query);
    mysqli_stmt_bind_param($stmt, 's', $file_id);
    mysqli_stmt_execute($stmt);
    return mysqli_stmt_get_result($stmt);
}
```

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

Figure 55 shows the code snippet for `the getFilePath` function, which 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, 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.

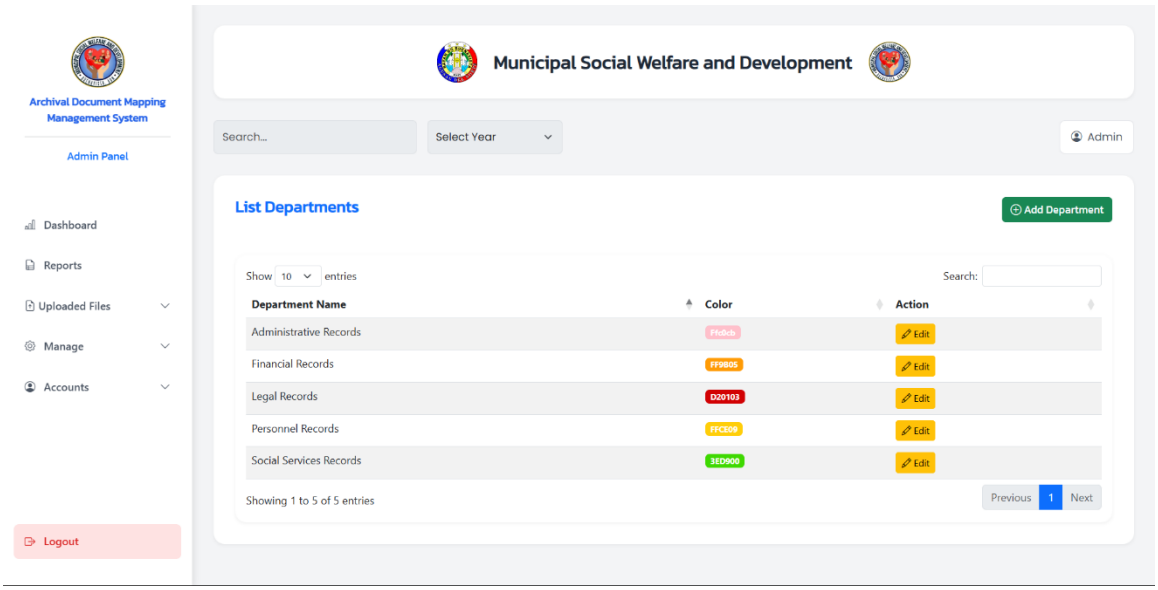


Figure 56. Admin - List Departments

Figure 56 shows the system's 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.

```php
<?php
require_once '../../controllers/conn.php';
$query = "SELECT * FROM department WHERE department_name != 'Admin'";
$result = $conn->query($query);

if ($result && $result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo '<tr>';
        echo '<td>' . htmlspecialchars($row['department_name']) . '</td>';
        echo '<td><span class="badge" style="background-color: #' . htmlspecialchars($row['folder_color']) . '">' .
            htmlspecialchars($row['folder_color']) . '</span></td>';
        echo '<td>
            <button class="btn btn-warning btn-sm edit-btn"
                    data-bs-toggle="modal"
                    data-bs-target="#editDepartmentModal"
                    data-id="' . htmlspecialchars($row['id']) . '"
                    data-name="' . htmlspecialchars($row['department_name']) . '"
                    data-color="#' . htmlspecialchars($row['folder_color']) . '">
                <i class="bi bi-pencil"></i> Edit
            </button>
        </td>';
        echo '</tr>';
    }
}
?>
```

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

Figure 57 shows the code snippet for the PHP script that 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. A while loop iterates through each department record if the query returns rows. 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.

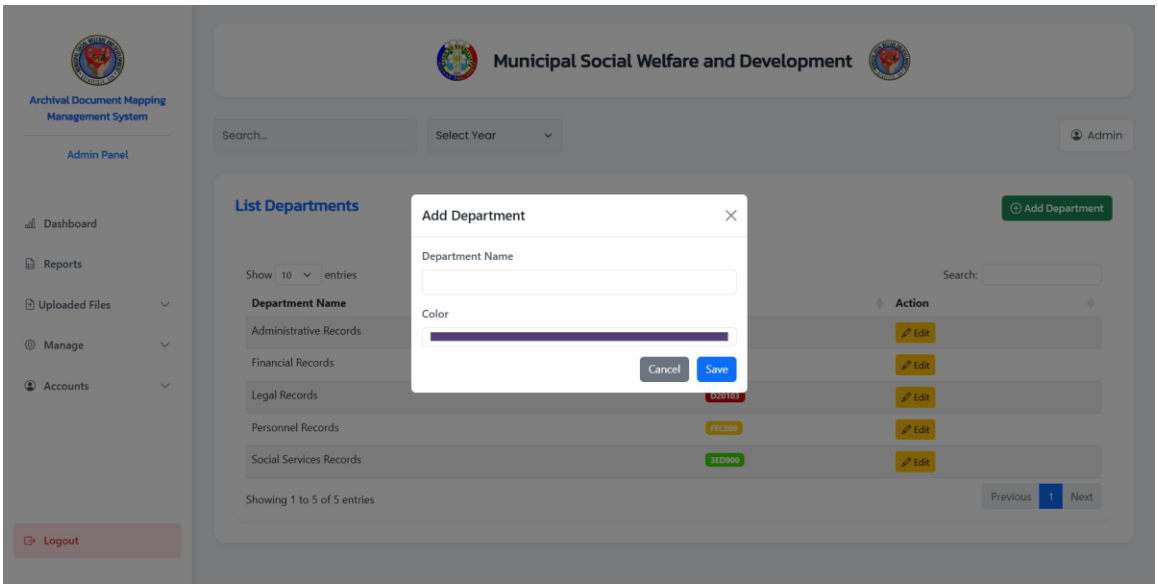


Figure 58. Admin - Add Department

Figure 58 shows how to add 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.

```
$(document).ready(function () {
    $('#departmentsTable').DataTable();

    $('.edit-btn').on('click', function () {
        let id = $(this).data('id');
        let name = $(this).data('name');
        let color = $(this).data('color');

        $('#editDepartmentId').val(id);
        $('#editDepartmentName').val(name);
        $('#editDepartmentColor').val(color);
    });

    $('#addDepartmentForm').on('submit', function (e) {
        e.preventDefault();

        const modal = bootstrap.Modal.getInstance($('#addDepartmentModal'));

```

```
    $.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 `$(document).ready(function () {...})` function initializes various event handlers and functionalities as soon as the page loads, ensuring seamless user interactions with the department management interface. It initializes the `DataTable` plugin on the #departmentsTable element, enabling sorting, searching, and pagination for better data organization and navigation. Next, the `$('.edit-btn').on('click,' function () {...})` function sets up an event listener for edit buttons in the department list. When clicking an edit button, it extracts the department's ID, name, and color from the button's data attributes. It 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. It 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              error: function (xhr, status, error) {
94                  modal.hide();
95
96                  console.error(xhr, status, error);
97                  Swal.fire({
98                      icon: 'error',
99                      title: 'Error',
100                     text: 'An error occurred while processing your request'
101                 });
102             }
103         });
104     });
105
106     $('#addDepartmentModal, #editDepartmentModal').on('hidden.bs.modal', function () {
107         $(this).find('form').trigger('reset');
108     });
109 });
```

Figure 60. Code snippet for department – edit department

Figure 60 shows the code snippet for `$('#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 completing 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 of an issue. 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.

```
            }

            $stmt->bind_param("ss", $department_name, $folder_color);
            if (!$stmt->execute()) {
                throw new Exception('Database execution error: ' . $stmt->error);
            }

            echo json_encode([
                'success' => true,
                'message' => 'Department added successfully'
            ]);

        } catch (Exception $e) {
            echo json_encode([
                'success' => false,
                'message' => $e->getMessage()
            ]);
        } finally {
            if (isset($stmt))
                $stmt->close();
            if (isset($check_stmt))
                $check_stmt->close();
        }
    } else {
        echo json_encode([
            'success' => false,
            'message' => 'Invalid action!'
        ]);
    }
```

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

Figure 61 shows the code for `the add_department` function in this script, which handles adding new departments while ensuring administrator role-based access control. 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 and 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 final 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 avoid SQL injection while maintaining a structured and error-resilient execution flow.

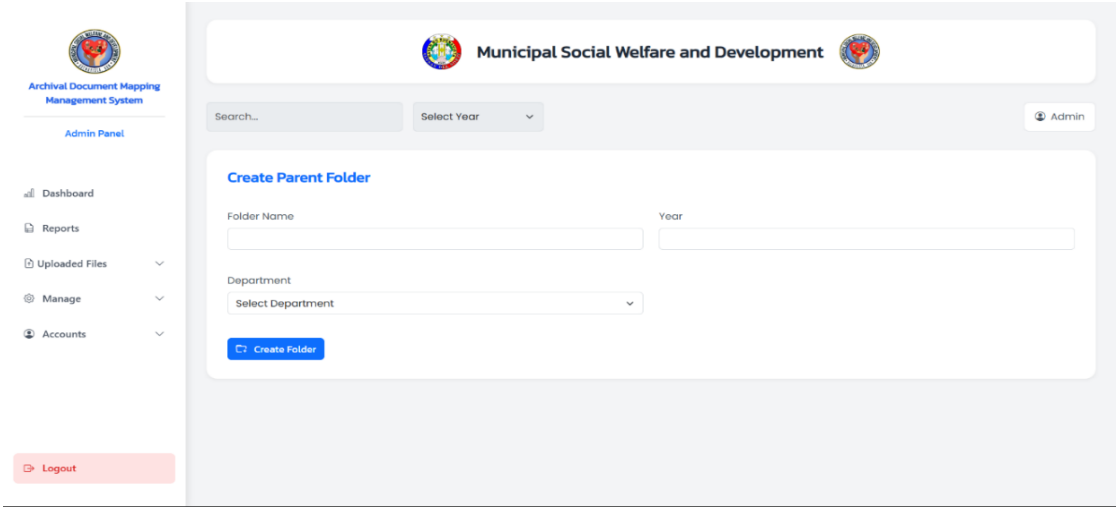


Figure 62. Admin - Create Parent folder

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

```php
} elseif ($action === 'create_folder') {
    // for creating parent folder (admin | add folder)
    date_default_timezone_set('Asia/Manila');
    $currentDate = date('Y-m-d');

    $folderName = $_POST['folderName'];
    $year = $_POST['year'];
    $department = $_POST['department'];

    $stmt = $conn->prepare("INSERT INTO folders (folder_name, year, department, created_at) VALUES (?, ?, ?, ?)");
    $stmt->bind_param("siss", $folderName, $year, $department, $currentDate);

    if ($stmt->execute()) {
        echo json_encode([
            'success' => true,
            'message' => 'Folder created successfully!'
        ]);
    } else {
        echo json_encode([
            'success' => false,
            'message' => "Error: " . $stmt->error
        ]);
    }

    $stmt->close();
```

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

Figure 63 shows the code for the 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 accurate timestamps. 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.

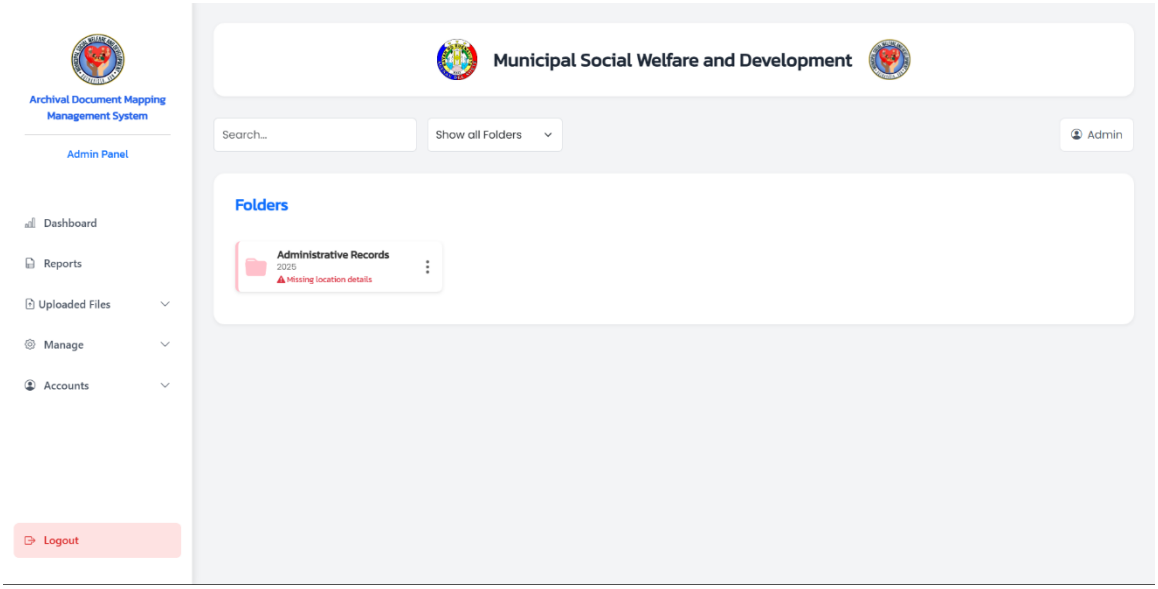
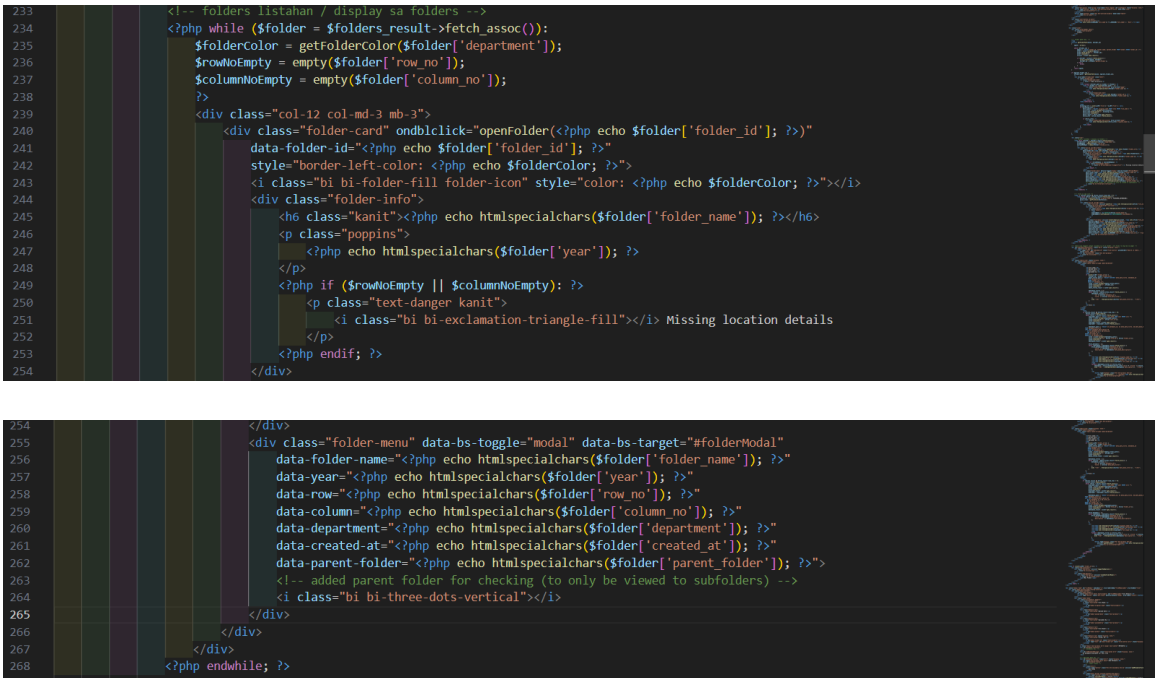


Figure 64. Admin – Display Folders

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 organize, access, and oversee folders, ensuring efficient document storage and retrieval.



```
<!-- folders listahan / display sa folders -->
<?php while ($folder = $folders_result->fetch_assoc()):
    $folderColor = getFolderColor($folder['department']);
    $rowNoEmpty = empty($folder['row_no']);
    $columnNoEmpty = empty($folder['column_no']);
    ?>
    <div class="col-12 col-md-3 mb-3">
        <div class="folder-card" ondblclick="openFolder(<?php echo $folder['folder_id']; ?>)"
            data-folder-id="<?php echo $folder['folder_id']; ?>"
            style="border-left-color: <?php echo $folderColor; ?>">
            <i class="bi bi-folder-fill folder-icon" style="color: <?php echo $folderColor; ?>"></i>
            <div class="folder-info">
                <h6 class="kanit"><?php echo htmlspecialchars($folder['folder_name']); ?></h6>
                <p class="poppins">
                    <?php echo htmlspecialchars($folder['year']); ?>
                </p>
                <?php if ($rowNoEmpty || $columnNoEmpty): ?>
                    <p class="text-danger kanit">
                        <i class="bi bi-exclamation-triangle-fill"></i> Missing location details
                    </p>
                <?php endif; ?>
            </div>
            </div>
            <div class="folder-menu" data-bs-toggle="modal" data-bs-target="#folderModal"
                data-folder-name="<?php echo htmlspecialchars($folder['folder_name']); ?>"
                data-year="<?php echo htmlspecialchars($folder['year']); ?>"
                data-row="<?php echo htmlspecialchars($folder['row_no']); ?>"
                data-column="<?php echo htmlspecialchars($folder['column_no']); ?>"
                data-department="<?php echo htmlspecialchars($folder['department']); ?>"
                data-created-at="<?php echo htmlspecialchars($folder['created_at']); ?>"
                data-parent-folder="<?php echo htmlspecialchars($folder['parent_folder']); ?>">
                <!-- added parent folder for checking (to only be viewed to subfolders) -->
                <i class="bi bi-three-dots-vertical"></i>
            </div>
        </div>
    </div>
<?php endwhile; ?>
```

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

Figure 65 shows the code 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.

```php
if ($user_department === 'admin') {
    // for admin side folders
    if ($parent_folder_id) {
        $query = "SELECT * FROM folders WHERE parent_folder = ?";
        $stmt = $conn->prepare($query);
        $stmt->bind_param("i", $parent_folder_id);
        $stmt->execute();
        $folders_result = $stmt->get_result();

        $files_query = "SELECT * FROM files WHERE folder_id = ? AND status = 'approved'";
        $files_stmt = $conn->prepare($files_query);
        $files_stmt->bind_param("i", $parent_folder_id);
        $files_stmt->execute();
        $files_result = $files_stmt->get_result();

        $parent_query = "SELECT folder_name FROM folders WHERE folder_id = ?";
        $parent_stmt = $conn->prepare($parent_query);
        $parent_stmt->bind_param("i", $parent_folder_id);
        $parent_stmt->execute();
        $parent_result = $parent_stmt->get_result();
        $parent_folder = $parent_result->fetch_assoc();
    } else {
        $query = "SELECT * FROM folders WHERE (parent_folder IS NULL OR parent_folder = 0)";
        $stmt = $conn->prepare($query);
        $stmt->execute();
        $folders_result = $stmt->get_result();

        $files_result = false;
    }

    $years_query = "SELECT DISTINCT year FROM folders ORDER BY year DESC";
    $years_stmt = $conn->prepare($years_query);
    $years_stmt->execute();
    $years_result = $years_stmt->get_result();
} else {
```

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

Figure 66 shows the PHP script code 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. It also retrieves files associated with the parent folder with 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 NULL or 0, indicating that these are root-level directories. For non-admin users, the script simply retrieves a 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.

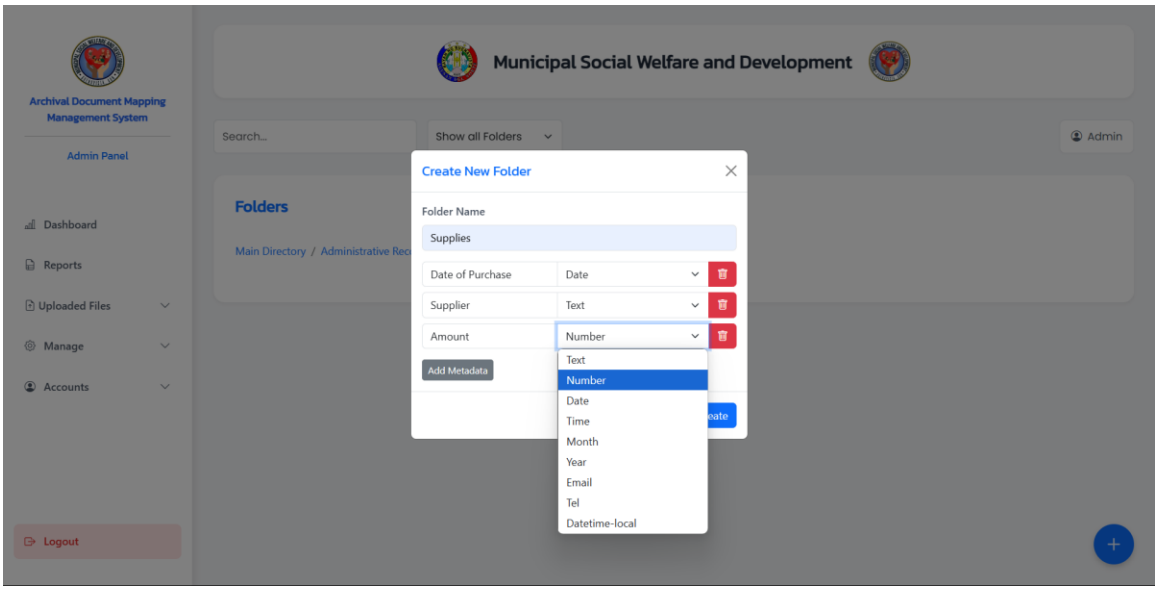


Figure 67. Admin – Create SubFolder

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. Users can specify the input type when adding metadata, 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.

```
function addMetadataTitleField() {
    const metadataTitlesList = document.getElementById('metadata-titles-list');
    const metadataItem = document.createElement('div');
    metadataItem.className = 'metadata-item d-flex justify-content-between align-items-center mb-2';

    const options = ['number', 'text', 'date', 'time', 'month', 'year', 'email', 'tel', 'datetime-local'];

    let selectOptions = options.map(opt => `<option value="${opt}">${opt.charAt(0).toUpperCase() + opt.slice(1)}</option>`).join('');

    metadataItem.innerHTML = `
        <input type="text" class="form-control form-control-sm me-2" id="new-metadata-title" placeholder="Enter metadata title">
        <select class="form-select form-select-sm me-2" id="new-input-type">
            ${selectOptions}
        </select>
        <div>
            <button type="button" class="btn btn-sm btn-outline-success" onclick="saveNewMetadataTitle()">
                <i class="bi bi-check"></i>
            </button>
            <button type="button" class="btn btn-sm btn-outline-danger" onclick="cancelNewMetadataTitle()">
                <i class="bi bi-x"></i>
            </button>
        </div>
    `;

    metadataTitlesList.appendChild(metadataItem);
}
```

```
function saveNewMetadataTitle() {
    const newMetadataTitle = document.getElementById('new-metadata-title').value;
    const newInputType = document.getElementById('new-input-type').value;

    if (newMetadataTitle.trim() === '') {
        Swal.fire({
            title: 'Error!',
            text: 'Metadata title cannot be empty',
            icon: 'error',
            confirmButtonText: 'Ok'
        });
        return;
    }

    fetch('../../controllers/create.php', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({
            action: 'add_metadata_title',
            folder_id: currentEditFolderId,
            meta_data_title: newMetadataTitle,
            input_type: newInputType
        })
    })
        .then(response => response.json())
        .then(data => {
            if (data.success) {
                fetchMetadataTitles(currentEditFolderId);
                Swal.fire({
                    title: 'Success!',
                    text: 'Metadata title added successfully',
                    icon: 'success',
                    timer: 1500,
                    showConfirmButton: false
                });
            } else {
                Swal.fire({
                    title: 'Error!',
                    text: data.message || 'Unknown error',
                    icon: 'error',
                    confirmButtonText: 'Ok'
                });
            }
        })
        .catch(error => {
            console.error('Error:', error);
            Swal.fire({
                title: 'Error!',
                text: error.message || 'An unexpected error occurred',
                icon: 'error',
                confirmButtonText: 'Ok'
            });
        });
}
```

Figure 68. Set metadata and input type – create subfolder

Figure 68 shows the code for the `addMetadataTitleField()` function, which 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.

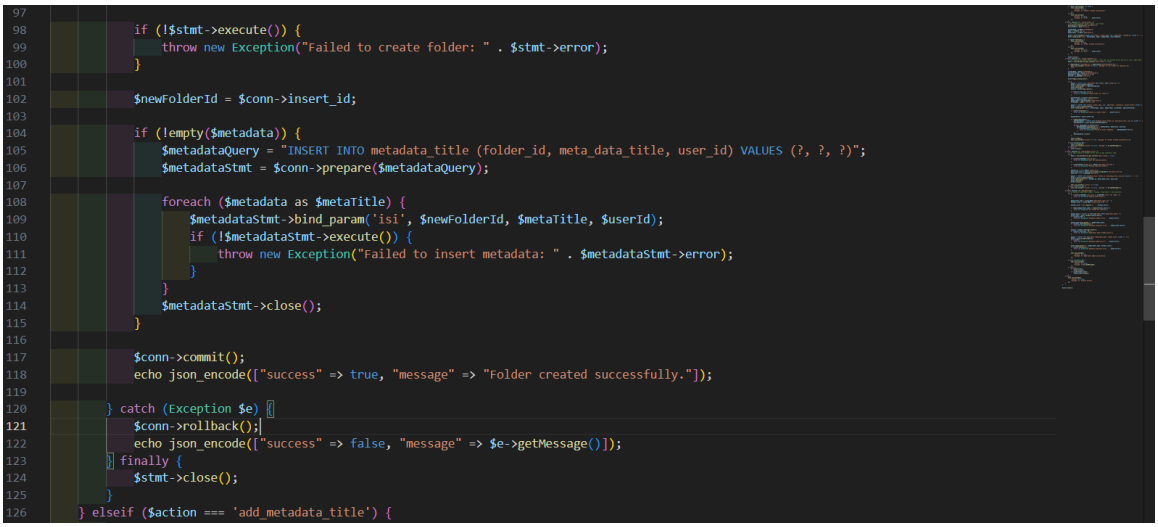
The function defines an array of possible input types (options), such as text, date, number, and email, ensuring flexibility in metadata collection. These options are dynamically mapped into

<option> elements and inserted into the <select> dropdown. Once 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. An error message is displayed in case of an error, ensuring users are informed of any issues in the saving process.

```php
} elseif ($action === 'create_subfolder') {
    // for creating sub folder (admin folders, under fab icon (katong button nga naa sa lower right side))
    $data = json_decode(file_get_contents('php://input'), true);

    if (empty($data['folderName']) || empty($data['parentFolderId'])) {
        echo json_encode(["success" => false, "message" => "All fields are required."]);
        exit;
    }

    $folderName = $data['folderName'];
    $parentFolderId = $data['parentFolderId'];
    $metadata = $data['metadata'] ?? [];
    $userId = $_SESSION['user_id'];

    $conn->begin_transaction();

    try {
        $query = "SELECT year, department FROM folders WHERE folder_id = ?";
        $stmt = $conn->prepare($query);
        $stmt->bind_param('i', $parentFolderId);
        $stmt->execute();
        $result = $stmt->get_result();

        if ($result->num_rows === 0) {
            throw new Exception("Parent folder not found.");
        }

        $parentFolder = $result->fetch_assoc();
        $year = $parentFolder['year'];
        $department = $parentFolder['department'];
        $createdAt = date('Y-m-d H:i:s');

        $query = "INSERT INTO folders (folder_name, year, department, created_at, parent_folder) VALUES (?, ?, ?, ?, ?)";
        $stmt = $conn->prepare($query);
        $stmt->bind_param('ssssi', $folderName, $year, $department, $createdAt, $parentFolderId);
```



```
97
98              if (!$stmt->execute()) {
99                  throw new Exception("Failed to create folder: " . $stmt->error);
100             }
101
102             $newFolderId = $conn->insert_id;
103
104             if (!empty($metadata)) {
105                 $metadataQuery = "INSERT INTO metadata_title (folder_id, meta_data_title, user_id) VALUES (?, ?, ?)";
106                 $metadataStmt = $conn->prepare($metadataQuery);
107
108                 foreach ($metadata as $metaTitle) {
109                     $metadataStmt->bind_param('isi', $newFolderId, $metaTitle, $userId);
110                     if (!$metadataStmt->execute()) {
111                         throw new Exception("Failed to insert metadata: " . $metadataStmt->error);
112                     }
113                 }
114                 $metadataStmt->close();
115             }
116
117             $conn->commit();
118             echo json_encode(["success" => true, "message" => "Folder created successfully."]);
119
120         } catch (Exception $e) {
121             $conn->rollback();
122             echo json_encode(["success" => false, "message" => $e->getMessage()]);
123         } finally {
124             $stmt->close();
125         }
126     } elseif ($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. Before proceeding, the required `folderName` and `parentFolderId` fields are first ensured. 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 avoid SQL injection and maintain security. Exception handling through a try...catch block ensures that database errors are seen, 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.

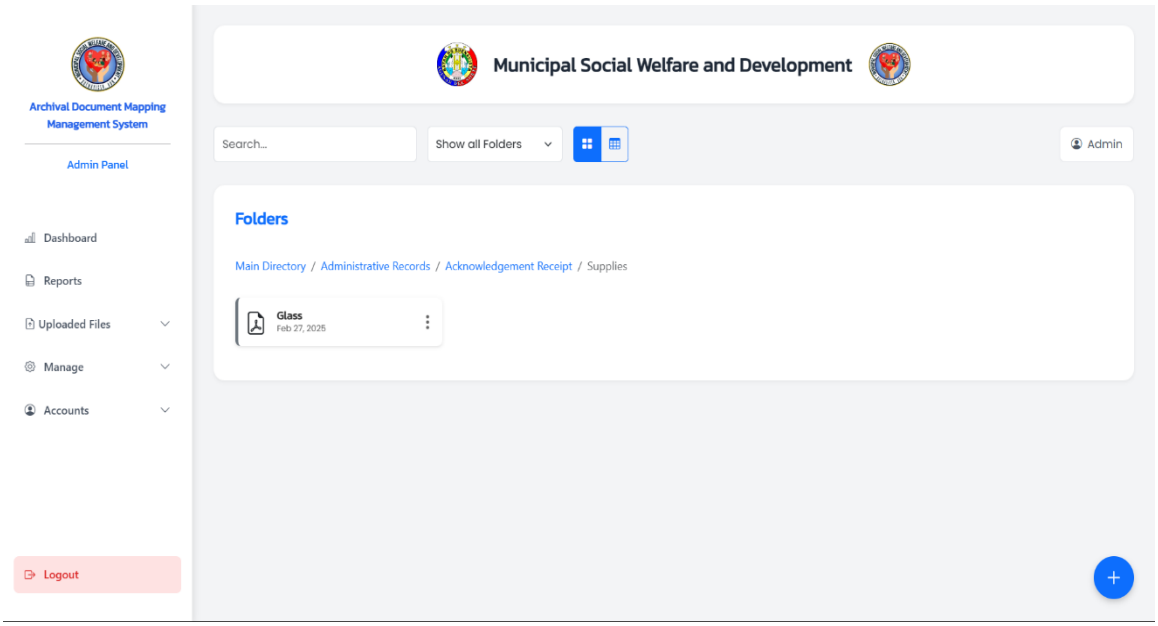


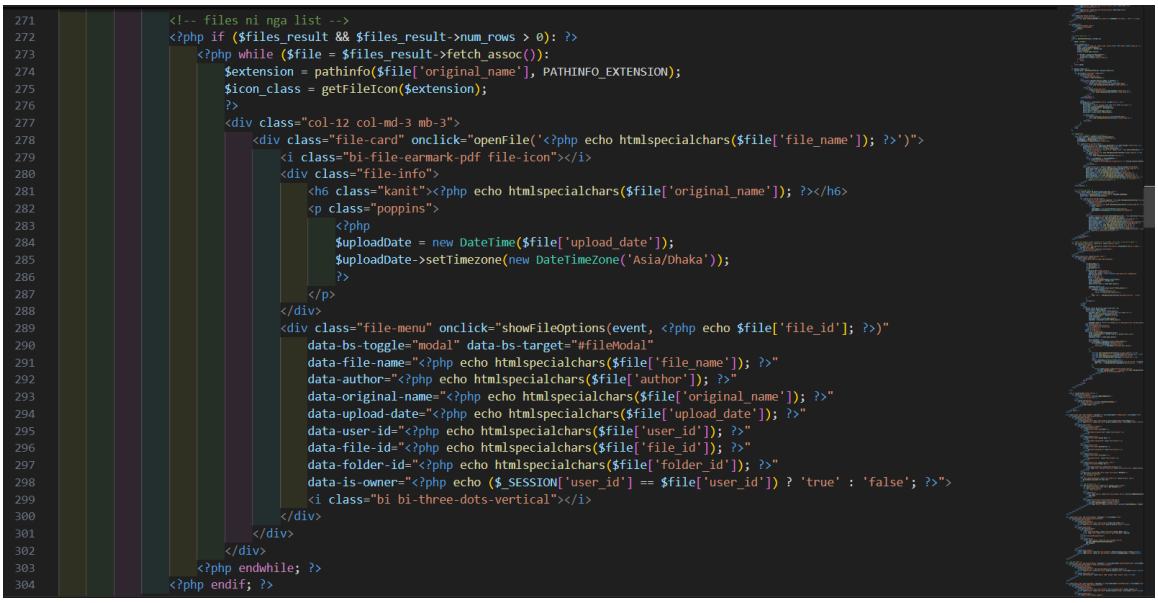
Figure 70. Admin – Display Files Card Style

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

```
document.addEventListener('DOMContentLoaded', function () {
    const viewToggle = document.getElementById('viewToggle');
    const cardView = document.querySelector('.row');
    const tableView = document.querySelector('.table-view');
    const folderCards = document.querySelectorAll('.folder-card');
    const tableSearchContainer = document.getElementById('tableSearchContainer');
    const tableSearchInput = document.getElementById('tableSearch');
    const printButton = document.getElementById('printButton');

    if (folderCards.length === 0 && document.querySelectorAll('.file-card').length > 0) {
        viewToggle.style.display = 'inline-flex';
    }

    viewToggle.querySelectorAll('button').forEach(button => {
        button.addEventListener('click', function () {
            viewToggle.querySelectorAll('button').forEach(btn => {
                btn.classList.remove('active');
            });

            this.classList.add('active');

            if (this.dataset.view === 'card') {
                cardView.style.display = 'flex';
                tableView.style.display = 'none';
                tableSearchContainer.style.display = 'none';
            } else {
                cardView.style.display = 'none';
                tableView.style.display = 'block';
                tableSearchContainer.style.display = 'block';
            }
        });
    });
```

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

Figure 71 shows the code snippet for the `document.addEventListener('DOMContentLoaded,' function () {...})` 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, enabling seamless toggling between different display modes while ensuring that interface elements respond dynamically to available content.



```
<!-- files ni nga list -->
<?php if ($files_result && $files_result->num_rows > 0): ?>
    <?php while ($file = $files_result->fetch_assoc()):
        $extension = pathinfo($file['original_name'], PATHINFO_EXTENSION);
        $icon_class = getFileIcon($extension);
        ?>
        <div class="col-12 col-md-3 mb-3">
            <div class="file-card" onclick="openFile('<?php echo htmlspecialchars($file['file_name']); ?>')">
                <i class="bi-file-earmark-pdf file-icon"></i>
                <div class="file-info">
                    <h6 class="kanit"><?php echo htmlspecialchars($file['original_name']); ?></h6>
                    <p class="poppins">
                        <?php
                        $uploadDate = new DateTime($file['upload_date']);
                        $uploadDate->setTimezone(new DateTimeZone('Asia/Dhaka'));
                        ?>
                    </p>
                </div>
                <div class="file-menu" onclick="showFileOptions(event, <?php echo $file['file_id']; ?>)"
                    data-bs-toggle="modal" data-bs-target="#fileModal"
                    data-file-name="<?php echo htmlspecialchars($file['file_name']); ?>"
                    data-author="<?php echo htmlspecialchars($file['author']); ?>"
                    data-original-name="<?php echo htmlspecialchars($file['original_name']); ?>"
                    data-upload-date="<?php echo htmlspecialchars($file['upload_date']); ?>"
                    data-user-id="<?php echo htmlspecialchars($file['user_id']); ?>"
                    data-file-id="<?php echo htmlspecialchars($file['file_id']); ?>"
                    data-folder-id="<?php echo htmlspecialchars($file['folder_id']); ?>"
                    data-is-owner="<?php echo ($_SESSION['user_id'] == $file['user_id']) ? 'true' : 'false'; ?>">
                    <i class="bi bi-three-dots-vertical"></i>
                </div>
            </div>
        </div>
    <?php endwhile; ?>
<?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.

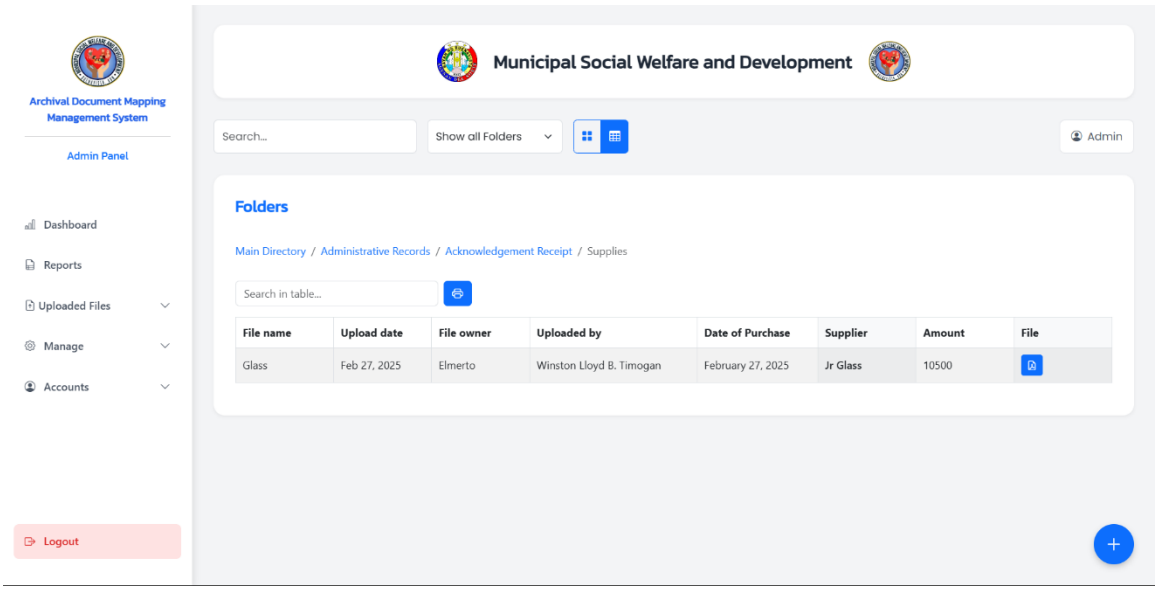
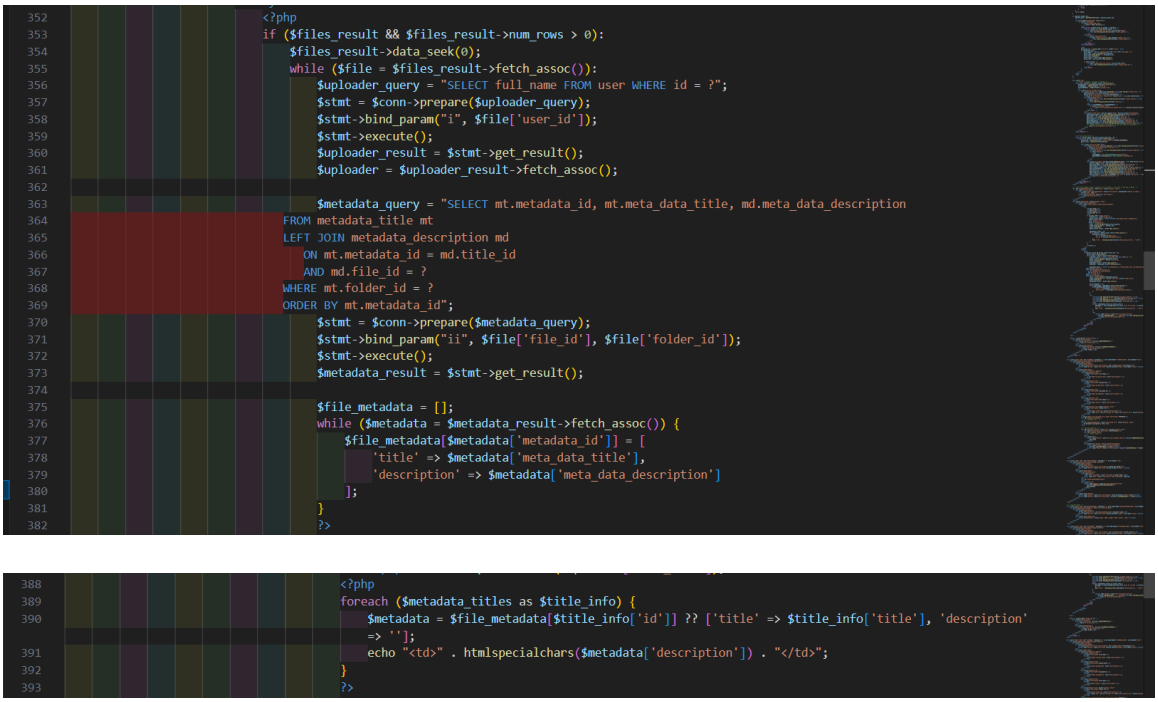


Figure 73. Admin – Display Files Table Style

Figure 73 shows the Display Files Table Style section on the admin interface, which 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.

```php
<?php
if (isset($_GET['folder_id'])) {
    $folder_id = $_GET['folder_id'];
    $meta_titles_query = "SELECT DISTINCT meta_data_title, metadata_id
    FROM metadata_title
    WHERE folder_id = ?
    ORDER BY metadata_id";
    $stmt = $conn->prepare($meta_titles_query);
    $stmt->bind_param("i", $folder_id);
    $stmt->execute();
    $meta_titles_result = $stmt->get_result();

    $metadata_titles = [];
    while ($title = $meta_titles_result->fetch_assoc()) {
        $metadata_titles[] = [
            'id' => $title['metadata_id'],
            'title' => $title['meta_data_title']
        ];
        echo "<th>" . htmlspecialchars($title['meta_data_title']) . "</th>";
    }
}
?>
```



```php
<?php
if ($files_result && $files_result->num_rows > 0):
    $files_result->data_seek(0);
    while ($file = $files_result->fetch_assoc()):
        $uploader_query = "SELECT full_name FROM user WHERE id = ?";
        $stmt = $conn->prepare($uploader_query);
        $stmt->bind_param("i", $file['user_id']);
        $stmt->execute();
        $uploader_result = $stmt->get_result();
        $uploader = $uploader_result->fetch_assoc();

        $metadata_query = "SELECT mt.metadata_id, mt.meta_data_title, md.meta_data_description
    FROM metadata_title mt
    LEFT JOIN metadata_description md
      ON mt.metadata_id = md.title_id
      AND md.file_id = ?
    WHERE mt.folder_id = ?
    ORDER BY mt.metadata_id";
        $stmt = $conn->prepare($metadata_query);
        $stmt->bind_param("ii", $file['file_id'], $file['folder_id']);
        $stmt->execute();
        $metadata_result = $stmt->get_result();

        $file_metadata = [];
        while ($metadata = $metadata_result->fetch_assoc()) {
            $file_metadata[$metadata['metadata_id']] = [
                'title' => $metadata['meta_data_title'],
                'description' => $metadata['meta_data_description']
            ];
        }
        ?>
```

```php
<?php
foreach ($metadata_titles as $title_info) {
    $metadata = $file_metadata[$title_info['id']] ?? ['title' => $title_info['title'], 'description' => ''];
    echo "<td>" . htmlspecialchars($metadata['description']) . "</td>";
}
?>
```

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. 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 to 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. Improving database performance by indexing key columns `(metadata_id, folder_id, file_id)`

would also 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.

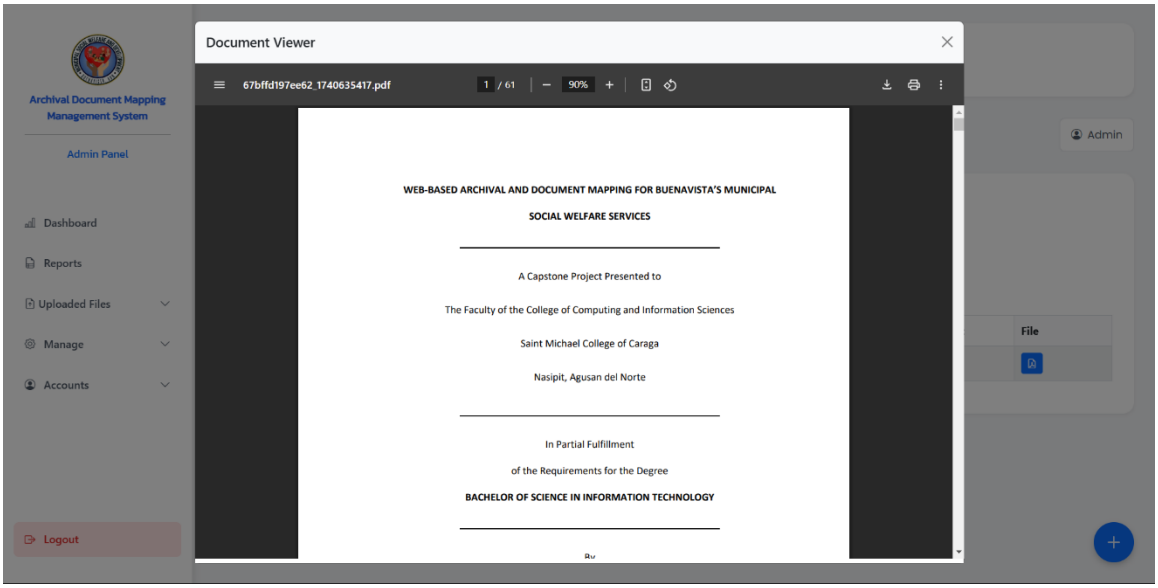


Figure 75. Admin - Document PDF Viewer

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

```
open-file.js
assets > js > open-file.js > addEventListener('hidden.bs.modal') callback
function openFile(filename, shouldShowModal = false) {
    if (shouldShowModal) {
        const url = new URL(window.location.href);
        url.searchParams.set('file', filename);
        window.history.pushState({}, '', url);

        // file id
        const fileCard = document.querySelector(`.file-card [data-file-name="${filename}"]`);
        const fileId = fileCard.getAttribute('data-file-id');

        document.getElementById('pdfViewer').src = '../../uploads/' + filename;
        fetchAndDisplayMetadata(fileId);

        const pdfModal = new bootstrap.Modal(document.getElementById('pdfViewerModal'));
        pdfModal.show();
    }
}

document.getElementById('pdfViewerModal').addEventListener('hidden.bs.modal', function () {
    const url = new URL(window.location.href);
    url.searchParams.delete('file');
    window.history.pushState({}, '', url);

    document.getElementById('pdfViewer').src = '';
});

document.querySelectorAll('.file-card').forEach(card => {
    let tapedTwice = false;
    let lastTap = 0;
    let clickTimeout = null;

    card.addEventListener('touchend', function (e) {
        const currentTime = new Date().getTime();
        const tapLength = currentTime - lastTap;
```

```
        if (tapLength < 300 && tapLength > 0) {
            e.preventDefault();
            const filename = this.querySelector('.file-menu').getAttribute('data-file-name');
            openFile(filename, true);
        }

        lastTap = currentTime;
    });

    card.addEventListener('dblclick', function (e) {
        if (clickTimeout) {
            clearTimeout(clickTimeout);
            clickTimeout = null;
        }

        const filename = this.querySelector('.file-menu').getAttribute('data-file-name');
        openFile(filename, true);
    });
});

document.addEventListener('DOMContentLoaded', function () {
    const urlParams = new URLSearchParams(window.location.search);
    const fileParam = urlParams.get('file');
    if (fileParam) {
        openFile(fileParam, true);
    }
});
```

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 clears the PDF source when the modal is closed to optimize memory usage. The script also includes touch and click event listeners, allowing users to open files via single or double taps and 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.

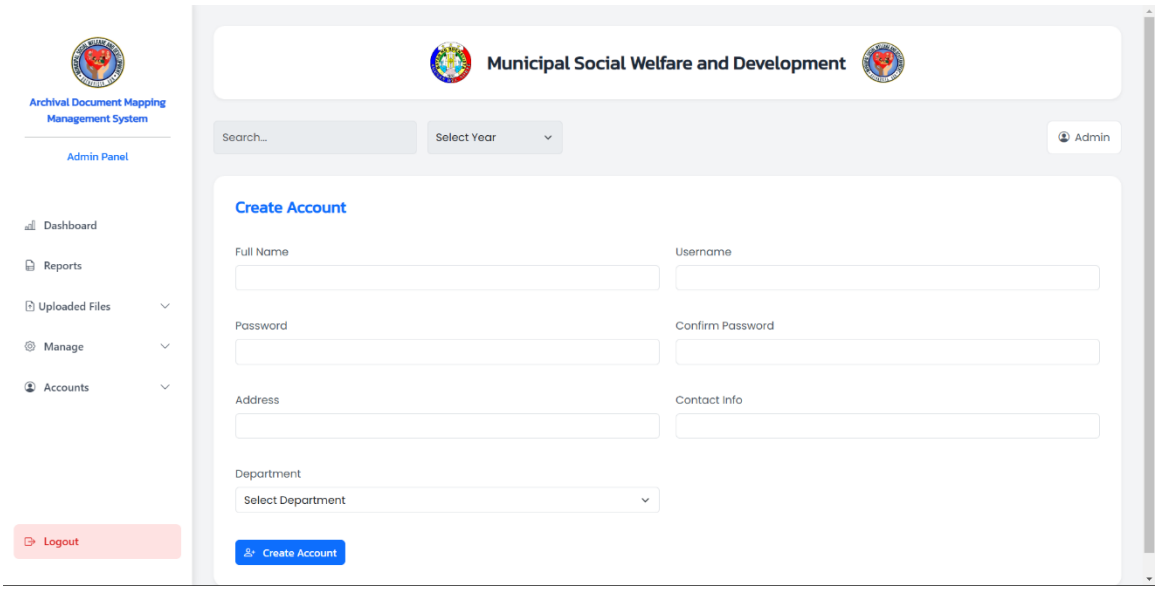


Figure 77. Admin - Create Account

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

```
// create account function (sugod diri padung ubos)
$(document).ready(function () {
    const $password = $('#password');
    const $confirmPassword = $('#confirmPassword');
    const $passwordMatchMessage = $('#passwordMatchMessage');
    const $submitButton = $('#submitButton');
    let passwordsMatch = false;

    function validatePasswords() {
        const password = $password.val();
        const confirmPassword = $confirmPassword.val();

        if (confirmPassword === '') {
            $confirmPassword.removeClass('password-mismatch');
            $passwordMatchMessage.text('');
            passwordsMatch = false;
        } else if (password !== confirmPassword) {
            $confirmPassword.addClass('password-mismatch');
            $passwordMatchMessage.text('Passwords do not match');
            passwordsMatch = false;
        } else {
            $confirmPassword.removeClass('password-mismatch');
            $passwordMatchMessage.text('');
            passwordsMatch = true;
        }
    }

    $password.on('input', validatePasswords);
    $confirmPassword.on('input', validatePasswords);

    $('#createAccountForm').on('submit', function (e) {
        e.preventDefault();

        validatePasswords();
```

```
250
251     if (!passwordsMatch) {
252         Swal.fire({
253             title: 'Error!',
254             text: 'Passwords do not match!',
255             icon: 'error',
256             confirmButtonColor: '#dc2626'
257         });
258         return;
259     }
260
261     $.ajax({
262         url: '../../controllers/create.php',
263         type: 'POST',
264         data: $(this).serialize() + '&action=create_account',
265         dataType: 'json',
266         success: function (response) {
267             if (response.success) {
268                 Swal.fire({
269                     title: 'Success!',
270                     text: 'Account has been created successfully!',
271                     icon: 'success',
272                     confirmButtonColor: '#2563eb'
273                 }).then((result) => {
274                     if (result.isConfirmed || result.isDismissed) {
275                         window.location.href = 'accounts.php';
276                     }
277                 });
278             } else {
279                 Swal.fire({
280                     title: 'Error!',
281                     text: response.message || 'Something went wrong!',
282                     icon: 'error',
283                     confirmButtonColor: '#dc2626'
284                 });
285             }
286         },
287         error: function (xhr, status, error) {
288             let errorMessage = xhr.responseJSON && xhr.responseJSON.message
289                 ? xhr.responseJSON.message
290                 : 'Something went wrong with the server!';
291
292             Swal.fire({
293                 title: 'Error!',
294                 text: errorMessage,
295                 icon: 'error',
296                 confirmButtonColor: '#dc2626'
297             });
298         }
299     });
300 });
301 });
```

Figure 78. Code snippet for account – create account

Figure 78 shows the code snippet for the `validatePasswords` function, which ensures that the password and confirm password fields match before allowing form submission. It monitors user input continuously and provides immediate feedback by checking whether the confirm password field is empty, mismatched, or matches the original password correctly. 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 can 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.

```php
// for creating acount (admin | add account)
if ($action === 'create_account') {
    $full_name = $conn->real_escape_string($_POST['full_name']);
    $username = $conn->real_escape_string($_POST['username']);
    $department = $conn->real_escape_string($_POST['department']);
    $address = $conn->real_escape_string($_POST['address']);
    $contact_info = $conn->real_escape_string($_POST['contact_info']);
    $role = 'User';
    $status = 'enabled';
    $password = password_hash($conn->real_escape_string($_POST['password']), PASSWORD_BCRYPT);

    $sql = "INSERT INTO user (full_name, username, department, address, contact_info, password, role, status, created_at)
            VALUES ('$full_name', '$username', '$department', '$address', '$contact_info', '$password', 'role', '$status', NOW())";

    if ($conn->query($sql) === TRUE) {
        echo json_encode([
            'success' => true,
            'message' => 'Account created successfully!'
        ]);
    } else {
        echo json_encode([
            'success' => false,
            'message' => "Error: " . $conn->error
        ]);
    }
} elseif ($action === 'create_folder') {
```

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

Figure 79 shows the code snippet for the provided PHP script that 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 a 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.

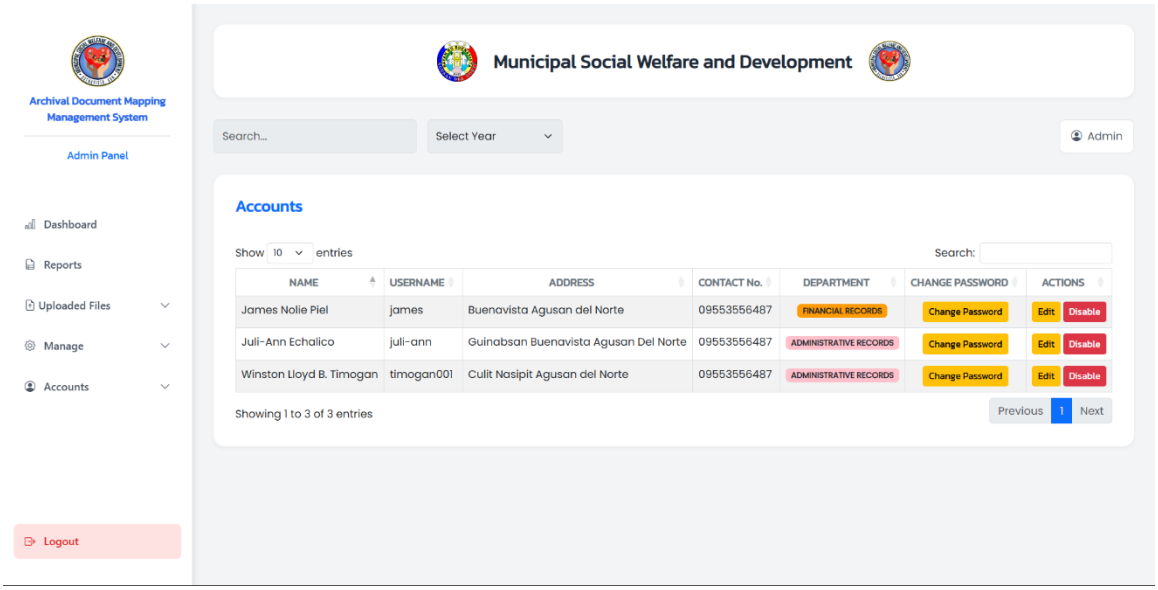


Figure 80. Admin – Display Accounts

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

```
// function to display lists of accounts (admin | accounts list)
function loadAccounts() {
    const tableName = 'user';
    $.getJSON(`../../controllers/get.php?table=${tableName}`, function (data) {
        if (data.error) {
            Swal.fire('Error', data.error, 'error');
            return;
        }

        let rows = '';
        if (data.length > 0) {
            data.forEach(item => {
                let badgeHtml = '';
                if (item.folder_color) {
                    badgeHtml = `<span class="badge text-dark" style="background-color: #${item.folder_color};">${item.department.toUpperCase()}</span>`;
                } else {
                    badgeHtml = `<span class="badge bg-primary">SYSTEM ADMIN</span>`;
                }

                const statusBtn = item.status === 'enabled' ?
                    `<button class='btn btn-sm btn-danger' onclick="toggleStatus(${item.id}, 'disabled')">Disable</button>` :
                    `<button class='btn btn-sm btn-success' onclick="toggleStatus(${item.id}, 'enabled')">Enable</button>`;

                rows += `
                <tr>
                    <td>${item.full_name}</td>
                    <td>${item.username}</td>
                    <td>${item.address}</td>
                    <td>${item.contact_info}</td>
                    <td class="text-center">${badgeHtml}</td>
                    <td class="text-center kanit">
                        <button class='btn btn-sm btn-warning' onclick="changePassword(${item.id})">
                            Change Password
                        </button>
                    </td>
                    <td class="text-center kanit">
                        <button class='btn btn-sm btn-warning' onclick="editUser(${item.id})">
                            Edit
```

```
                                Edit
                            </button>
                            ${statusBtn}
                        </td>
                    </tr>
                `;
            });
        } else {
            rows = '<tr><td colspan="6" class="text-center">No data</td></tr>';
        }

        $('#accounts-table').html(rows);

        try {
            if ($.fn.DataTable.isDataTable('#table')) {
                $('#table').DataTable().destroy();
            }
            $('#table').DataTable({});
        } catch (error) {
            console.error('Error initializing DataTable:', error);
        }
    }).fail(function (jqXHR, textStatus, errorThrown) {
        Swal.fire('Error', 'Failed to load accounts', 'error');
    });
}
```

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 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, suppose an AJAX request fails due to network or server errors. In that case, the function gracefully handles it by displaying an appropriate `SweetAlert` error message, ensuring a seamless and user-friendly experience.

```php
// handle requests for user table ----
$tableName = $_GET['table'];

// Fetching a single user
if (isset($_GET['id'])) {
    $id = $conn->real_escape_string($_GET['id']);
    $query = "SELECT id, full_name, username, address, department, contact_info FROM $tableName WHERE id = '$id'";
    $result = $conn->query($query);

    if ($result && $result->num_rows > 0) {
        echo json_encode($result->fetch_assoc());
    } else {
        echo json_encode(['error' => 'User not found']);
    }
}
// get user full name ----
elseif (isset($_GET['user_id'])) {
    $user_id = intval($_GET['user_id']);
    $query = "SELECT full_name FROM $tableName WHERE id = ?";
    $stmt = $conn->prepare($query);
    $stmt->bind_param("i", $user_id);
    $stmt->execute();
    $result = $stmt->get_result();

    if ($user = $result->fetch_assoc()) {
        echo json_encode(['success' => true, 'full_name' => $user['full_name']]);
    } else {
        echo json_encode(['success' => false, 'error' => 'User not found']);
    }
}
```

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

    $data = [];
    if ($result && $result->num_rows > 0) {
        while ($row = $result->fetch_assoc()) {
            $data[] = $row;
        }
    }

    echo json_encode($data);
}

$conn->close();
?>
```

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

Figure 82 shows the code snippet for the provided PHP script that 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 a user's full name based on a `User_id` parameter from the GET request. It prepares an SQL statement, securely binds the user ID parameter, 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.

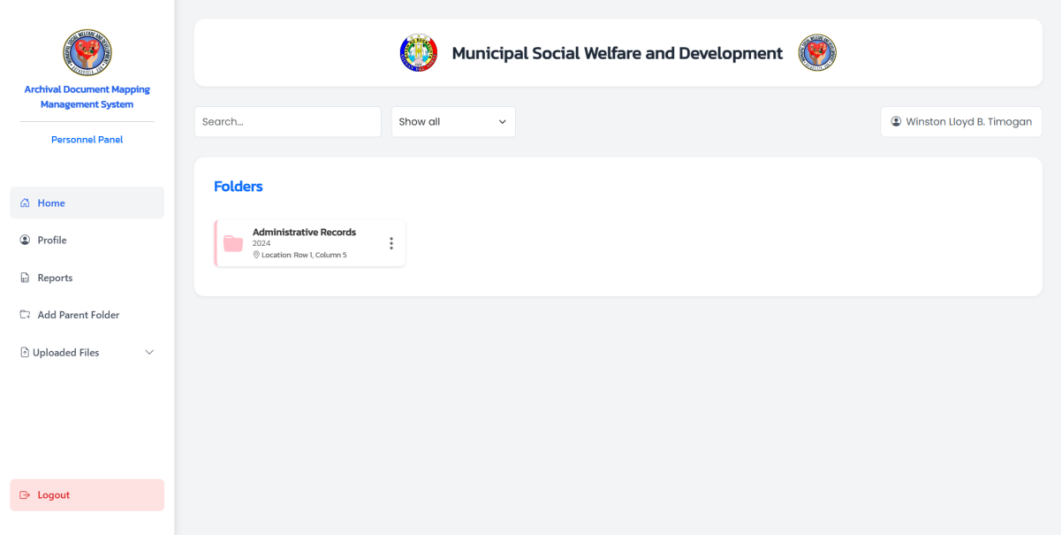


Figure 83. Personnel – Display Folders

Figure 83 shows the Display Folders section on the personnel interface, which presents 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.

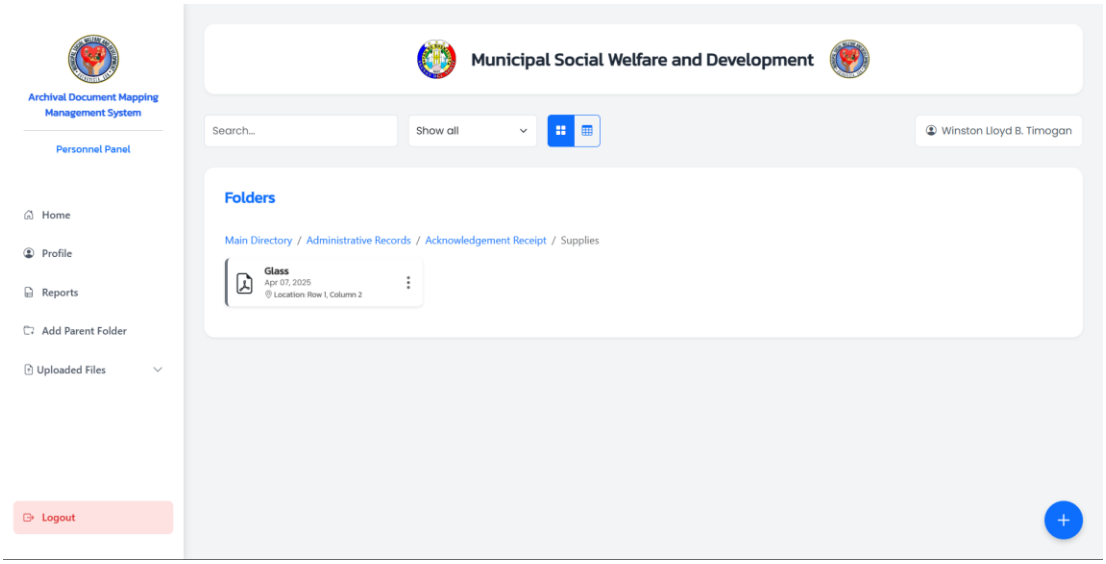


Figure 84. Personnel – Display Files Card Style

Figure 84 shows the Display Files Card Style section on the personnel interface, which 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.

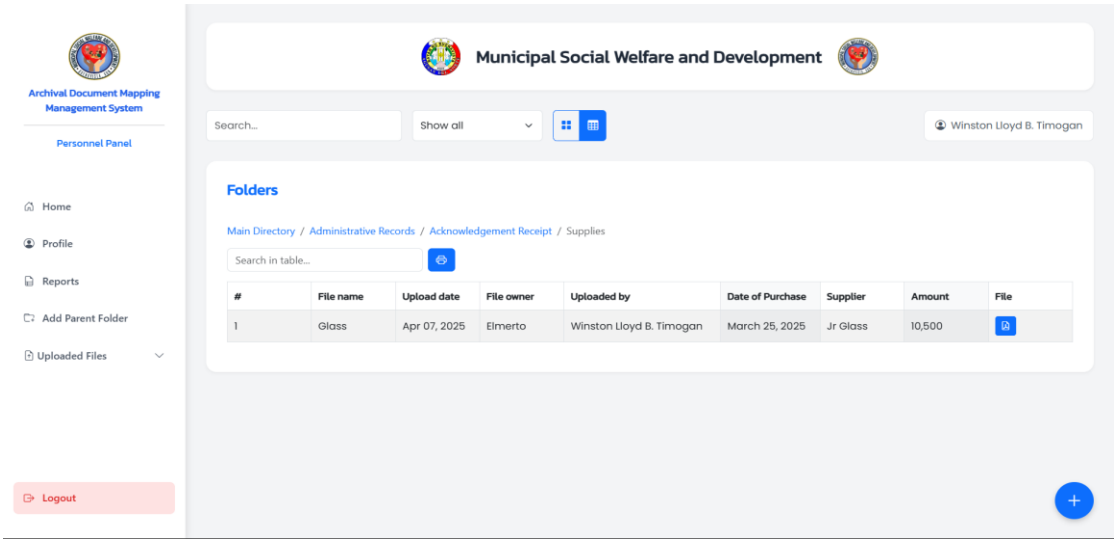


Figure 85. Personnel – Display Files Table Style

Figure 85 shows the Display Files Table Style section on the personnel interface, which 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.

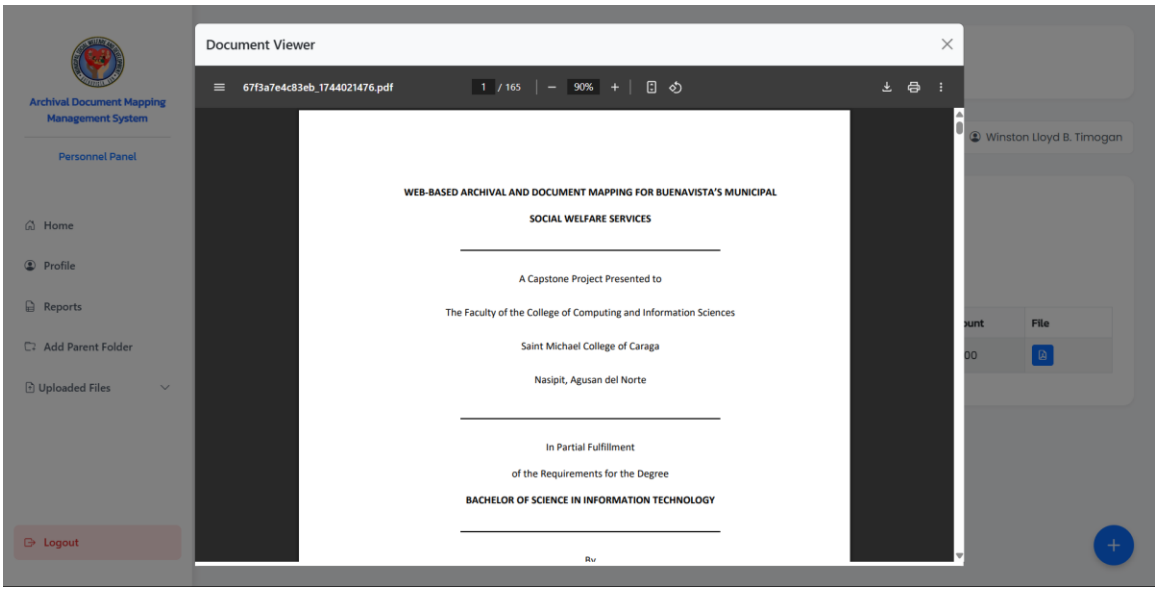


Figure 86. Personnel – Document PDF Viewer

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.

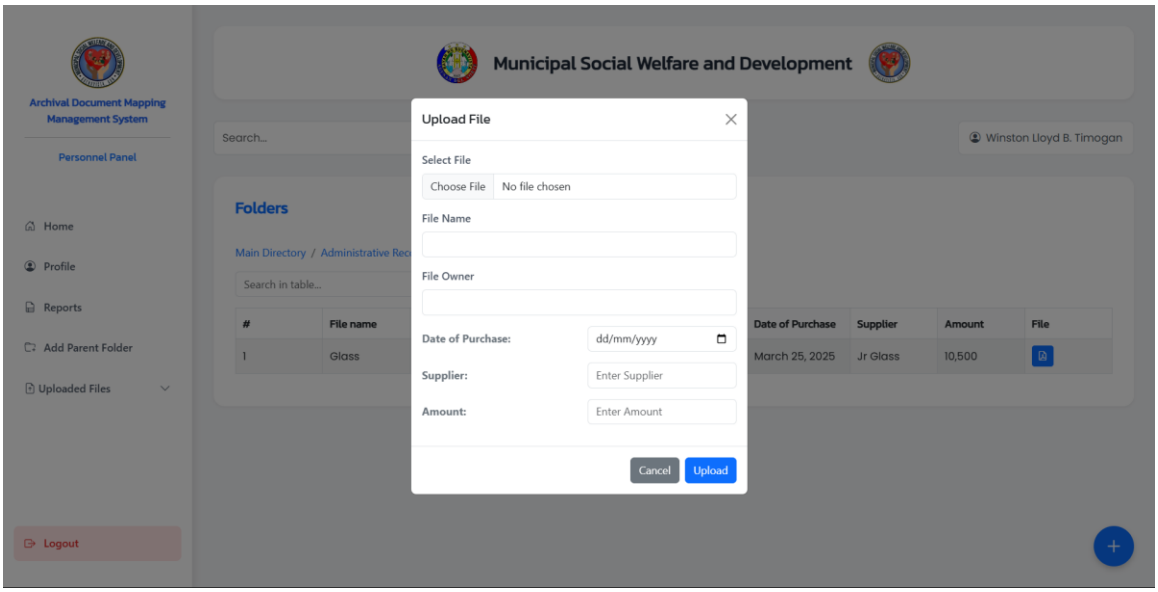


Figure 87. Personnel - Upload a file

Figure 87 shows the Upload a File section, which allows personnel to select and upload files to the system. Before submission, users can input details such as the selected file, file name,

file owner, and metadata description. This feature ensures organized and efficient document management.

```
function showUploadModal() {
    const modal = new bootstrap.Modal(document.getElementById('uploadModal'));
    const folderId = new URLSearchParams(window.location.search).get('folder_id');

    fetch(`../../controllers/get.php?table=metadata_titles&folder_id=${folderId}`)
        .then(response => response.json())
        .then(data => {
            const metadataSection = document.getElementById('metadataSection');
            metadataSection.innerHTML = '';

            if (data && data.length > 0) {
                data.forEach(metadata => {
                    const metadataRow = document.createElement('div');
                    metadataRow.className = 'row mb-3 align-items-center';

                    const labelCol = document.createElement('div');
                    labelCol.className = 'col-6 fw-bold text-secondary text-start';
                    labelCol.textContent = `${metadata.meta_data_title}:`;

                    const inputCol = document.createElement('div');
                    inputCol.className = 'col-6';

                    const input = document.createElement('input');
                    input.type = metadata.input_type || 'text'; // Use stored input type
                    input.className = 'form-control';
                    input.name = `description_${metadata.metadata_id}`;
                    input.placeholder = `Enter ${metadata.meta_data_title}`;

                    inputCol.appendChild(input);
                    metadataRow.appendChild(labelCol);
                    metadataRow.appendChild(inputCol);
                    metadataSection.appendChild(metadataRow);
                });

                console.log('Created metadata fields:', data.map(m => ({
                    title: m.meta_data_title,
                    id: m.metadata_id,
                    type: m.input_type
                })));
            } else {
                metadataSection.innerHTML = '<p>No metadata titles found for this folder.</p>';
            }
        })
        .catch(error => {
            console.error('Error fetching metadata titles:', error);
        });

    modal.show();
    document.querySelector('.fab-options').classList.remove('show');
}
```

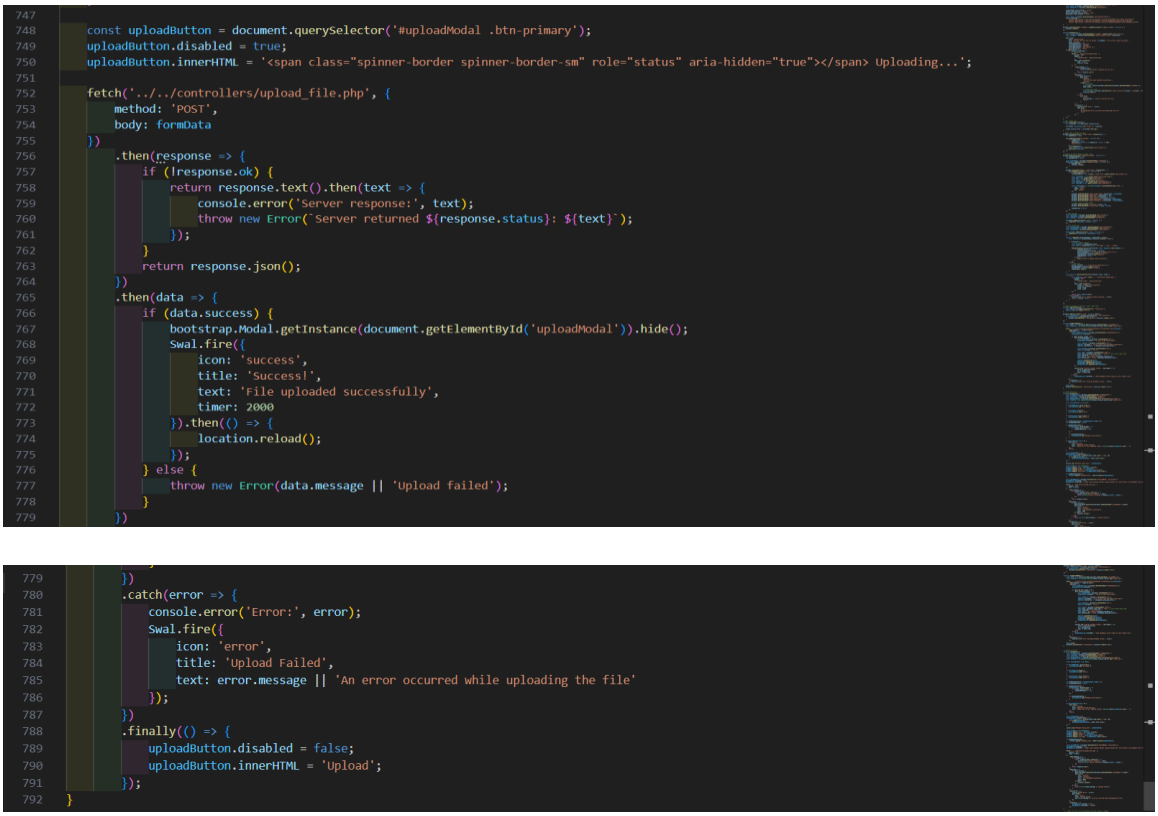
Figure 88. Code snippet for modal – upload file

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.

```
// upload file func.
function uploadFile() {
    const fileNameInput = document.getElementById('fileNameInput');
    const fileInput = document.getElementById('fileInput');
    const authorInput = document.getElementById('author');
    const metadataInputs = document.querySelectorAll('#metadataSection input');
    const folderId = new URLSearchParams(window.location.search).get('folder_id');

    const missingFields = new Set();

    if (!fileNameInput.value.trim()) {
        missingFields.add('File Name');
    }

    if (!fileInput.files[0]) {
        missingFields.add('File');
    }

    if (!authorInput.value.trim()) {
        missingFields.add('Author');
    }

    let hasMetadataFields = metadataInputs.length > 0;
    let hasEmptyMetadata = false;

    if (hasMetadataFields) {
        metadataInputs.forEach(input => {
            if (!input.value.trim()) {
                hasEmptyMetadata = true;
            }
        });

        if (hasEmptyMetadata) {
            missingFields.add('Metadata description');
        }
    }
```

```
    if (missingFields.size > 0) {
        Swal.fire({
            icon: 'warning',
            title: 'Required Fields Missing',
            text: `Please fill in all required fields: ${Array.from(missingFields).join(', ')}`
        });
        return;
    }

    const metadataData = {};
    metadataInputs.forEach(input => {
        const metadataId = parseInt(input.name.split('_')[1], 10);
        if (!isNaN(metadataId)) {
            metadataData[metadataId] = input.value.trim();
        }
    });

    console.log('Metadata being sent:', metadataData);

    const formData = new FormData();
    formData.append('file', fileInput.files[0]);
    formData.append('folder_id', folderId);
    formData.append('author', authorInput.value.trim());
    formData.append('file_name', fileNameInput.value.trim());

    if (hasMetadataFields) {
        formData.append('metadata_json', JSON.stringify(metadataData));
    }
```



```
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         })
780         .catch(error => {
781             console.error('Error:', error);
782             Swal.fire({
783                 icon: 'error',
784                 title: 'Upload Failed',
785                 text: error.message || 'An error occurred while uploading the file'
786             });
787         })
788         .finally(() => {
789             uploadButton.disabled = false;
790             uploadButton.innerHTML = 'Upload';
791         });
792 }
```

Figure 89. Code snippet for uploading – upload file

Figure 89 shows the code snippet for the `uploadFile()` function handles 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, 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 meaningfully. 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.

```php
    $metadata_json = $_POST['metadata_json'] ?? null;
    $metadata = ($metadata_json) ? json_decode($metadata_json, true) : [];

    if (json_last_error() !== JSON_ERROR_NONE) {
        throw new Exception('Invalid metadata format');
    }

    $metadata_ids = array_keys($metadata);
    $valid_ids = [];

    if (!empty($metadata_ids)) {
        $placeholders = str_repeat('?,', count($metadata_ids) - 1) . '?';

        $validation_query = "SELECT metadata_id FROM metadata_title
                             WHERE metadata_id IN ($placeholders)
                             AND folder_id = ?";

        $validation_stmt = $conn->prepare($validation_query);

        $types = str_repeat('i', count($metadata_ids)) . 'i';
        $args = array_merge([$types], $metadata_ids, [$folder_id]);

        call_user_func_array([$validation_stmt, 'bind_param'], $args);
        $validation_stmt->execute();
        $result = $validation_stmt->get_result();

        while ($row = $result->fetch_assoc()) {
            $valid_ids[] = $row['metadata_id'];
        }
        $validation_stmt->close();

        $invalid_ids = array_diff($metadata_ids, $valid_ids);
        if (!empty($invalid_ids)) {
            throw new Exception('Invalid metadata IDs: ' . implode(', ', $invalid_ids));
        }
    }
```

```php
    $conn->begin_transaction();

    try {
        date_default_timezone_set('Asia/Manila');
        $upload_dir = '../uploads/';
        if (!file_exists($upload_dir)) {
            mkdir($upload_dir, 0777, true);
        }

        $file_extension = pathinfo($file['name'], PATHINFO_EXTENSION);
        $unique_filename = uniqid() . '_' . time() . '.' . $file_extension;
        $upload_path = $upload_dir . $unique_filename;

        if (!move_uploaded_file($file['tmp_name'], $upload_path)) {
            throw new Exception('Failed to move uploaded file');
        }

        $query = "INSERT INTO files (file_name, original_name, upload_date, user_id, folder_id, author, department)
                  VALUES (?, ?, ?, ?, ?, ?, ?)";
        $stmt = $conn->prepare($query);
        $current_date = date('Y-m-d H:i:s');

        $stmt->bind_param(
            "sssisss",
            $unique_filename,
            $file_name,
            $current_date,
            $user_id,
            $folder_id,
            $author,
            $department
        );

        if (!$stmt->execute()) {
            throw new Exception('Failed to insert file record: ' . $stmt->error);
        }
```

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

```
146
147 } catch (Exception $e) {
148     error_log('Upload error: ' . $e->getMessage());
149     echo json_encode([
150         'success' => false,
151         'message' => $e->getMessage()
152     ]);
153 }
154
155 $conn->close();
156 ?>
```

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

Figure 90 shows the code snippet for the PHP script that 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 the execution and returning an error message in JSON format. The script also validates metadata by checking if it is a properly formatted JSON string. It 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 would 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 to the designated upload directory from its temporary location. 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 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 and 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.

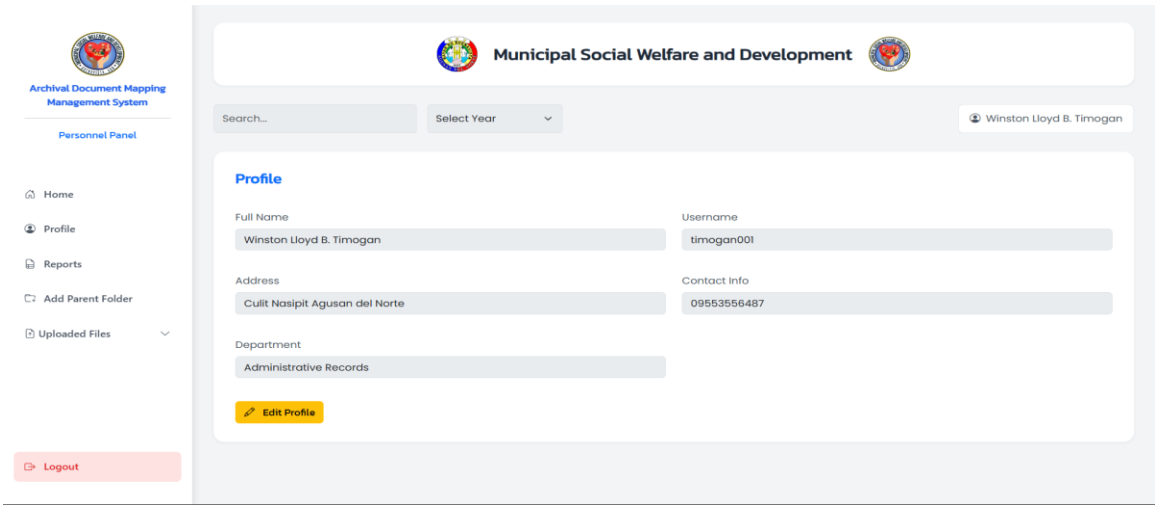


Figure 91. Personnel – Profile

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

```php
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'];

        $redirect = $user['role'] === 'Admin'
            ? 'pages/admin/dashboard.php'
            : 'pages/personnel/home.php';

        echo json_encode([
            'success' => true,
            'message' => 'Login successful!',
            'redirect' => $redirect
        ]);
    } else {
        echo json_encode([
            'success' => false,
            'message' => 'Invalid password!'
        ]);
    }
```

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 front-end applications. If 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 front end to request and display user profile details dynamically while maintaining data integrity and preventing unauthorized access.

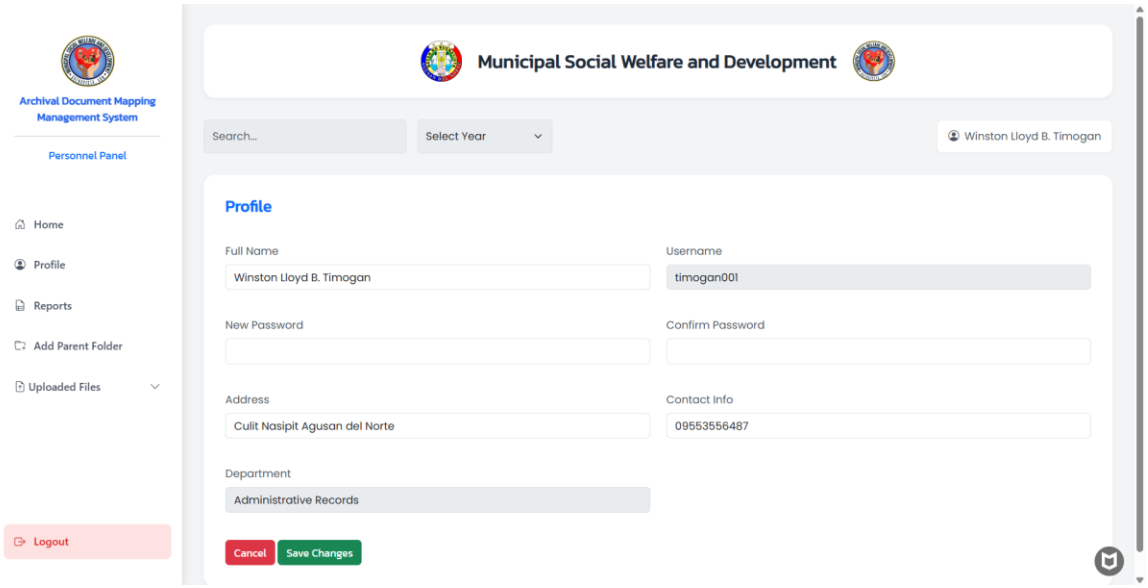


Figure 93. Personnel – update profile

Figure 93 shows the Update Profile section, which 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.

```
$(document).ready(function () {
    const originalValues = {
        fullName: $('#fullName').val(),
        address: $('#address').val(),
        contactInfo: $('#contactInfo').val()
    };

    $('#editButton').click(function () {
        $('#profile input:not(#username, #department)').prop('disabled', false);
        $('.password-field').show();
        $(this).addClass('d-none');
        $('#saveButton').removeClass('d-none');
        $('#cancelButton').removeClass('d-none');
    });

    $('#cancelButton').click(function () {
        $('#fullName').val(originalValues.fullName);
        $('#address').val(originalValues.address);
        $('#contactInfo').val(originalValues.contactInfo);

        $('#password').val('');
        $('#confirmPassword').val('');
        $('#passwordMatchMessage').text('');

        $('#profile input:not(#username, #department)').prop('disabled', true);
        $('.password-field').hide();

        $('#editButton').removeClass('d-none');
        $('#saveButton').addClass('d-none');
        $(this).addClass('d-none');
    });

    const $password = $('#password');
    const $confirmPassword = $('#confirmPassword');
    const $passwordMatchMessage = $('#passwordMatchMessage');
```

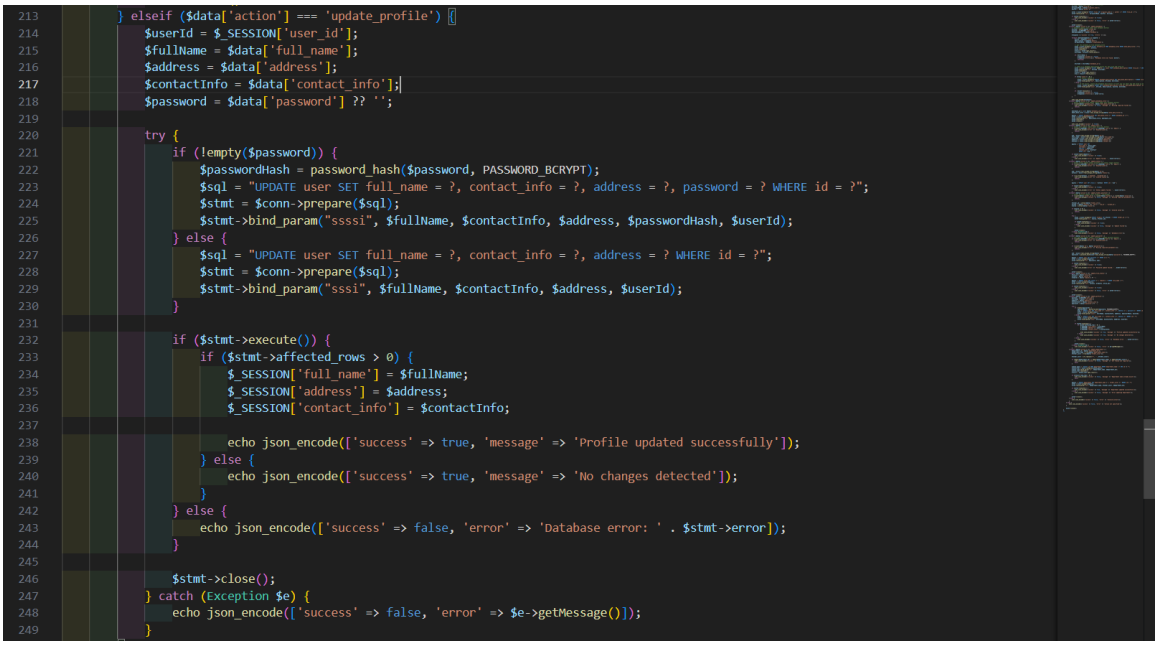
```
    $password.add($confirmPassword).on('input', function () {
        validatePasswords();
    });

    function validatePasswords() {
        const password = $password.val();
        const confirmPassword = $confirmPassword.val();

        if (confirmPassword === '') {
            $confirmPassword.removeClass('password-mismatch');
            $passwordMatchMessage.text('');
        } else if (password !== confirmPassword) {
            $confirmPassword.addClass('password-mismatch');
            $passwordMatchMessage.text('Passwords do not match');
        } else {
            $confirmPassword.removeClass('password-mismatch');
            $passwordMatchMessage.text('');
        }
    }

    $('#profile').submit(function (e) {
        e.preventDefault();

        const formData = {
            action: 'update_profile',
            full_name: $('#fullName').val(),
            address: $('#address').val(),
            contact_info: $('#contactInfo').val(),
            password: $('#password').val()
        };
```

```
202
203             $.ajax({
204                 type: 'POST',
205                 url: '../../controllers/update.php',
206                 data: JSON.stringify(formData),
207                 contentType: 'application/json',
208                 success: function (response) {
209                     const result = JSON.parse(response);
210                     if (result.success) {
211                         Swal.fire({
212                             icon: 'success',
213                             title: 'Profile Updated',
214                             text: result.message,
215                         }).then(() => {
216                             location.reload();
217                         });
218                     } else {
219                         Swal.fire({
220                             icon: 'error',
221                             title: 'Error',
222                             text: result.error || 'There was an error updating your profile.',
223                         });
224                     }
225                 }
226             });
227         });
228     });
```

Figure 94. Code snippet for profile – edit profile

Figure 94 shows the code snippet for the provided jQuery script that manages user profile editing by enabling input fields, validating password inputs, and handling form submissions 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. Suppose the user decides to cancel the changes. In that case, 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, ensuring that the confirmation field matches the entered password, providing immediate user feedback, and visually highlighting mismatches 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.



```
} elseif ($data['action'] === 'update_profile') {
    $userId = $_SESSION['user_id'];
    $fullName = $data['full_name'];
    $address = $data['address'];
    $contactInfo = $data['contact_info'];
    $password = $data['password'] ?? '';

    try {
        if (!empty($password)) {
            $passwordHash = password_hash($password, PASSWORD_BCRYPT);
            $sql = "UPDATE user SET full_name = ?, contact_info = ?, address = ?, password = ? WHERE id = ?";
            $stmt = $conn->prepare($sql);
            $stmt->bind_param("ssssi", $fullName, $contactInfo, $address, $passwordHash, $userId);
        } else {
            $sql = "UPDATE user SET full_name = ?, contact_info = ?, address = ? WHERE id = ?";
            $stmt = $conn->prepare($sql);
            $stmt->bind_param("sssi", $fullName, $contactInfo, $address, $userId);
        }

        if ($stmt->execute()) {
            if ($stmt->affected_rows > 0) {
                $_SESSION['full_name'] = $fullName;
                $_SESSION['address'] = $address;
                $_SESSION['contact_info'] = $contactInfo;

                echo json_encode(['success' => true, 'message' => 'Profile updated successfully']);
            } else {
                echo json_encode(['success' => true, 'message' => 'No changes detected']);
            }
        } else {
            echo json_encode(['success' => false, 'error' => 'Database error: ' . $stmt->error]);
        }

        $stmt->close();
    } catch (Exception $e) {
        echo json_encode(['success' => false, 'error' => $e->getMessage()]);
    }
```

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

Figure 95 shows the code snippet for the function in the provided PHP script that handles the profile update process for a user, allowing changes to their `full_name`, `contact`, `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. Try-catch enhances the function's robustness by catching exceptions and preventing system crashes, ensuring a smooth user experience and security in profile management.

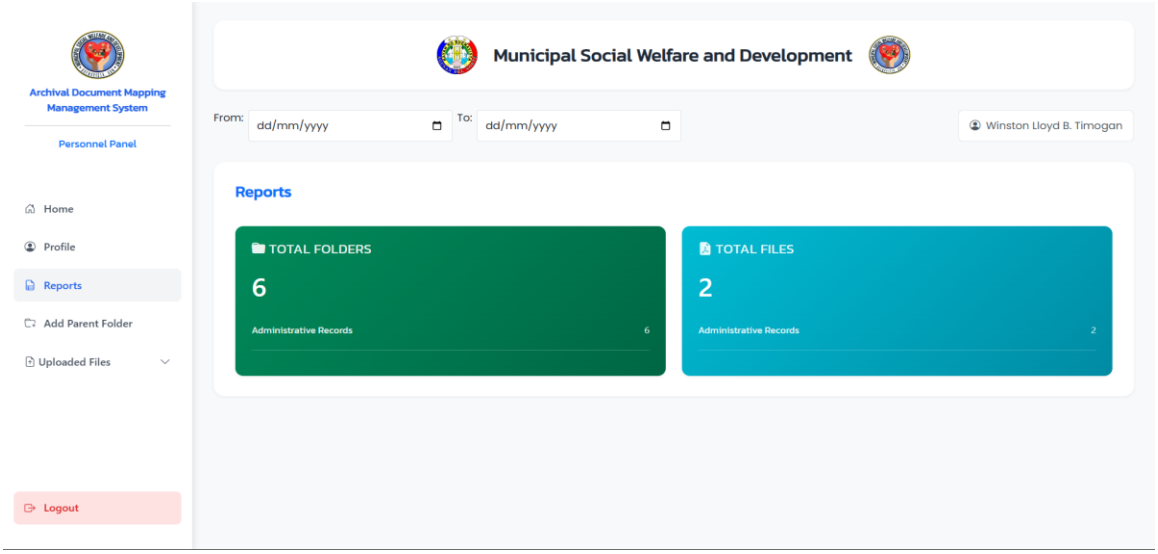


Figure 96. Personnel – Reports

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

```php
$userStats = [];
$folderStats = [];
$fileStats = [];

$fromDate = isset($_GET['fromDate']) ? $_GET['fromDate'] : null;
$toDate = isset($_GET['toDate']) ? $_GET['toDate'] : null;

$dateCondition = "";
if ($fromDate && $toDate) {
    $fileDate = "AND DATE(upload_date) BETWEEN '$fromDate' AND '$toDate'";
    $createdDate = "AND DATE(created_at) BETWEEN '$fromDate' AND '$toDate'";
} else {
    $fileDate = "";
    $createdDate = "";
}

$userDepartment = $_SESSION['department'];

$filesQuery = "SELECT department, COUNT(*) as count
               FROM files
               WHERE department = '$userDepartment' $fileDate
               GROUP BY department";
$filesResult = mysqli_query($conn, $filesQuery);

$totalFiles = 0;
while ($row = mysqli_fetch_assoc($filesResult)) {
    $fileStats[$row['department']] = $row['count'];
    $totalFiles += $row['count'];
}

$foldersQuery = "SELECT department, COUNT(*) as count
                 FROM folders
                 WHERE department = '$userDepartment' $createdDate
                 GROUP BY department";
$foldersResult = mysqli_query($conn, $foldersQuery);
```

```
$totalFolders = 0;
while ($row = mysqli_fetch_assoc($foldersResult)) {
    $folderStats[$row['department']] = $row['count'];
    $totalFolders += $row['count'];
}

mysqli_close($conn);
```

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

Figure 97 shows the code snippet for the script is 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 $fileStats`, which would store statistical data. It then retrieves `fromDate and toDate` parameters from the GET request to filter results based on date ranges; if both are provided, conditions are dynamically added to restrict queries to the specified period. The current user's department is extracted from the session, ensuring that the statistics reflect only the relevant department's data. The script first executes a query on the files table to count the files associated with the user's department while applying the date condition if available. The results are stored in `$fileStats`, and the total files are 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 `$totalFolders`. Using `mysqli_query and mysqli_fetch_assoc` ensures an iterative approach to collecting data efficiently, and finally, `mysqli_close($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.

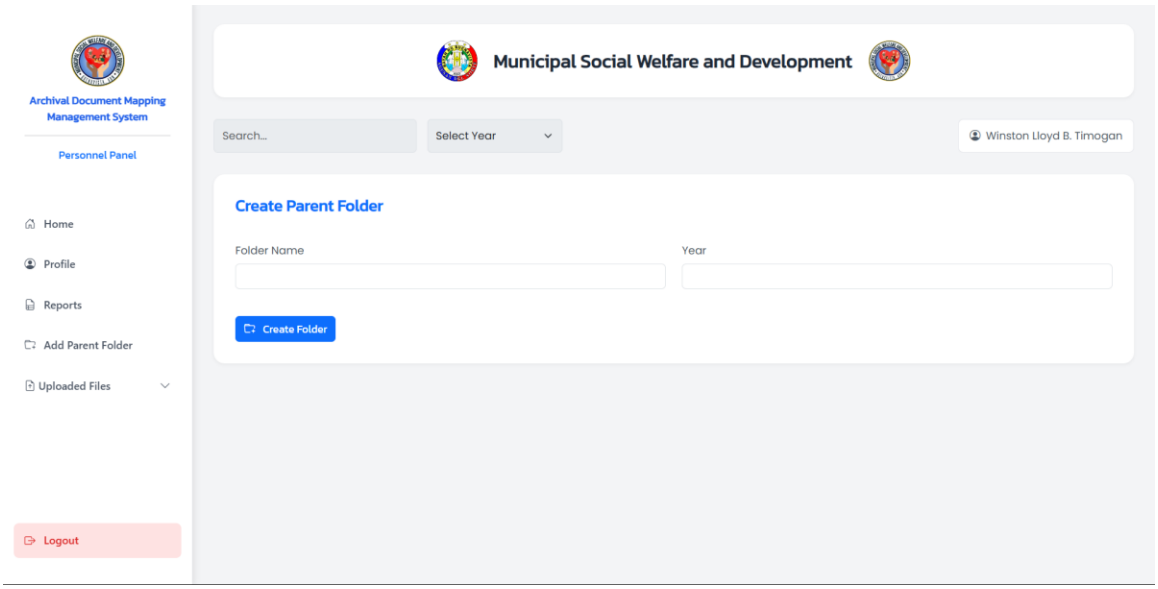


Figure 98. Personnel – Add Parent Folder

Figure 98 shows the Add Parent Folder section, which 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.

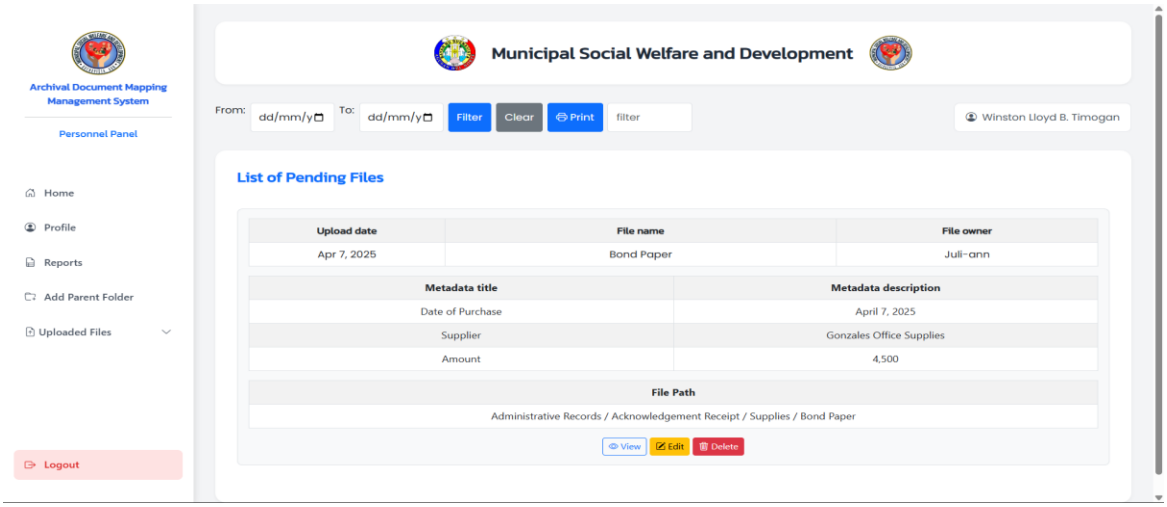


Figure 99. Personnel - List of Pending Files

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

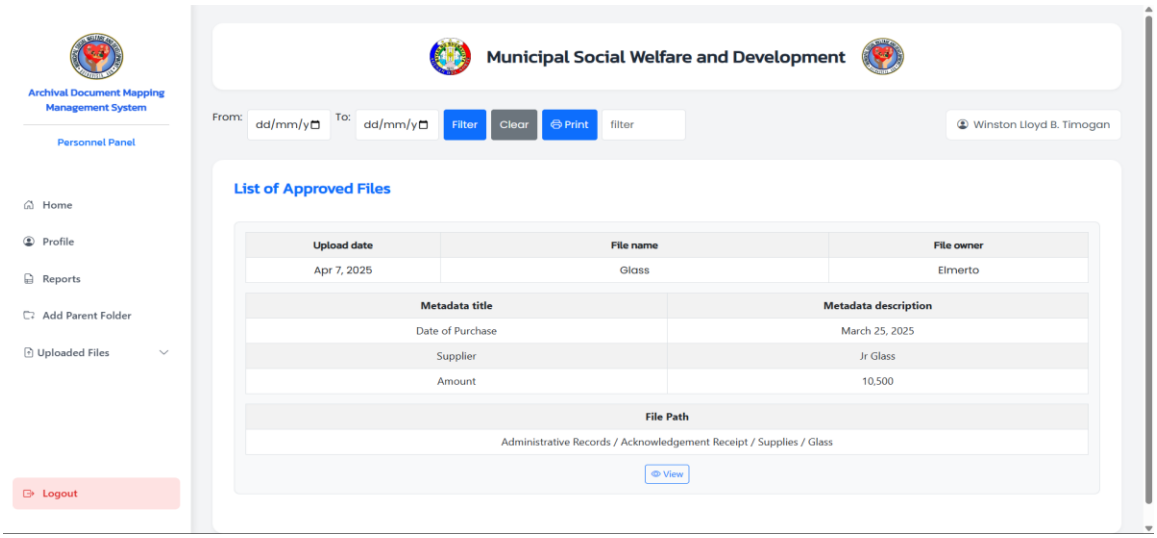


Figure 100. Personnel – List of Approved Files

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

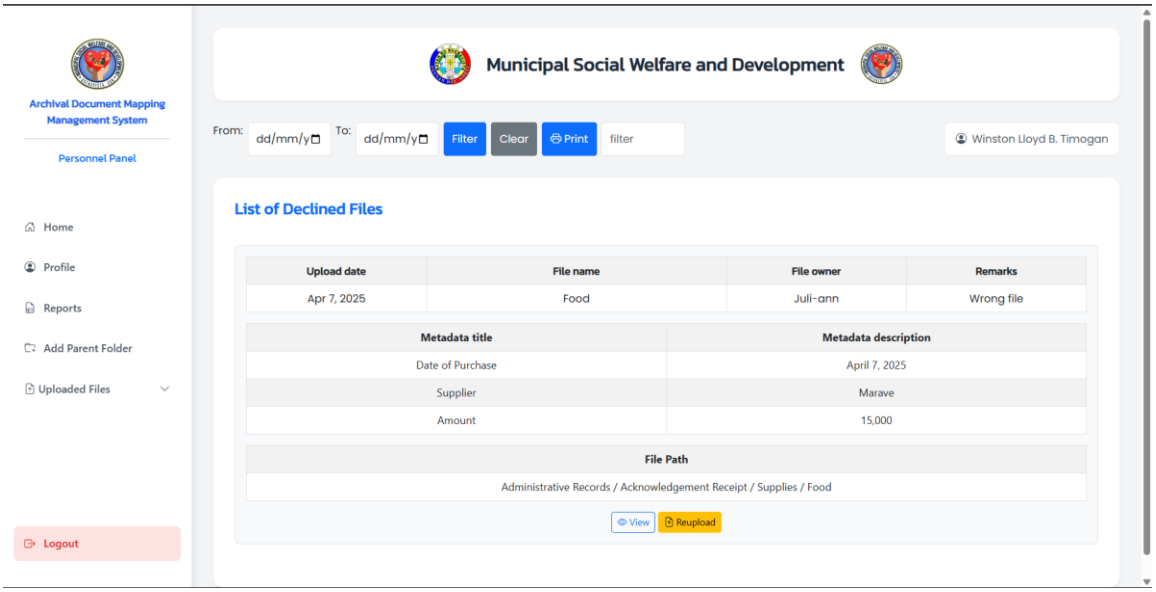


Figure 101. Personnel – List of Declined Files

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

```
    if (isAdmin) {
        if (!isDepartmentFilterApplied) {
            departmentColumn = `<th style="width: 20%;" class="text-center">Department</th>`;
        }
        uploadedByColumn = `<th style="width: 20%;" class="text-center">Uploaded By</th>`;
    }

    if (showRemarks) {
        remarksColumn = `<th style="width: 20%;" class="text-center">Remarks</th>`;
    }

    return `
        <table class="table table-bordered" style="width: 100%; margin-bottom: 0;">
            <tr>
                <th style="width: 15%;" class="text-center">Upload Date</th>
                <th style="width: 30%;" class="text-center">File Name</th>
                ${departmentColumn}
                <th style="width: 20%;" class="text-center">File Owner</th>
                ${uploadedByColumn}
                ${remarksColumn}
            </tr>
            <tr>
                <td>${uploadDate}</td>
                <td>${fileName}</td>
                ${isAdmin && !isDepartmentFilterApplied ? `<td>${department}</td>` : ''}
                <td>${fileOwner}</td>
                ${isAdmin ? `<td>${uploadedBy}</td>` : ''}
                ${showRemarks ? `<td>${remarks}</td>` : ''}
            </tr>
        </table>
    `;
}
```

```
generateMetadataPrintTable(metadataTable) {
    let content = `
        <table class="table table-bordered" style="width: 100%; margin-bottom: 0;">
            <tr>
                <th style="width: 50%;" class="text-center">Metadata title</th>
                <th style="width: 50%;" class="text-center">Metadata description</th>
            </tr>`;

    const metadataRows = metadataTable.querySelectorAll('tr:not(:first-child)');
    let hasValidData = false;

    metadataRows.forEach(row => {
        const cells = row.querySelectorAll('td');
        if (cells.length === 2) {
            const title = cells[0].textContent.trim();
            const description = cells[1].textContent.trim();
            if (title || description) {
                hasValidData = true;
                content += `
                    <tr>
                        <td>${title || 'No data found'}</td>
                        <td>${description || 'No data found'}</td>
                    </tr>`;
            }
        }
    });
```

```

    if (!hasValidData) {
        content += `
            <tr>
                <td colspan="2" class="text-center">No data found</td>
            </tr>`;
    }

    content += `</table>`;
    return content;
}

generateFilePathPrintTable(filePathTable) {
    const filePath = filePathTable.querySelector('tr:last-child td')?.textContent || '';
    return `
        <table class="table table-bordered mb-4" style="width: 100%; margin-bottom: 2rem;">
            <tr>
                <th class="text-center">File Path</th>
            </tr>
            <tr>
                <td>${filePath}</td>
            </tr>
        </table>`;
}
```

```
openPrintWindow(titleText, printContent) {
    const userRole = window.userRole;
    const userFullName = window.userFullName || "Unknown User";
    const userDepartment = window.userDepartment || "Unknown Department";

    let signatorySection = '';

    if (userRole === 'Admin') {
        signatorySection = `
        <div class="signatory-row">
            <div class="signatory" style="margin-left: auto; margin-right: auto;">
                <div class="name">MARIA BERNADITA N. QUIJANO, RSW</div>
                <div class="position">Municipal Social Welfare and Development Officer</div>
            </div>
        </div>
    `;
    } else {
        signatorySection = `
        <div class="signatory-row">
            <div class="signatory">
                <div class="name">${userFullName}</div>
                <div class="position">${userDepartment} Personnel</div>
            </div>

            <div class="signatory">
                <div class="name">MARIA BERNADITA N. QUIJANO, RSW</div>
                <div class="position">Municipal Social Welfare and Development Officer</div>
            </div>
        </div>
    `;
    }
```

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 table's last row, 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 and 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 access levels.

4.2 Testing Process

The testing process outlines the various procedures to assess the system's accuracy, functionality, and overall performance. The researchers utilized a survey instrument based on the System Usability, Functionality, and Efficiency Scale test to evaluate its effectiveness, efficiency, and reliability. 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 one 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.

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 most feedback comes from personnel directly interacting 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 accomplishment of specified tasks and objectives.	3.75	VF
	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 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 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 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 operates swiftly, 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 and satisfying interaction for the user.	3.62	VU
6.	Accessibility - Degree to which a system can be used by people with the widest range of characteristics and capabilities to achieve a specified goal in a specified context of use.	3.75	VU
	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 system's usability, 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 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 summarizes the system's overall acceptability, 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 summarizes the research's key findings, derives conclusions from the results, and suggests recommendations for future enhancements or further studies. It highlights the project's 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. Regarding 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 help users complete 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 would 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 objectives by providing a functional, efficient, and user-friendly platform for managing and accessing archival records. The system demonstrated strong functional suitability, meeting 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 effectively supported the department's archival and documentation processes.

5.3. Recommendation/s

With these enhancements, the Web-Based Archival and Document Mapping System would 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 would enable faster and more accurate document retrieval, ensuring seamless access even during peak usage. Comprehensive training materials would 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 would 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, would 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 promptly access the information they need.

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 would 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 improves workflow and supports data integrity and ease of access, which are essential for effective governance and service delivery.

Refinements to document mapping and reporting capabilities are recommended to better align with institutional needs, 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 would enhance the 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.

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/ijsait/2024/031332024.

[4] M. Churiyah, N. Muhajiroh, 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 CONTOL."

[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

```php
<?php
// database connection
$host = "localhost";
$username = "root";
$password = "";
$database = "admms";

$conn = new mysqli($host, $username, $password, $database);

if ($conn->connect_error) {
    die(json_encode([
        'success' => false,
        'message' => "Connection failed: " . $conn->connect_error
    ]));
}
?>
```

chart.php

```php
<?php
// chart functions (admin | dashboard)
require 'conn.php';

$fromDate = isset($_GET['fromDate']) ? $_GET['fromDate'] : null;
$toDate = isset($_GET['toDate']) ? $_GET['toDate'] : null;

// empty chart arrays (default)
$data = [
    'departments' => [],
    'files' => [],
    'folders' => [],
    'stackedFiles' => []
];

$deptQuery = "SELECT department_name, folder_color FROM department";
$result = $conn->query($deptQuery);
while ($row = $result->fetch_assoc()) {
    $data['departments'][] = $row;
}

$fileQuery = "SELECT department, upload_date FROM files";
if ($fromDate && $toDate) {
    $fileQuery .= " WHERE DATE(upload_date) BETWEEN '$fromDate' AND '$toDate'";
}
$result = $conn->query($fileQuery);
while ($row = $result->fetch_assoc()) {
    $data['files'][] = $row;
}

$folderQuery = "SELECT department, created_at FROM folders";
if ($fromDate && $toDate) {
    $folderQuery .= " WHERE DATE(created_at) BETWEEN '$fromDate' AND '$toDate'";
}
$result = $conn->query($folderQuery);
while ($row = $result->fetch_assoc()) {
    $data['folders'][] = $row;
}

$stackedQuery = "SELECT department, DATE(upload_date) as upload_date, COUNT(*) as count FROM files";
if ($fromDate && $toDate) {
    $stackedQuery .= " WHERE DATE(upload_date) BETWEEN '$fromDate' AND '$toDate'";
}
$stackedQuery .= " GROUP BY department, upload_date ORDER BY upload_date";
$result = $conn->query($stackedQuery);
while ($row = $result->fetch_assoc()) {
    $data['stackedFiles'][] = $row;
}

header('Content-Type: application/json');
echo json_encode($data);
```

check_duplicate.php

```php
<?php
session_start();
require_once 'conn.php';

header('Content-Type: application/json');

try {
    $data = json_decode(file_get_contents('php://input'), true);

    if (!isset($data['type']) || !isset($data['name'])) {
        throw new Exception('Missing required parameters');
    }

    $type = $data['type'];
    $name = $data['name'];

    if ($type === 'folder') {
        $parent_folder_id = $data['parent_folder_id'] ?? null;

        if ($parent_folder_id) {
            $query = "SELECT COUNT(*) as count FROM folders
                      WHERE folder_name = ? AND parent_folder = ?";
            $stmt = $conn->prepare($query);
            $stmt->bind_param("si", $name, $parent_folder_id);
        } else {
            $query = "SELECT COUNT(*) as count FROM folders
                      WHERE folder_name = ? AND (parent_folder IS NULL OR parent_folder = 0)";
            $stmt = $conn->prepare($query);
            $stmt->bind_param("s", $name);
        }

    } else if ($type === 'file') {
        $folder_id = $data['folder_id'];

        if (!$folder_id) {
            throw new Exception('Folder ID is required for file duplicate check');
        }
```

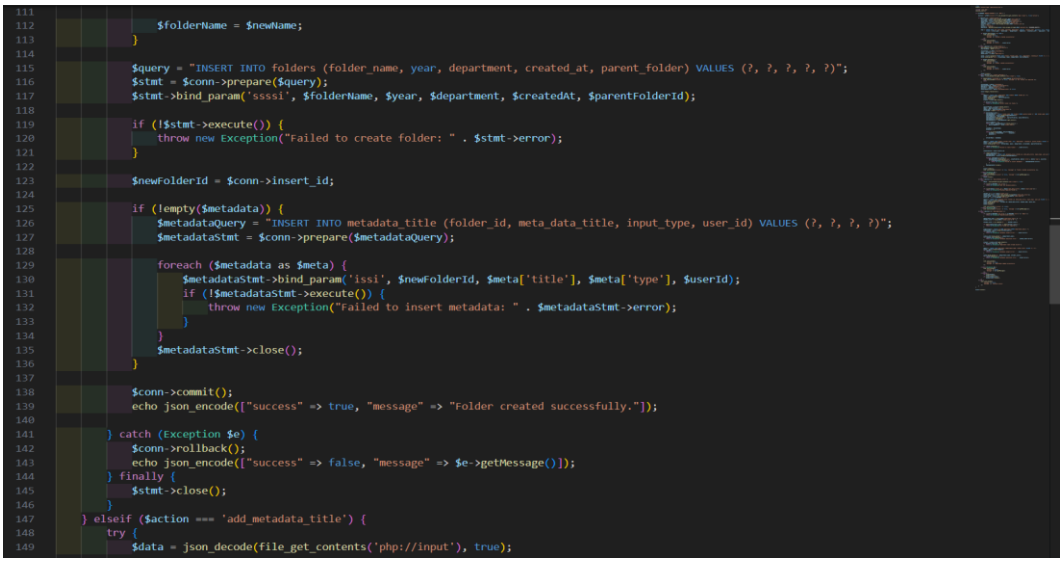
```php
        $query = "SELECT COUNT(*) as count FROM files
                  WHERE original_name = ? AND folder_id = ?";
        $stmt = $conn->prepare($query);
        $stmt->bind_param("si", $name, $folder_id);

    } else {
        throw new Exception('Invalid type specified');
    }

    $stmt->execute();
    $result = $stmt->get_result();
    $row = $result->fetch_assoc();

    echo json_encode([
        'success' => true,
        'exists' => ($row['count'] > 0)
    ]);

} catch (Exception $e) {
    echo json_encode([
        'success' => false,
        'message' => $e->getMessage()
    ]);
}

$conn->close();
?>
```

create.php

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

include 'conn.php';
session_start();

if ($_SERVER['REQUEST_METHOD'] === 'POST') {

    $action = $_POST['action'] ?? json_decode(file_get_contents('php://input'), true)['action'];

    if ($action === 'create_account') {
        $full_name = $conn->real_escape_string($_POST['full_name']);
        $username = $conn->real_escape_string($_POST['username']);
        $department = $conn->real_escape_string($_POST['department']);
        $address = $conn->real_escape_string($_POST['address']);
        $contact_info = $conn->real_escape_string($_POST['contact_info']);
        $role = "User";
        $status = 'enabled';
        $password = password_hash($conn->real_escape_string($_POST['password']), PASSWORD_BCRYPT);

        $sql = "INSERT INTO user (full_name, username, department, address, contact_info, password, role, status, created_at)
                VALUES ('$full_name', '$username', '$department', '$address', '$contact_info', '$password', '$role', '$status', NOW())";

        if ($conn->query($sql) === TRUE) {
            echo json_encode([
                'success' => true,
                'message' => 'Account created successfully!'
            ]);
        } else {
            echo json_encode([
                'success' => false,
                'message' => "Error: " . $conn->error
            ]);
        }
    } elseif ($action === 'create_folder') {
        date_default_timezone_set('Asia/Manila');
        $currentDate = date('Y-m-d');
```

```php
    $folderName = $_POST['folderName'];
    $year = $_POST['year'];
    $department = $_POST['department'];

    $stmt = $conn->prepare("INSERT INTO folders (folder_name, year, department, created_at) VALUES (?, ?, ?, ?)");
    $stmt->bind_param("siss", $folderName, $year, $department, $currentDate);

    if ($stmt->execute()) {
        echo json_encode([
            'success' => true,
            'message' => 'Folder created successfully!'
        ]);
    } else {
        echo json_encode([
            'success' => false,
            'message' => "Error: " . $stmt->error
        ]);
    }

    $stmt->close();
} elseif ($action === 'create_subfolder') {
    $data = json_decode(file_get_contents('php://input'), true);

    if (empty($data['folderName']) || empty($data['parentFolderId'])) {
        echo json_encode(["success" => false, "message" => "All fields are required."]);
        exit;
    }

    $folderName = $data['folderName'];
    $parentFolderId = $data['parentFolderId'];
    $metadata = $data['metadata'] ?? [];
    $userId = $_SESSION['user_id'];
    $handleDuplicate = $data['handleDuplicate'] ?? false;

    $conn->begin_transaction();
```

```php
    try {
        $query = "SELECT year, department FROM folders WHERE folder_id = ?";
        $stmt = $conn->prepare($query);
        $stmt->bind_param('i', $parentFolderId);
        $stmt->execute();
        $result = $stmt->get_result();

        if ($result->num_rows === 0) {
            throw new Exception("Parent folder not found.");
        }

        $parentFolder = $result->fetch_assoc();
        $year = $parentFolder['year'];
        $department = $parentFolder['department'];
        $createdAt = date('Y-m-d H:i:s');

        if ($handleDuplicate) {
            $baseNameQuery = "SELECT folder_name FROM folders WHERE parent_folder = ? AND folder_name LIKE ?";
            $baseNameStmt = $conn->prepare($baseNameQuery);
            $searchPattern = $folderName . '%';
            $baseNameStmt->bind_param('is', $parentFolderId, $searchPattern);
            $baseNameStmt->execute();
            $baseNameResult = $baseNameStmt->get_result();

            $existingNames = [];
            while ($row = $baseNameResult->fetch_assoc()) {
                $existingNames[] = $row['folder_name'];
            }

            $newName = $folderName;
            $counter = 2;

            while (in_array($newName, $existingNames)) {
                $newName = $folderName . ' ' . $counter;
                $counter++;
            }
```



```php
            $folderName = $newName;
        }

        $query = "INSERT INTO folders (folder_name, year, department, created_at, parent_folder) VALUES (?, ?, ?, ?, ?)";
        $stmt = $conn->prepare($query);
        $stmt->bind_param('ssssi', $folderName, $year, $department, $createdAt, $parentFolderId);

        if (!$stmt->execute()) {
            throw new Exception("Failed to create folder: " . $stmt->error);
        }

        $newFolderId = $conn->insert_id;

        if (!empty($metadata)) {
            $metadataQuery = "INSERT INTO metadata_title (folder_id, meta_data_title, input_type, user_id) VALUES (?, ?, ?, ?)";
            $metadataStmt = $conn->prepare($metadataQuery);

            foreach ($metadata as $meta) {
                $metadataStmt->bind_param('issi', $newFolderId, $meta['title'], $meta['type'], $userId);
                if (!$metadataStmt->execute()) {
                    throw new Exception("Failed to insert metadata: " . $metadataStmt->error);
                }
            }
            $metadataStmt->close();
        }

        $conn->commit();
        echo json_encode(["success" => true, "message" => "Folder created successfully."]);

    } catch (Exception $e) {
        $conn->rollback();
        echo json_encode(["success" => false, "message" => $e->getMessage()]);
    } finally {
        $stmt->close();
    }
} elseif ($action === 'add_metadata_title') {
    try {
        $data = json_decode(file_get_contents('php://input'), true);
```

```php
        $data = json_decode(file_get_contents('php://input'), true);

        if (!isset($_SESSION['user_id'])) {
            throw new Exception('User not authenticated');
        }

        if (!isset($data['folder_id'], $data['meta_data_title'], $data['input_type'])) {
            throw new Exception('Missing required fields');
        }

        $folder_id = (int) $data['folder_id'];
        $meta_data_title = $conn->real_escape_string($data['meta_data_title']);
        $input_type = $conn->real_escape_string($data['input_type']);
        $user_id = (int) $_SESSION['user_id'];

        $query = "INSERT INTO metadata_title (folder_id, meta_data_title, input_type, user_id) VALUES (?, ?, ?, ?)";
        $stmt = $conn->prepare($query);
        $stmt->bind_param("issi", $folder_id, $meta_data_title, $input_type, $user_id);
        $stmt->execute();
        $stmt->close();

        echo json_encode(['success' => true]);
    } catch (Exception $e) {
        echo json_encode(['success' => false, 'message' => $e->getMessage()]);
    }
} elseif ($action === 'add_department') {
    try {
        if (!isset($_SESSION['user_id']) || $_SESSION['role'] !== 'Admin') {
            throw new Exception('Unauthorized access');
        }

        $department_name = trim($_POST['department_name'] ?? '');
        $folder_color = trim($_POST['folder_color'] ?? '');

        $folder_color = str_replace('#', '', $folder_color);

        if (empty($department_name) || empty($folder_color)) {
            throw new Exception('All fields are required');
        }
```

```php
        }

        $check_query = "SELECT id FROM department WHERE department_name = ?";
        $check_stmt = $conn->prepare($check_query);
        if (!$check_stmt) {
            throw new Exception('Database prepare error: ' . $conn->error);
        }

        $check_stmt->bind_param("s", $department_name);
        if (!$check_stmt->execute()) {
            throw new Exception('Database execution error: ' . $check_stmt->error);
        }

        $result = $check_stmt->get_result();
        if ($result->num_rows > 0) {
            throw new Exception('Department name already exists');
        }

        $query = "INSERT INTO department (department_name, folder_color) VALUES (?, ?)";
        $stmt = $conn->prepare($query);
        if (!$stmt) {
            throw new Exception('Database prepare error: ' . $conn->error);
        }

        $stmt->bind_param("ss", $department_name, $folder_color);
        if (!$stmt->execute()) {
            throw new Exception('Database execution error: ' . $stmt->error);
        }

        echo json_encode([
            'success' => true,
            'message' => 'Department added successfully'
        ]);
```

```php
    } catch (Exception $e) {
        echo json_encode([
            'success' => false,
            'message' => $e->getMessage()
        ]);
    } finally {
        if (isset($stmt))
            $stmt->close();
        if (isset($check_stmt))
            $check_stmt->close();
    }
} else {
    echo json_encode([
        'success' => false,
        'message' => 'Invalid action!'
    ]);
}
}

$conn->close();
```

db_functions.php

```php
<?php

function getFilePath($conn, $folder_id)
{
    $path = [];

    while ($folder_id) {
        $folder_query = "SELECT folder_name, parent_folder FROM folders WHERE folder_id = ?";
        $stmt = mysqli_prepare($conn, $folder_query);
        mysqli_stmt_bind_param($stmt, 'i', $folder_id);
        mysqli_stmt_execute($stmt);
        $result = mysqli_stmt_get_result($stmt);

        if ($folder = mysqli_fetch_assoc($result)) {
            array_unshift($path, $folder['folder_name']);
            $folder_id = $folder['parent_folder'];
        } else {
            break;
        }
    }

    return implode(' / ', $path);
}

function getMetadataTitles($conn)
{
    $metadata_titles = [];
    $title_query = "SELECT DISTINCT meta_data_title, metadata_id FROM metadata_title";
    $title_result = mysqli_query($conn, $title_query);
    while ($row = mysqli_fetch_assoc($title_result)) {
        $metadata_titles[$row['metadata_id']] = $row['meta_data_title'];
    }
    return $metadata_titles;
}

function getDepartments($conn)
{
    $departments = [];
    $dept_query = "SELECT DISTINCT department_name FROM department WHERE department_name != 'admin'";
    $dept_result = mysqli_query($conn, $dept_query);
    while ($row = mysqli_fetch_assoc($dept_result)) {
        $departments[] = $row['department_name'];
    }
    return $departments;
}

function getFiles($conn, $status, $user_id, $filters = [], $is_admin = false)
{
    $file_query = "
        SELECT f.file_id, f.original_name, f.upload_date, f.author, f.folder_id, f.user_id,
               f.file_name, f.department, f.remarks, u.full_name
        FROM files f
        JOIN user u ON f.user_id = u.id
        WHERE f.status = ?
    ";

    if (!$is_admin) {
        $file_query .= " AND f.user_id = ?";
    }

    $where_clauses = [];
    $params = [$status];

    if (!$is_admin) {
        $params[] = $user_id;
    }

    if (isset($filters['fromDate']) && !empty($filters['fromDate'])) {
        $where_clauses[] = "f.upload_date >= ?";
        $params[] = $filters['fromDate'];
    }

    if (isset($filters['toDate']) && !empty($filters['toDate'])) {
        $where_clauses[] = "f.upload_date <= ?";
        $params[] = $filters['toDate'];
    }

    if (isset($filters['department']) && !empty($filters['department'])) {
        $where_clauses[] = "f.department = ?";
        $params[] = $filters['department'];
    }

    if (!empty($where_clauses)) {
        $file_query .= " AND " . implode(" AND ", $where_clauses);
    }

    $stmt = mysqli_prepare($conn, $file_query);
    if (!empty($params)) {
        $types = str_repeat('s', count($params));
        mysqli_stmt_bind_param($stmt, $types, ...$params);
    }
    mysqli_stmt_execute($stmt);
    return mysqli_stmt_get_result($stmt);
}

function getFileMetadata($conn, $file_id)
{
    $metadata_query = "
        SELECT mt.meta_data_title, md.meta_data_description
        FROM metadata_description md
        JOIN metadata_title mt ON md.title_id = mt.metadata_id
        WHERE md.file_id = ?
    ";
    $stmt = mysqli_prepare($conn, $metadata_query);
    mysqli_stmt_bind_param($stmt, 's', $file_id);
    mysqli_stmt_execute($stmt);
    return mysqli_stmt_get_result($stmt);
}
```

delete.php

```php
<?php
error_reporting(E_ALL);
ini_set('log_errors', 1);
ini_set('error_log', 'php_errors.log');

require_once 'conn.php';

session_start();

header('Content-Type: application/json');

try {
    $data = json_decode(file_get_contents('php://input'), true);

    if (!isset($data['type'])) {
        throw new Exception('Missing required field: type');
    }

    $type = $data['type'];

    if ($type === 'metadata_title') {
        // for deleting medata title (admin | manage | folders (for subfolders only))
        if (!isset($data['metadata_id'])) {
            throw new Exception('Missing required field: metadata_id');
        }

        $metadata_id = (int) $data['metadata_id'];

        $query = "DELETE FROM metadata_title WHERE metadata_id = ?";
        $stmt = $conn->prepare($query);
        $stmt->bind_param("i", $metadata_id);
        $stmt->execute();
        $stmt->close();

        echo json_encode(['success' => true]);
    } elseif ($type === 'folder_status') {
        // for updating folder status (activate/deactivate) recursively
        if (!isset($data['folder_id']) || !isset($data['status'])) {
            throw new Exception('Missing required fields: folder_id or status');
```

```php
            throw new Exception('Missing required fields: folder_id or status');
        }

        $folder_id = $data['folder_id'];
        $new_status = $data['status']; // Will be 'active' or 'inactive'

        try {
            // Begin transaction
            $conn->begin_transaction();

            // Update folder status recursively
            $stats = updateFolderStatusRecursively($conn, $folder_id, $new_status);

            // Commit transaction
            $conn->commit();

            echo json_encode([
                'success' => true,
                'message' => "Folder and its subfolders updated to $new_status",
                'folders_updated' => $stats['folders_updated']
            ]);
        } catch (Exception $e) {
            // Rollback transaction in case of error
            $conn->rollback();

            error_log("Error in delete.php: " . $e->getMessage());
            echo json_encode([
                'success' => false,
                'message' => 'An error occurred: ' . $e->getMessage()
            ]);
        }
    } elseif ($type === 'file') {
        // handle file deletion ----
        if (!isset($_SESSION['user_id'])) {
            throw new Exception('Unauthorized access');
        }
```

```php
        if (!isset($data['file_id'])) {
            throw new Exception('File ID is required');
        }

        $fileId = $data['file_id'];
        $userId = $_SESSION['user_id'];

        // check ownership ----
        $checkOwnership = $conn->prepare("SELECT file_name FROM files WHERE file_id = ? AND user_id = ?");
        $checkOwnership->bind_param("ii", $fileId, $userId);
        $checkOwnership->execute();
        $result = $checkOwnership->get_result();

        if ($result->num_rows === 0) {
            throw new Exception('Unauthorized to delete this file');
        }

        $fileData = $result->fetch_assoc();
        $fileName = $fileData['file_name'];
        $filePath = "../uploads/" . $fileName;

        // start transaction ----
        $conn->begin_transaction();

        try {
            // delete file record ----
            $deleteFile = $conn->prepare("DELETE FROM files WHERE file_id = ? AND user_id = ?");
            $deleteFile->bind_param("ii", $fileId, $userId);
            $deleteFile->execute();

            // delete physical file ----
            if (file_exists($filePath)) {
                unlink($filePath);
            }

            // commit transaction ----
            $conn->commit();
```

```
            echo json_encode(['success' => true]);
        } catch (Exception $e) {
            // rollback transaction on error
            $conn->rollback();
            throw new Exception('Database error occurred');
        }
    } else {
        throw new Exception('Invalid type specified');
    }
} catch (Exception $e) {
    // rollback transaction if still in progress ----
    if (isset($conn) && $conn->in_transaction) {
        $conn->rollback();
    }

    echo json_encode(['success' => false, 'message' => $e->getMessage()]);
}

$conn->close();

/**
 * recursively update the status of a folder and its subfolders.
 *
 * @param mysqli $conn database conn.
 * @param int $folder_id folder ID.
 * @param string $status status to set ('active' / 'inactive').
 * @return array stats about the folders updated.
 */
function updateFolderStatusRecursively($conn, $folder_id, $status)
{
    $stats = [
        'folders_updated' => 0
    ];

    try {
        // update the current folder's status
        $update_folder = $conn->prepare("UPDATE folders SET status = ? WHERE folder_id = ?");
        $update_folder->bind_param("si", $status, $folder_id);
        $update_folder->execute();
        $update_folder->close();
        $stats['folders_updated']++;

        // find and update subfolders recursively
        $get_children = $conn->prepare("SELECT folder_id FROM folders WHERE parent_folder = ?");
        $get_children->bind_param("i", $folder_id);
        $get_children->execute();
        $children_result = $get_children->get_result();

        while ($child = $children_result->fetch_assoc()) {
            $child_stats = updateFolderStatusRecursively($conn, $child['folder_id'], $status);
            $stats['folders_updated'] += $child_stats['folders_updated'];
        }
        $get_children->close();

        return $stats;
    } catch (Exception $e) {
        error_log("Error in updateFolderStatusRecursively: " . $e->getMessage());
        throw $e; // re-throw to be caught in the calling function
    }
}
?>
```

folder_functions.php

```
<?php
// folder functions (used for: get_folders.php)
require_once 'conn.php';

function getFolderColor($department)
{
    global $conn;

    $stmt = $conn->prepare("SELECT folder_color FROM department WHERE LOWER(department_name) = LOWER(?)");
    $stmt->bind_param("s", $department);
    $stmt->execute();
    $stmt->bind_result($folderColor);

    if ($stmt->fetch()) {
        $stmt->close();
        return '#' . $folderColor;
    } else {
        $stmt->close();
        return '#6C757d';
    }
}

function getFileIcon($extension)
{
    $icons = [
        'pdf' => 'bi-file-pdf',
        'doc' => 'bi-file-word',
        'docx' => 'bi-file-word',
        'xls' => 'bi-file-excel',
        'xlsx' => 'bi-file-excel',
        'ppt' => 'bi-file-ppt',
        'pptx' => 'bi-file-ppt',
        'txt' => 'bi-file-text',
        'rtf' => 'bi-file-text'
    ];

    return isset($icons[strtolower($extension)]) ? $icons[strtolower($extension)] : 'bi-file';
}
?>
```

get_folders.php

```php
<?php
session_start();


require_once 'conn.php';
require_once 'folder_functions.php';

$parent_folder_id = isset($_GET['folder_id']) ? intval($_GET['folder_id']) : null;
$user_department = $_SESSION['department'];
$role = $_SESSION['role'];

if ($user_department === 'admin') {
    // for admin side folders
    if ($parent_folder_id) {
        $query = "SELECT * FROM folders WHERE parent_folder = ?";
        $stmt = $conn->prepare($query);
        $stmt->bind_param("i", $parent_folder_id);
        $stmt->execute();
        $folders_result = $stmt->get_result();

        $files_query = "SELECT f.*, fld.row_no, fld.column_no, fld.status as folder_status
        FROM files f
        JOIN folders fld ON f.folder_id = fld.folder_id
        WHERE f.folder_id = ? AND f.status = 'approved'";
        $files_stmt = $conn->prepare($files_query);
        $files_stmt->bind_param("i", $parent_folder_id);
        $files_stmt->execute();
        $files_result = $files_stmt->get_result();

        $parent_query = "SELECT folder_name FROM folders WHERE folder_id = ?";
        $parent_stmt = $conn->prepare($parent_query);
        $parent_stmt->bind_param("i", $parent_folder_id);
        $parent_stmt->execute();
        $parent_result = $parent_stmt->get_result();
        $parent_folder = $parent_result->fetch_assoc();
    } else {
        $query = "SELECT * FROM folders WHERE (parent_folder IS NULL OR parent_folder = 0)";
        $stmt = $conn->prepare($query);
        $stmt->execute();
```

```php
        $stmt->execute();
        $folders_result = $stmt->get_result();

        $files_result = false;
    }

    $years_query = "SELECT DISTINCT year FROM folders ORDER BY year DESC";
    $years_stmt = $conn->prepare($years_query);
    $years_stmt->execute();
    $years_result = $years_stmt->get_result();
} else {
    // for user folders
    if ($parent_folder_id) {
        // $parent_details_query = "SELECT row_no, column_no FROM folders WHERE folder_id = ?";
        // $parent_details_stmt = $conn->prepare($parent_details_query);
        // $parent_details_stmt->bind_param("i", $parent_folder_id);
        // $parent_details_stmt->execute();
        // $parent_details_result = $parent_details_stmt->get_result();
        // $parent_details = $parent_details_result->fetch_assoc();
        // $parent_row = $parent_details['row_no'] ?? 0;
        // $parent_col = $parent_details['column_no'] ?? 0;

        function getTopMostParentDetails($conn, $folder_id)
        {
            while ($folder_id) {
                $query = "SELECT parent_folder, row_no, column_no FROM folders WHERE folder_id = ?";
                $stmt = $conn->prepare($query);
                $stmt->bind_param("i", $folder_id);
                $stmt->execute();
                $result = $stmt->get_result();
                $folder = $result->fetch_assoc();

                if (!$folder || !$folder['parent_folder']) {
                    return [
                        'row_no' => $folder['row_no'] ?? 0,
                        'column_no' => $folder['column_no'] ?? 0
                    ];
                }
```

```php
                // move to the next parent in the hierarchy
                $folder_id = $folder['parent_folder'];
            }
            return ['row_no' => 0, 'column_no' => 0]; // default data / values
        }

        $parent_details = getTopMostParentDetails($conn, $parent_folder_id);
        $parent_row = $parent_details['row_no'];
        $parent_col = $parent_details['column_no'];

        $query = "SELECT * FROM folders WHERE parent_folder = ? AND department = ?";
        $stmt = $conn->prepare($query);
        $stmt->bind_param("is", $parent_folder_id, $user_department);
        $stmt->execute();
        $folders_result = $stmt->get_result();

        $files_query = "SELECT f.*, fld.row_no, fld.column_no, fld.status as folder_status
        FROM files f
        JOIN folders fld ON f.folder_id = fld.folder_id
        WHERE f.folder_id = ? AND f.status = 'approved'";
        $files_stmt = $conn->prepare($files_query);
        $files_stmt->bind_param("i", $parent_folder_id);
        $files_stmt->execute();
        $files_result = $files_stmt->get_result();

        $parent_query = "SELECT folder_name FROM folders WHERE folder_id = ?";
        $parent_stmt = $conn->prepare($parent_query);
        $parent_stmt->bind_param("i", $parent_folder_id);
        $parent_stmt->execute();
        $parent_result = $parent_stmt->get_result();
        $parent_folder = $parent_result->fetch_assoc();
    } else {
        $query = "SELECT * FROM folders WHERE (parent_folder IS NULL OR parent_folder = 0) AND department = ?";
        $stmt = $conn->prepare($query);
        $stmt->bind_param("s", $user_department);
        $stmt->execute();
        $folders_result = $stmt->get_result();
```

```
        $folders_result = $stmt->get_result();

        $files_result = false;
    }

    $years_query = "SELECT DISTINCT year FROM folders WHERE department = ? ORDER BY year DESC";
    $years_stmt = $conn->prepare($years_query);
    $years_stmt->bind_param("s", $user_department);
    $years_stmt->execute();
    $years_result = $years_stmt->get_result();
}


?>
```

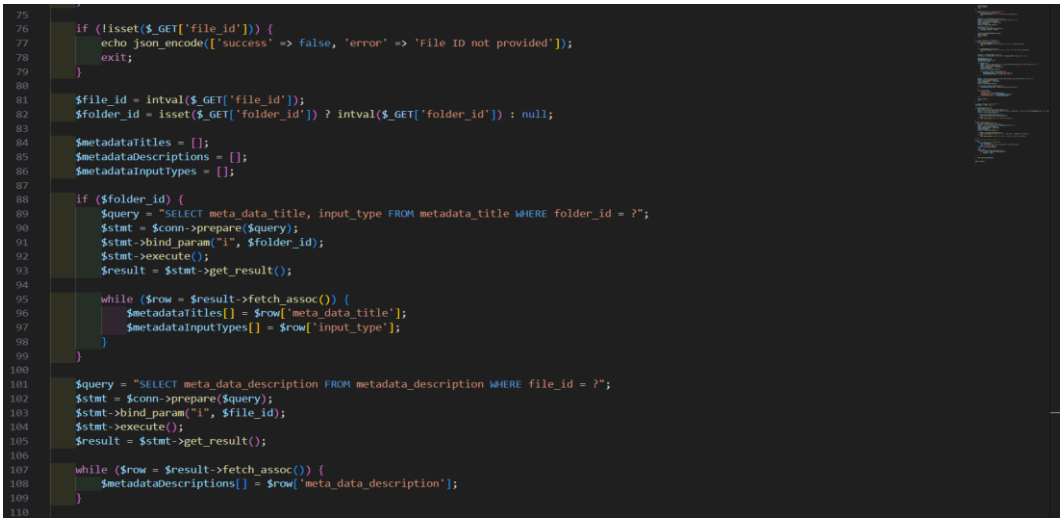
get_metadata.php

```
<?php
session_start();
include 'conn.php';

if (isset($_GET['file_id'])) {
    $file_id = $_GET['file_id'];
    $metadata_query = "
        SELECT mt.meta_data_title, md.meta_data_description, md.title_id, mt.input_type
        FROM metadata_description md
        JOIN metadata_title mt ON md.title_id = mt.metadata_id
        WHERE md.file_id = ?
    ";
    $stmt = mysqli_prepare($conn, $metadata_query);
    mysqli_stmt_bind_param($stmt, 'i', $file_id);
    mysqli_stmt_execute($stmt);
    $result = mysqli_stmt_get_result($stmt);

    while ($metadata = mysqli_fetch_assoc($result)) {
        $input_type = isset($metadata['input_type']) && in_array($metadata['input_type'],
            ['text', 'number', 'date', 'time', 'month', 'year', 'email', 'tel', 'datetime-local'])
            ? $metadata['input_type'] : 'text';

        echo "<div class='mb-3'>
                <label for='metadata_{$metadata['title_id']}' class='form-label'>{$metadata['meta_data_title']}</label>
                <input type='{$input_type}' class='form-control' id='metadata_{$metadata['title_id']}' name='metadata_descriptions[{$metadata
                ['title_id']}]' value='{$metadata['meta_data_description']}'>
            </div>";
    }
}
?>
```

get.php

```
<?php

header('Content-Type: application/json');
require_once 'conn.php';
session_start();

// allowed tables ----
$tables = ['user', 'metadata', 'metadata_titles'];

// ensure the requested table exists ----
if (!isset($_GET['table']) || !in_array($_GET['table'], $tables)) {
    echo json_encode(['error' => 'Invalid or missing table parameter']);
    exit;
}

// new endpoint for fetch_metadata_title functionality (removed fetch_metadata_title.php) ----
if (isset($_GET['fetch_metadata_title']) && isset($_GET['folder_id'])) {
    if (!isset($_SESSION['user_id']) || !isset($_SESSION['department'])) {
        echo json_encode(['success' => false, 'message' => 'User not authenticated or department not set']);
        exit;
    }

    $folder_id = $_GET['folder_id'];
    $query = "SELECT metadata_id, meta_data_title FROM metadata_title WHERE folder_id = ?";
    $stmt = $conn->prepare($query);
    $stmt->bind_param("i", $folder_id);
    $stmt->execute();
    $result = $stmt->get_result();

    $metadata_titles = [];
    while ($row = $result->fetch_assoc()) {
        $metadata_titles[] = $row;
    }

    echo json_encode([
        'success' => true,
        'metadata_titles' => $metadata_titles
    ]);
```

```
    $stmt->close();
    $conn->close();
    exit;
}

if ($_GET['table'] === 'metadata_titles') {
    if (!isset($_GET['folder_id'])) {
        echo json_encode(['error' => 'Folder ID not provided']);
        exit;
    }

    $folder_id = intval($_GET['folder_id']);
    $query = "SELECT * FROM metadata_title WHERE folder_id = ?";
    $stmt = $conn->prepare($query);
    $stmt->bind_param("i", $folder_id);
    $stmt->execute();
    $result = $stmt->get_result();

    $metadata_titles = [];
    while ($row = $result->fetch_assoc()) {
        $metadata_titles[] = $row;
    }

    echo json_encode($metadata_titles);
    $stmt->close();
    $conn->close();
    exit;
}

// check if metadata is requested ----
if ($_GET['table'] === 'metadata') {
    if (!isset($_SESSION['user_id'])) {
        echo json_encode(['success' => false, 'error' => 'Unauthorized']);
        exit;
    }
```



```
    if (!isset($_GET['file_id'])) {
        echo json_encode(['success' => false, 'error' => 'File ID not provided']);
        exit;
    }

    $file_id = intval($_GET['file_id']);
    $folder_id = isset($_GET['folder_id']) ? intval($_GET['folder_id']) : null;

    $metadataTitles = [];
    $metadataDescriptions = [];
    $metadataInputTypes = [];

    if ($folder_id) {
        $query = "SELECT meta_data_title, input_type FROM metadata_title WHERE folder_id = ?";
        $stmt = $conn->prepare($query);
        $stmt->bind_param("i", $folder_id);
        $stmt->execute();
        $result = $stmt->get_result();

        while ($row = $result->fetch_assoc()) {
            $metadataTitles[] = $row['meta_data_title'];
            $metadataInputTypes[] = $row['input_type'];
        }
    }

    $query = "SELECT meta_data_description FROM metadata_description WHERE file_id = ?";
    $stmt = $conn->prepare($query);
    $stmt->bind_param("i", $file_id);
    $stmt->execute();
    $result = $stmt->get_result();

    while ($row = $result->fetch_assoc()) {
        $metadataDescriptions[] = $row['meta_data_description'];
    }
```

```

    echo json_encode([
        'success' => true,
        'metadataTitles' => $metadataTitles,
        'metadataDescriptions' => $metadataDescriptions,
        'metadataInputTypes' => $metadataInputTypes
    ]);

    $conn->close();
    exit;
}

// handle requests for user table ----
$tableName = $_GET['table'];

// Fetching a single user
if (isset($_GET['id'])) {
    $id = $conn->real_escape_string($_GET['id']);
    $query = "SELECT id, full_name, username, address, department, contact_info FROM $tableName WHERE id = '$id'";
    $result = $conn->query($query);

    if ($result && $result->num_rows > 0) {
        echo json_encode($result->fetch_assoc());
    } else {
        echo json_encode(['error' => 'User not found']);
    }
}
// get user full name ----
elseif (isset($_GET['user_id'])) {
    $user_id = intval($_GET['user_id']);
    $query = "SELECT full_name FROM $tableName WHERE id = ?";
    $stmt = $conn->prepare($query);
    $stmt->bind_param("i", $user_id);
    $stmt->execute();
    $result = $stmt->get_result();
```

```php
    if ($user = $result->fetch_assoc()) {
        echo json_encode(['success' => true, 'full_name' => $user['full_name']]);
    } else {
        echo json_encode(['success' => false, 'error' => 'User not found']);
    }
}
// get all users except the admin ----
else {
    $sql = "SELECT u.*, d.folder_color
        FROM $tableName u
        LEFT JOIN department d ON u.department = d.department_name
        WHERE u.role != 'Admin'";
    $result = $conn->query($sql);

    $data = [];
    if ($result && $result->num_rows > 0) {
        while ($row = $result->fetch_assoc()) {
            $data[] = $row;
        }
    }

    echo json_encode($data);
}

$conn->close();
?>
```

login.php

```php
<?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'];

            $redirect = $user['role'] === 'Admin'
                ? 'pages/admin/dashboard.php'
                : 'pages/personnel/home.php';

            echo json_encode([
                'success' => true,
                'message' => 'Login successful!',
                'redirect' => $redirect
            ]);
        } else {
            echo json_encode([
                'success' => false,
                'message' => 'Invalid password!'
            ]);
        }
    } else {
        echo json_encode([
            'success' => false,
            'message' => 'Username not found!'
        ]);
    }
}

$conn->close();
?>
```

logout.php

```php
<?php
session_start();

session_unset();

session_destroy();

header("Location: ../");
exit;
?>
```

reupload_file.php

```php
<?php
session_start();
include 'conn.php';

// error reporting  || debug ---- (apr 2, 2025)
error_reporting(E_ALL & ~E_NOTICE);

if (!isset($_SESSION['user_id'])) {
    echo json_encode(['success' => false, 'message' => 'Unauthorized access']);
    exit;
}

if ($_SERVER['REQUEST_METHOD'] !== 'POST' || !isset($_POST['reupload_file'])) {
    echo json_encode(['success' => false, 'message' => 'Invalid request']);
    exit;
}

$file_id = $_POST['file_id'] ?? null;
$file_name = $_POST['file_name'] ?? null;
$file_owner = $_POST['file_owner'] ?? null;

if (!$file_id || !$file_name || !$file_owner) {
    echo json_encode(['success' => false, 'message' => 'Required fields are missing']);
    exit;
}

$current_file_query = "SELECT file_name, folder_id FROM files WHERE file_id = ?";
$stmt = $conn->prepare($current_file_query);
$stmt->bind_param("i", $file_id);
$stmt->execute();
$result = $stmt->get_result();
$current_file = $result->fetch_assoc();

if (!$current_file) {
    echo json_encode(['success' => false, 'message' => 'File not found']);
    exit;
}
```

```php

$new_filename = $current_file['file_name'];
$upload_success = true;

if (isset($_FILES['new_file']) && $_FILES['new_file']['error'] === UPLOAD_ERR_OK) {
    $file = $_FILES['new_file'];

    $file_type = mime_content_type($file['tmp_name']);
    if ($file_type !== 'application/pdf') {
        echo json_encode(['success' => false, 'message' => 'Only PDF files are allowed']);
        exit;
    }

    $file_ext = pathinfo($file['name'], PATHINFO_EXTENSION);
    $new_filename = uniqid() . '.' . $file_ext;
    $upload_path = '../../uploads/' . $new_filename;

    if (!move_uploaded_file($file['tmp_name'], $upload_path)) {
        $upload_success = false;
    } else {
        $old_file_path = '../../uploads/' . $current_file['file_name'];
        if (file_exists($old_file_path)) {
            unlink($old_file_path);
        }
    }
}

if (!$upload_success) {
    echo json_encode(['success' => false, 'message' => 'Failed to upload new file']);
    exit;
}
```

```php

$update_query = "UPDATE files SET
    original_name = ?,
    file_name = ?,
    author = ?,
    upload_date = NOW(),
    status = 'pending',
    remarks = ''
    WHERE file_id = ?";

$stmt = $conn->prepare($update_query);
$stmt->bind_param(
    "sssi",
    $file_name,
    $new_filename,
    $file_owner,
    $file_id
);

if ($stmt->execute()) {
    if (isset($_POST['metadata_descriptions']) && is_array($_POST['metadata_descriptions'])) {
        $metadata_descriptions = $_POST['metadata_descriptions'];

        foreach ($metadata_descriptions as $metadata_id => $description) {
            if (is_array($description)) {
                $description = implode(', ', $description);
            }

            $update_metadata_query = "UPDATE metadata_description SET meta_data_description = ?
                                      WHERE file_id = ? AND title_id = ?";
            $meta_stmt = $conn->prepare($update_metadata_query);
            $meta_stmt->bind_param('sii', $description, $file_id, $metadata_id);

            if (!$meta_stmt->execute()) {
                error_log("Failed to update metadata for file_id: $file_id, title_id: $metadata_id");
            }
            $meta_stmt->close();
        }
```

```
        }
    }

    // clear output buffer | send JSON res.
    ob_clean();
    echo json_encode(['success' => true, 'message' => 'File reuploaded successfully and sent for approval']);
} else {
    ob_clean();
    echo json_encode(['success' => false, 'message' => 'Failed to update file record']);
}

$stmt->close();
$conn->close();
?>
```

search.php

```
<?php
require_once 'conn.php';
session_start();

header('Content-Type: application/json');

$userDepartment = $_SESSION['department'];
$userId = $_SESSION['user_id'];
$yearFilter = isset($_GET['year']) && $_GET['year'] !== 'all' ? $_GET['year'] : null;

// get department colors
$departmentColors = [];
$colorQuery = "SELECT department_name, CONCAT('#', folder_color) as color FROM department";
$colorResult = $conn->query($colorQuery);
while ($row = $colorResult->fetch_assoc()) {
    $departmentColors[$row['department_name']] = $row['color'];
}

if ($userDepartment === 'admin') {
    try {
        $searchTerm = isset($_GET['term']) ? '%' . $_GET['term'] . '%' : '';

        $response = [
            'folders' => [],
            'files' => [],
            'departmentColors' => $departmentColors
        ];

        $folderQuery = "SELECT * FROM folders WHERE folder_name LIKE ?";
        $params = array($searchTerm);
        $types = "s";

        if ($yearFilter) {
            $folderQuery .= " AND year = ?";
            $params[] = $yearFilter;
            $types .= "s";
        }
```

```
        $stmt = $conn->prepare($folderQuery);
        $stmt->bind_param($types, ...$params);
        $stmt->execute();
        $result = $stmt->get_result();
        while ($row = $result->fetch_assoc()) {
            $folderPath = getFolderPath($conn, $row['folder_id']);
            $row['folder_path'] = $folderPath;
            $response['folders'][] = $row;
        }

        $fileQuery = "SELECT f.*, fo.row_no, fo.column_no FROM files f
            INNER JOIN folders fo ON f.folder_id = fo.folder_id
            WHERE (f.original_name LIKE ? OR f.author LIKE ?)
            AND f.status = 'approved'";

        $params = array($searchTerm, $searchTerm);
        $types = "ss";

        if ($yearFilter) {
            $fileQuery .= " AND fo.year = ?";
            $params[] = $yearFilter;
            $types .= "s";
        }

        $stmt = $conn->prepare($fileQuery);
        $stmt->bind_param($types, ...$params);
        $stmt->execute();
        $result = $stmt->get_result();
        while ($row = $result->fetch_assoc()) {
            $folderPath = getFolderPath($conn, $row['folder_id']);
            $row['folder_path'] = $folderPath;
            $row['is_owner'] = ($row['user_id'] == $userId);
            $response['files'][] = $row;
        }
```

```php
        echo json_encode([
            'success' => true,
            'data' => $response,
            'user_id' => $userId
        ]);

    } catch (Exception $e) {
        echo json_encode([
            'success' => false,
            'error' => $e->getMessage()
        ]);
    }
} else {
    try {
        $searchTerm = isset($_GET['term']) ? '%' . $_GET['term'] . '%' : '';
        $yearFilter = isset($_GET['year']) && $_GET['year'] !== 'all' ? $_GET['year'] : null;
        $userDepartment = $_SESSION['department'];

        $response = [
            'folders' => [],
            'files' => [],
            'departmentColors' => $departmentColors
        ];

        // search folders with department and optional year filter
        $folderQuery = "SELECT * FROM folders WHERE folder_name LIKE ? AND department = ?";
        $params = array($searchTerm, $userDepartment);
        $types = "ss";

        if ($yearFilter) {
            $folderQuery .= " AND year = ?";
            $params[] = $yearFilter;
            $types .= "s";
        }
```

```php
        $stmt = $conn->prepare($folderQuery);
        $stmt->bind_param($types, ...$params);
        $stmt->execute();
        $result = $stmt->get_result();
        while ($row = $result->fetch_assoc()) {
            $folderPath = getFolderPath($conn, $row['folder_id']);
            $row['folder_path'] = $folderPath;
            $response['folders'][] = $row;
        }

        // search files with department and optional year filter from parent folder
        $fileQuery = "SELECT f.*, fo.row_no, fo.column_no FROM files f
            INNER JOIN folders fo ON f.folder_id = fo.folder_id
            WHERE (f.original_name LIKE ? OR f.author LIKE ?)
            AND fo.department = ?
            AND f.status = 'approved'";

        $params = array($searchTerm, $searchTerm, $userDepartment);
        $types = "sss";

        if ($yearFilter) {
            $fileQuery .= " AND fo.year = ?";
            $params[] = $yearFilter;
            $types .= "s";
        }

        $stmt = $conn->prepare($fileQuery);
        $stmt->bind_param($types, ...$params);
        $stmt->execute();
        $result = $stmt->get_result();
        while ($row = $result->fetch_assoc()) {
            $folderPath = getFolderPath($conn, $row['folder_id']);
            $row['folder_path'] = $folderPath;
            $row['is_owner'] = ($row['user_id'] == $userId);
            $response['files'][] = $row;
        }
```

```php
        echo json_encode([
            'success' => true,
            'data' => $response,
            'user_id' => $userId
        ]);

    } catch (Exception $e) {
        header('Content-Type: application/json');
        echo json_encode([
            'success' => false,
            'error' => $e->getMessage()
        ]);
    }
}


function getFolderPath($conn, $folder_id)
{
    $path = array();
    $current_id = $folder_id;

    while ($current_id) {
        $query = "SELECT folder_id, folder_name, parent_folder FROM folders WHERE folder_id = ?";
        $stmt = $conn->prepare($query);
        $stmt->bind_param("i", $current_id);
        $stmt->execute();
        $result = $stmt->get_result();

        if ($folder = $result->fetch_assoc()) {
            array_unshift($path, $folder['folder_name']);
            $current_id = $folder['parent_folder'];
        } else {
            break;
        }
    }
```

```php

    if (count($path) === 1 && $current_id === null) {
        return '';
    }

    return implode(' / ', $path);
}

$conn->close();
?>
```

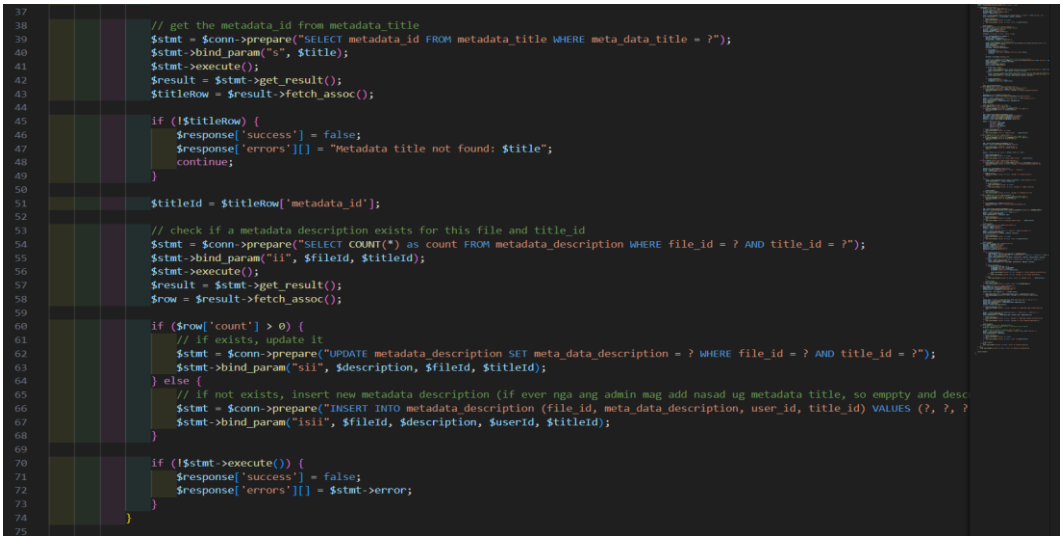
update_folder.php

```php
<?php
error_reporting(E_ALL);
ini_set('display_errors', 1);

include 'conn.php';

header('Content-Type: application/json');

try {
    $data = json_decode(file_get_contents('php://input'), true);

    if (!isset($data['folder_name'], $data['folder_id'])) {
        throw new Exception('Way data doy!');
    }

    // function to also update years for subfolders (occurs when admin updates `year` for parent folders)
    function updateSubfoldersYear($conn, $parent_id, $year)
    {
        $query = "UPDATE folders SET year = ? WHERE parent_folder = ?";
        $stmt = $conn->prepare($query);
        $stmt->bind_param("si", $year, $parent_id);
        $stmt->execute();
        $stmt->close();

        $query2 = "SELECT folder_id FROM folders WHERE parent_folder = ?";
        $stmt2 = $conn->prepare($query2);
        $stmt2->bind_param("i", $parent_id);
        $stmt2->execute();
        $result = $stmt2->get_result();

        while ($row = $result->fetch_assoc()) {
            updateSubfoldersYear($conn, $row['folder_id'], $year);
        }
        $stmt2->close();
    }
```

```php

    $folder_name = $conn->real_escape_string($data['folder_name']);
    $folder_id = (int) $data['folder_id'];
    $year = isset($data['year']) ? $conn->real_escape_string($data['year']) : null;

    if ($year !== null) {
        $query = "UPDATE folders SET folder_name = ?, year = ? WHERE folder_id = ?";
        $stmt = $conn->prepare($query);
        $stmt->bind_param("ssi", $folder_name, $year, $folder_id);
        $stmt->execute();
        $stmt->close();

        updateSubfoldersYear($conn, $folder_id, $year);
    } else {
        $query = "UPDATE folders SET folder_name = ? WHERE folder_id = ?";
        $stmt = $conn->prepare($query);
        $stmt->bind_param("si", $folder_name, $folder_id);
        $stmt->execute();
        $stmt->close();
    }

    $response = [
        'success' => true,
        'message' => 'Folder updated successfully'
    ];
} catch (Exception $e) {
    $response = [
        'success' => false,
        'message' => $e->getMessage()
    ];
}

echo json_encode($response);
$conn->close();
?>
```

update.php

```php
<?php
require_once 'conn.php';
session_start();

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $data = json_decode(file_get_contents('php://input'), true);

    if (isset($data['action'])) {
        if ($data['action'] === 'update_file') {
            // file update (user modal edit file button)
            $fileId = $data['file_id'];
            $originalName = $data['original_name'];
            $author = $data['author'];

            $stmt = $conn->prepare("UPDATE files SET original_name = ?, author = ? WHERE file_id = ?");
            $stmt->bind_param('ssi', $originalName, $author, $fileId);

            if ($stmt->execute()) {
                echo json_encode(['success' => true]);
            } else {
                echo json_encode(['success' => false, 'error' => $stmt->error]);
            }

            $stmt->close();
        } elseif ($data['action'] === 'update_metadata') {
            // metadata update (user modal edit metadata button)
            $fileId = intval($data['fileId']);
            $userId = $_SESSION['user_id'];
            $metadataUpdates = $data['metadata'];

            $response = ['success' => true, 'errors' => []];

            foreach ($metadataUpdates as $update) {
                $title = $update['title'];
                $description = $update['data'];
                $originalData = $update['originalData'];
```

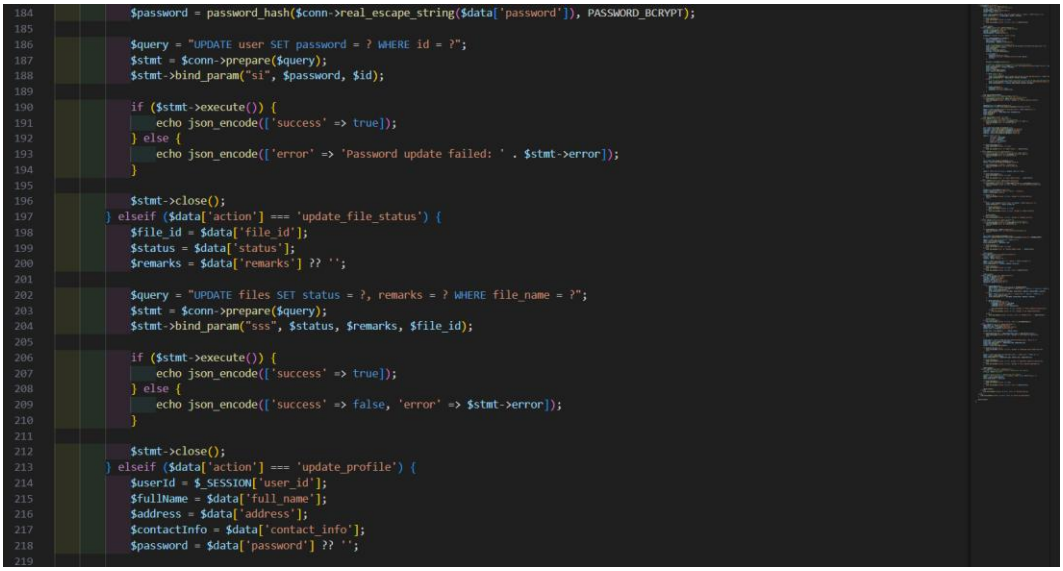


```php

                // get the metadata_id from metadata_title
                $stmt = $conn->prepare("SELECT metadata_id FROM metadata_title WHERE meta_data_title = ?");
                $stmt->bind_param("s", $title);
                $stmt->execute();
                $result = $stmt->get_result();
                $titleRow = $result->fetch_assoc();

                if (!$titleRow) {
                    $response['success'] = false;
                    $response['errors'][] = "Metadata title not found: $title";
                    continue;
                }

                $titleId = $titleRow['metadata_id'];

                // check if a metadata description exists for this file and title_id
                $stmt = $conn->prepare("SELECT COUNT(*) as count FROM metadata_description WHERE file_id = ? AND title_id = ?");
                $stmt->bind_param("ii", $fileId, $titleId);
                $stmt->execute();
                $result = $stmt->get_result();
                $row = $result->fetch_assoc();

                if ($row['count'] > 0) {
                    // if exists, update it
                    $stmt = $conn->prepare("UPDATE metadata_description SET meta_data_description = ? WHERE file_id = ? AND title_id = ?");
                    $stmt->bind_param("sii", $description, $fileId, $titleId);
                } else {
                    // if not exists, insert new metadata description (if ever nga ang admin mag add nasad ug metadata title, so emppty and desc
                    $stmt = $conn->prepare("INSERT INTO metadata_description (file_id, meta_data_description, user_id, title_id) VALUES (?, ?, ?
                    $stmt->bind_param("isii", $fileId, $description, $userId, $titleId);
                }

                if (!$stmt->execute()) {
                    $response['success'] = false;
                    $response['errors'][] = $stmt->error;
                }
            }
```

```php
            echo json_encode($response);
        } elseif ($data['action'] === 'update_metadata_title') {
            // handle metadata title update (admin modal edit metadata button)
            if (!isset($data['metadata_id'], $data['meta_data_title'])) {
                echo json_encode(['success' => false, 'message' => 'Missing required fields']);
                return;
            }

            $metadata_id = (int) $data['metadata_id'];
            $meta_data_title = $conn->real_escape_string($data['meta_data_title']);

            $query = "UPDATE metadata_title SET meta_data_title = ? WHERE metadata_id = ?";
            $stmt = $conn->prepare($query);
            $stmt->bind_param("si", $meta_data_title, $metadata_id);
            $stmt->execute();
            $stmt->close();

            echo json_encode(['success' => true]);
        } elseif ($data['action'] === 'update_user') {
            // handle user update (admin modal edit user button)
            if (!isset($_SESSION['user_id']) || $_SESSION['role'] !== 'Admin') {
                echo json_encode(['error' => 'Unauthorized']);
                return;
            }

            $id = $conn->real_escape_string($data['id']);
            $full_name = $conn->real_escape_string($data['full_name']);
            $username = $conn->real_escape_string($data['username']);
            $address = $conn->real_escape_string($data['address']);
            $contact = $conn->real_escape_string($data['contact']);

            $query = "UPDATE user SET
                        full_name = '$full_name',
                        username = '$username',
                        address = '$address',
                        contact_info = '$contact'
                        WHERE id = '$id'";
```

```
        if ($conn->query($query)) {
            echo json_encode(['success' => true]);
        } else {
            echo json_encode(['error' => 'Update failed: ' . $conn->error]);
        }
    } elseif ($data['action'] === 'update_status') {
        // handle status update (admin accounts list status toggle button)
        if (!isset($_SESSION['user_id']) || $_SESSION['role'] !== 'Admin') {
            echo json_encode(['error' => 'Unauthorized']);
            return;
        }

        $id = $conn->real_escape_string($data['id']);
        $status = $conn->real_escape_string($data['status']);

        if (!in_array($status, ['enabled', 'disabled'])) {
            echo json_encode(['error' => 'Invalid status']);
            return;
        }

        $query = "UPDATE user SET status = '$status' WHERE id = '$id'";

        if ($conn->query($query)) {
            echo json_encode(['success' => true]);
        } else {
            echo json_encode(['error' => 'Status update failed: ' . $conn->error]);
        }
    } elseif ($data['action'] === 'update_folder_position') {
        // folder position update (user folders - row/column)
        if (!isset($data['folder_id']) || !isset($data['field']) || !isset($data['value'])) {
            echo json_encode(['success' => false, 'message' => 'Missing required parameters']);
            return;
        }
```

```
        $folder_id = intval($data['folder_id']);
        $field = ($data['field'] === 'row') ? 'row_no' : 'column_no';
        $value = intval($data['value']);

        if ($value <= 0) {
            echo json_encode(['success' => false, 'message' => 'Invalid value']);
            return;
        }

        try {
            $stmt = $conn->prepare("UPDATE folders SET $field = ? WHERE folder_id = ?");
            $stmt->bind_param('ii', $value, $folder_id);

            if ($stmt->execute()) {
                echo json_encode(['success' => true]);
            } else {
                echo json_encode(['success' => false, 'message' => 'Update failed']);
            }

            $stmt->close();
        } catch (Exception $e) {
            echo json_encode(['success' => false, 'message' => 'Database error']);
        }
    } elseif ($data['action'] === 'update_password') {
        // user password update (admin, accounts - change password button)
        if (!isset($_SESSION['user_id']) || $_SESSION['role'] !== 'Admin') {
            echo json_encode(['error' => 'Unauthorized']);
            return;
        }

        if (!isset($data['id'], $data['password'])) {
            echo json_encode(['error' => 'Missing required parameters']);
            return;
        }

        $id = $conn->real_escape_string($data['id']);
        $password = password_hash($conn->real_escape_string($data['password']), PASSWORD_BCRYPT);
```



```
        $password = password_hash($conn->real_escape_string($data['password']), PASSWORD_BCRYPT);

        $query = "UPDATE user SET password = ? WHERE id = ?";
        $stmt = $conn->prepare($query);
        $stmt->bind_param("si", $password, $id);

        if ($stmt->execute()) {
            echo json_encode(['success' => true]);
        } else {
            echo json_encode(['error' => 'Password update failed: ' . $stmt->error]);
        }

        $stmt->close();
    } elseif ($data['action'] === 'update_file_status') {
        $file_id = $data['file_id'];
        $status = $data['status'];
        $remarks = $data['remarks'] ?? '';

        $query = "UPDATE files SET status = ?, remarks = ? WHERE file_name = ?";
        $stmt = $conn->prepare($query);
        $stmt->bind_param("sss", $status, $remarks, $file_id);

        if ($stmt->execute()) {
            echo json_encode(['success' => true]);
        } else {
            echo json_encode(['success' => false, 'error' => $stmt->error]);
        }

        $stmt->close();
    } elseif ($data['action'] === 'update_profile') {
        $userId = $_SESSION['user_id'];
        $fullName = $data['full_name'];
        $address = $data['address'];
        $contactInfo = $data['contact_info'];
        $password = $data['password'] ?? '';
```

```
        try {
            if (!empty($password)) {
                $passwordHash = password_hash($password, PASSWORD_BCRYPT);
                $sql = "UPDATE user SET full_name = ?, contact_info = ?, address = ?, password = ? WHERE id = ?";
                $stmt = $conn->prepare($sql);
                $stmt->bind_param("ssssi", $fullName, $contactInfo, $address, $passwordHash, $userId);
            } else {
                $sql = "UPDATE user SET full_name = ?, contact_info = ?, address = ? WHERE id = ?";
                $stmt = $conn->prepare($sql);
                $stmt->bind_param("sssi", $fullName, $contactInfo, $address, $userId);
            }

            if ($stmt->execute()) {
                if ($stmt->affected_rows > 0) {
                    $_SESSION['full_name'] = $fullName;
                    $_SESSION['address'] = $address;
                    $_SESSION['contact_info'] = $contactInfo;

                    echo json_encode(['success' => true, 'message' => 'Profile updated successfully']);
                } else {
                    echo json_encode(['success' => true, 'message' => 'No changes detected']);
                }
            } else {
                echo json_encode(['success' => false, 'error' => 'Database error: ' . $stmt->error]);
            }

            $stmt->close();
        } catch (Exception $e) {
            echo json_encode(['success' => false, 'error' => $e->getMessage()]);
        }
    } elseif ($data['action'] === 'update_department') {
        $department_id = $data['department_id'];
        $department_name = trim($data['department_name']);
        $folder_color = trim($data['folder_color']);

        $folder_color = str_replace('#', '', $folder_color);
```

```
        if (empty($department_id) || empty($department_name) || empty($folder_color)) {
            echo json_encode(['success' => false, 'message' => 'All fields are required']);
            exit;
        }

        $check_query = "SELECT id FROM department WHERE department_name = ? AND id != ?";
        $check_stmt = $conn->prepare($check_query);
        $check_stmt->bind_param("si", $department_name, $department_id);
        $check_stmt->execute();
        $result = $check_stmt->get_result();

        if ($result->num_rows > 0) {
            echo json_encode(['success' => false, 'message' => 'Department name already exists']);
            exit;
        }

        $query = "UPDATE department SET department_name = ?, folder_color = ? WHERE id = ?";
        $stmt = $conn->prepare($query);
        $stmt->bind_param("ssi", $department_name, $folder_color, $department_id);

        if ($stmt->execute()) {
            echo json_encode(['success' => true, 'message' => 'Department updated successfully']);
        } else {
            echo json_encode(['success' => false, 'message' => 'Error updating department']);
        }

        $stmt->close();
    } elseif ($data['action'] === 'reupload_file') {
        // Handle file reupload (change status to pending and clear remarks)
        $file_id = $data['file_id'];

        // Update the file status to 'pending' and clear remarks
        $query = "UPDATE files SET status = 'pending', remarks = NULL WHERE file_id = ?";
        $stmt = $conn->prepare($query);
        $stmt->bind_param("s", $file_id);
```

```
        if ($stmt->execute()) {
            echo json_encode(['success' => true]);
        } else {
            echo json_encode(['success' => false, 'error' => $stmt->error]);
        }

        $stmt->close();
    } else {
        echo json_encode(['success' => false, 'error' => 'Invalid action']);
    }
} else {
    echo json_encode(['success' => false, 'error' => 'Action not specified']);
}

$conn->close();
}
?>
```

upload_file.php

```php
<?php
session_start();
require_once 'conn.php';

error_reporting(E_ALL);
ini_set('display_errors', 0);
ini_set('log_errors', 1);
ini_set('error_log', 'upload_errors.log');

header('Content-Type: application/json');

try {
    if (!isset($_SESSION['user_id']) || !isset($_SESSION['department'])) {
        throw new Exception('User not authenticated or department not set');
    }

    if (!isset($_FILES['file'])) {
        throw new Exception('No file uploaded');
    }

    if (!isset($_POST['author']) || empty(trim($_POST['author']))) {
        throw new Exception('Author is required');
    }

    if (!isset($_POST['file_name']) || empty(trim($_POST['file_name']))) {
        throw new Exception('File Name is required');
    }

    $file = $_FILES['file'];
    $folder_id = $_POST['folder_id'];
    $author = trim($_POST['author']);
    $file_name = trim($_POST['file_name']);
    $user_id = $_SESSION['user_id'];
    $department = $_SESSION['department'];
    $status = $_POST['status'] ?? 'pending';
    $handle_duplicate = isset($_POST['handle_duplicate']) && $_POST['handle_duplicate'] === '1';
```

```php
    if ($file['error'] !== UPLOAD_ERR_OK) {
        throw new Exception('File upload error: ' . $file['error']);
    }

    $metadata_json = $_POST['metadata_json'] ?? null;
    $metadata = ($metadata_json) ? json_decode($metadata_json, true) : [];

    if (json_last_error() !== JSON_ERROR_NONE) {
        throw new Exception('Invalid metadata format');
    }

    $metadata_ids = array_keys($metadata);
    $valid_ids = [];

    if (!empty($metadata_ids)) {
        $placeholders = str_repeat('?,', count($metadata_ids) - 1) . '?';

        $validation_query = "SELECT metadata_id FROM metadata_title
                             WHERE metadata_id IN ($placeholders)
                             AND folder_id = ?";

        $validation_stmt = $conn->prepare($validation_query);

        $types = str_repeat('i', count($metadata_ids)) . 'i';
        $args = array_merge([$types], $metadata_ids, [$folder_id]);

        call_user_func_array([$validation_stmt, 'bind_param'], $args);
        $validation_stmt->execute();
        $result = $validation_stmt->get_result();

        while ($row = $result->fetch_assoc()) {
            $valid_ids[] = $row['metadata_id'];
        }
        $validation_stmt->close();
```

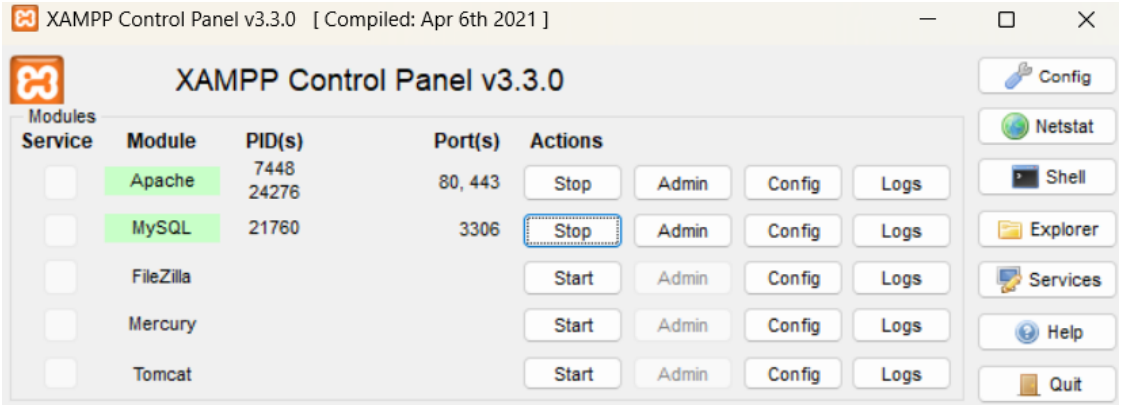
```php
        $invalid_ids = array_diff($metadata_ids, $valid_ids);
        if (!empty($invalid_ids)) {
            throw new Exception('Invalid metadata IDs: ' . implode(', ', $invalid_ids));
        }
    }

    $conn->begin_transaction();

    try {
        date_default_timezone_set('Asia/Manila');
        $upload_dir = '../uploads/';
        if (!file_exists($upload_dir)) {
            mkdir($upload_dir, 0777, true);
        }

        if ($handle_duplicate) {
            $check_query = "SELECT original_name FROM files WHERE folder_id = ? AND original_name LIKE ?";
            $check_stmt = $conn->prepare($check_query);
            $search_pattern = $file_name . '%';
            $check_stmt->bind_param('is', $folder_id, $search_pattern);
            $check_stmt->execute();
            $check_result = $check_stmt->get_result();

            $existing_names = [];
            while ($row = $check_result->fetch_assoc()) {
                $existing_names[] = $row['original_name'];
            }

            $new_name = $file_name;
            $counter = 2;

            while (in_array($new_name, $existing_names)) {
                $new_name = $file_name . ' ' . $counter;
                $counter++;
            }
```

```
            $file_name = $new_name;
        }

        $file_extension = pathinfo($file['name'], PATHINFO_EXTENSION);
        $unique_filename = uniqid() . '_' . time() . '.' . $file_extension;
        $upload_path = $upload_dir . $unique_filename;

        if (!move_uploaded_file($file['tmp_name'], $upload_path)) {
            throw new Exception('Failed to move uploaded file');
        }

        $query = "INSERT INTO files (file_name, original_name, upload_date, user_id, folder_id, author, department, status)
                  VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
        $stmt = $conn->prepare($query);
        $current_date = date('Y-m-d H:i:s');

        $stmt->bind_param(
            "sssissss",
            $unique_filename,
            $file_name,
            $current_date,
            $user_id,
            $folder_id,
            $author,
            $department,
            $status
        );

        if (!$stmt->execute()) {
            throw new Exception('Failed to insert file record: ' . $stmt->error);
        }

        $file_id = $stmt->insert_id;
        $stmt->close();
```

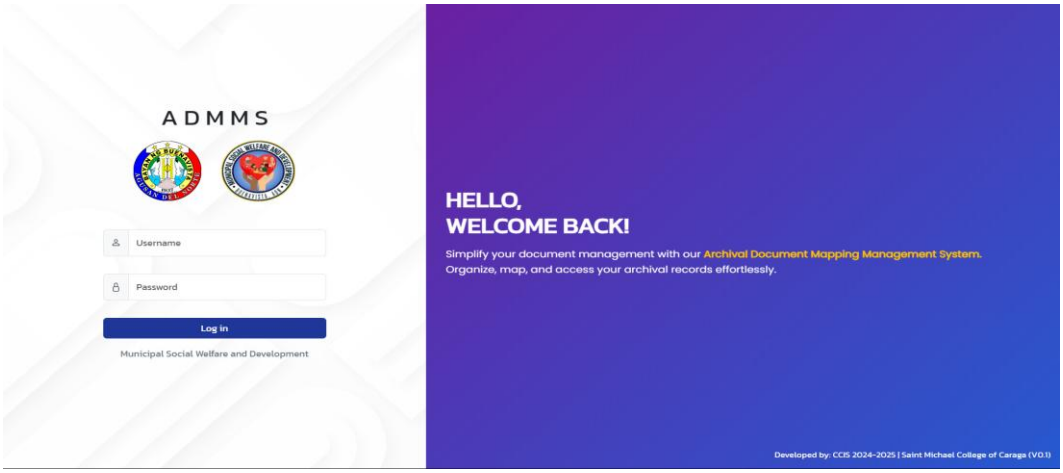
```
        if (!empty($valid_ids)) {
            foreach ($valid_ids as $meta_id) {
                if (isset($metadata[$meta_id]) && !empty(trim($metadata[$meta_id]))) {
                    $meta_query = "INSERT INTO metadata_description (file_id, title_id, meta_data_description, user_id)
                                   VALUES (?, ?, ?, ?)";
                    $meta_stmt = $conn->prepare($meta_query);
                    $meta_description = trim($metadata[$meta_id]);
                    $meta_stmt->bind_param("iiisi", $file_id, $meta_id, $meta_description, $user_id);

                    if (!$meta_stmt->execute()) {
                        throw new Exception('Failed to insert metadata: ' . $meta_stmt->error);
                    }
                    $meta_stmt->close();
                }
            }
        }

        $conn->commit();

        echo json_encode([
            'success' => true,
            'message' => 'File uploaded successfully',
            'file_name' => $unique_filename
        ]);

    } catch (Exception $e) {
        $conn->rollback();
        throw $e;
    }
```

```

} catch (Exception $e) {
    error_log('Upload error: ' . $e->getMessage());
    echo json_encode([
        'success' => false,
        'message' => $e->getMessage()
    ]);
}

$conn->close();
?>
```

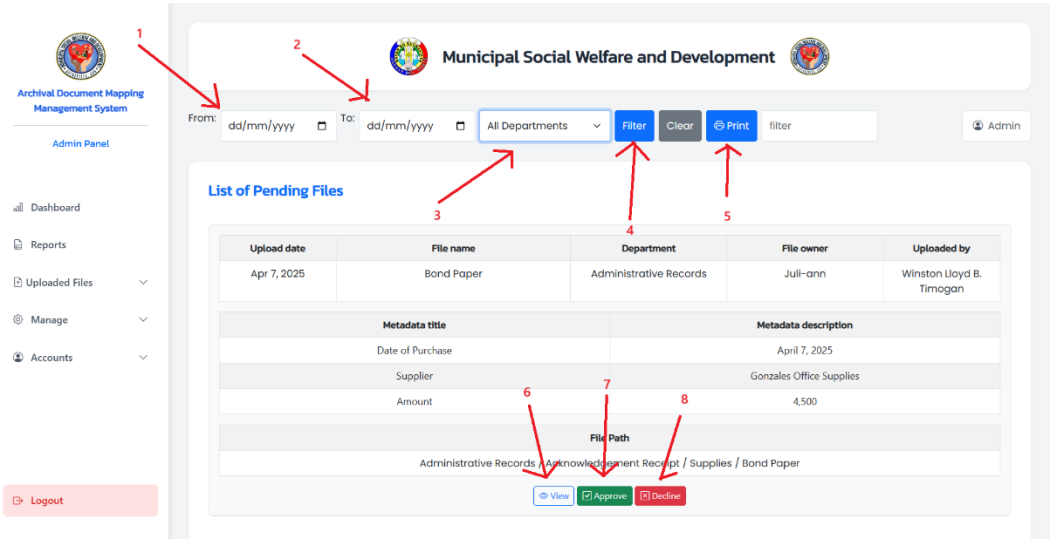
B. User's Manual



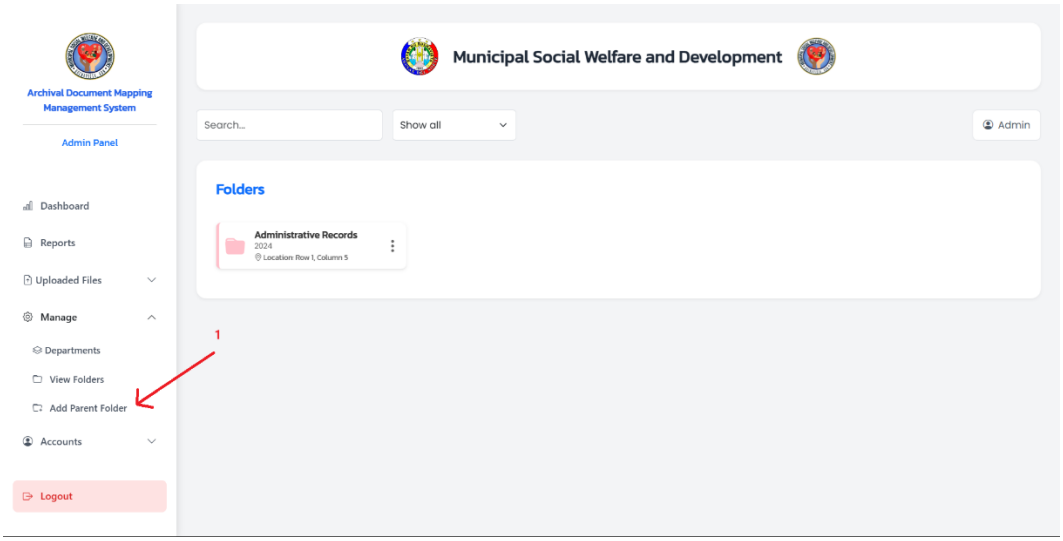
Make sure that XAMPP Apache and MySQL are up and running before clicking the desktop shortcut icon for ADMMS or opening your default browser and entering the following address: http://localhost/. The stop button indicates that MySQL and Apache are up and running. As a result, you can now use the system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



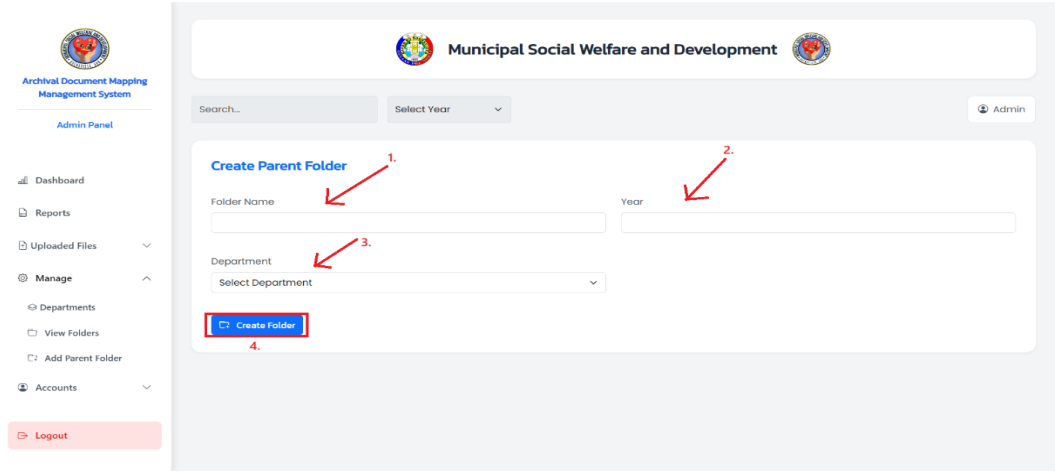
This is the log in page of both personnel and the admin. The personnel can only log in if they were given an account from the admin. It is a prevention of other user from accessing organization or admms.



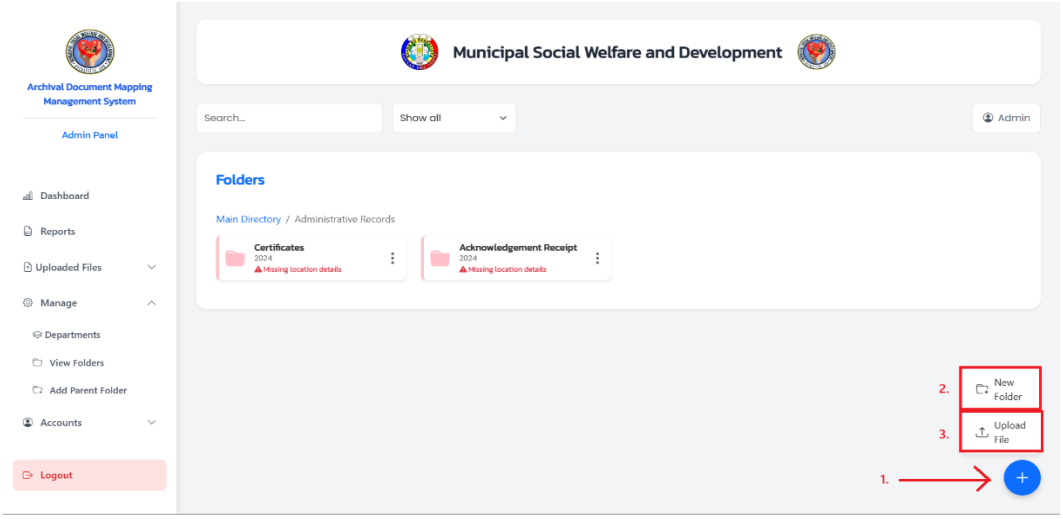
This is the admin UI for pendings you would encounter after logging in. The list of pending includes several key components: (1) a Date From filter to select the starting date, (2) a Date To filter to set the end date, (3) a Department Picker to choose the specific department for viewing related data, (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.



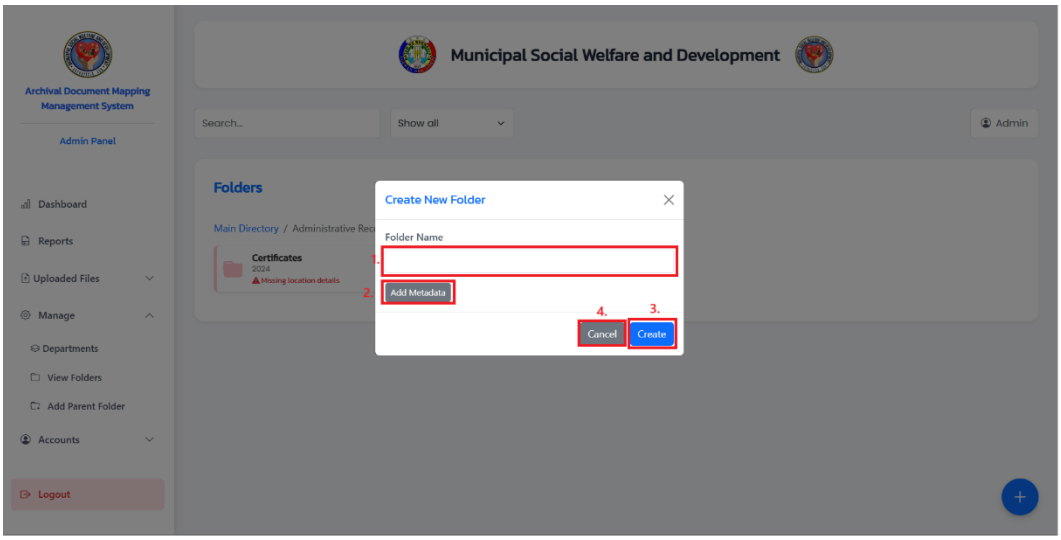
This is the UI for adding parent folder. (1) The Add Parent Folder button under Manage let create a main folder to organize related subfolders and documents.



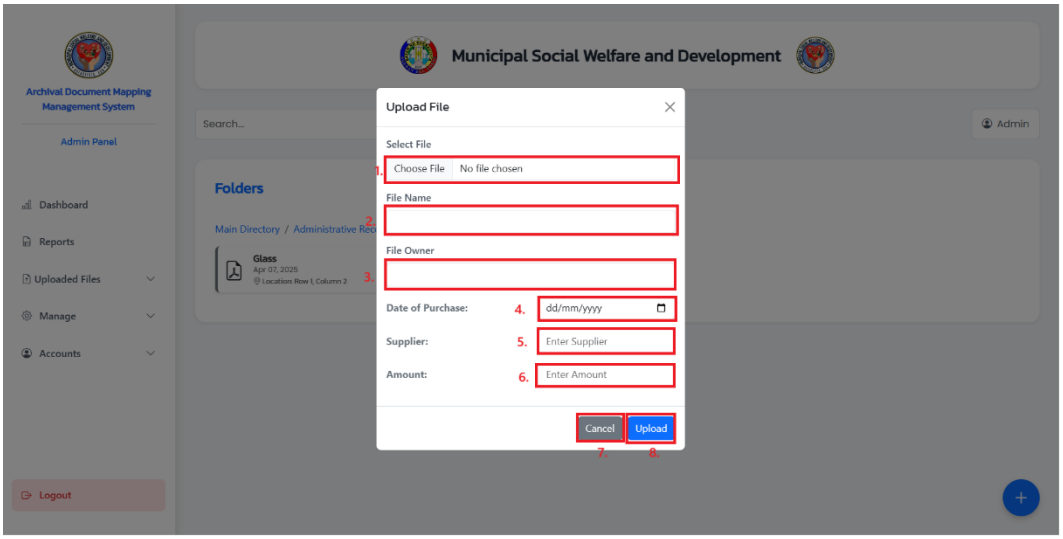
After redirecting to the Add Parent Folder section: The (1) Folder Name is where you enter the name of the folder, (2) Year is for specifying the folder's year, (3) Department is selected to assign the folder to a specific department, and (4) Create Folder button finalizes and adds the new parent folder.



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.

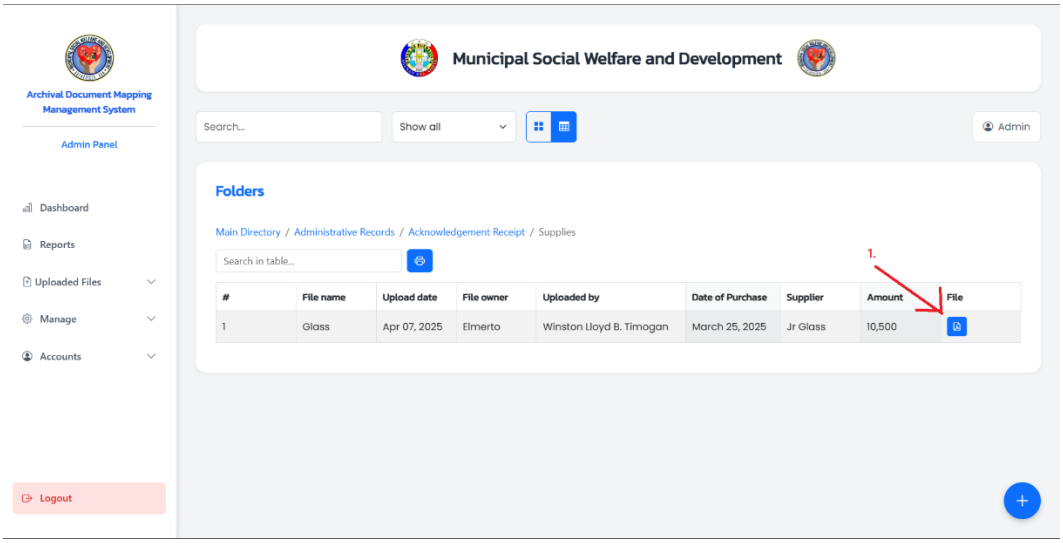


After clicking to the New Folder to create a subfolder, (1) input the Folder Name, (2) the admin can choice to add metadata, where they can specify the metadata name and input type, (3) click Create to finalize the subfolder, or (4) click Cancel to abort the process.



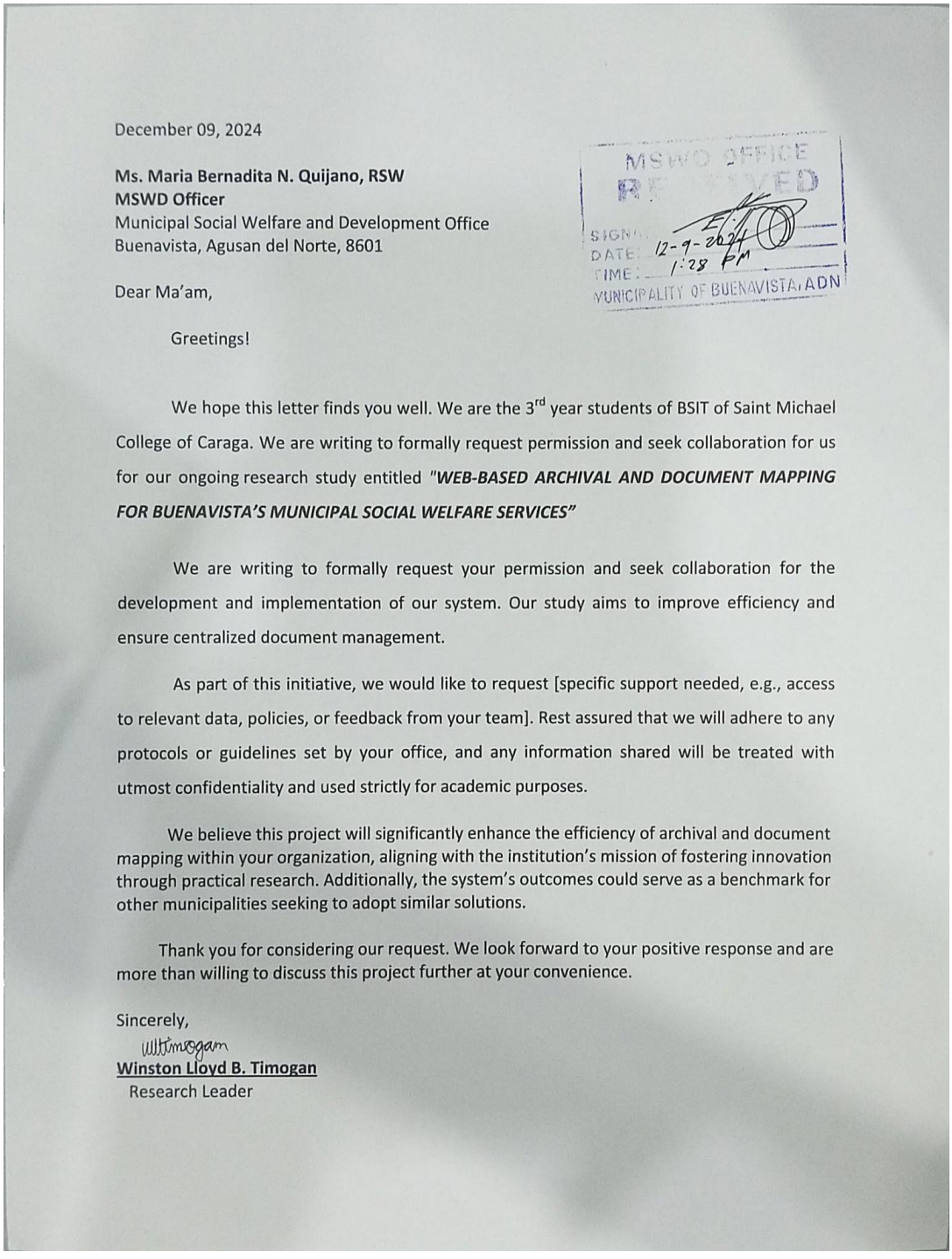
After tapping Upload File, a form would appear with the following fields: (1) Select File allows you to browse and choose a file from your device or computer, (2) File Name is where you can give the file a unique name for easy identification, (3) File Owner is where you enter the person for the file, (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 would submit the file along with the entered metadata, completing the process. This ensures all relevant information is captured for proper tracking and organization.



After uploading the file, (1) tapping the PDF file icon would 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

MSWD OFFICE
RECEIVED
SIGNATURE:
DATE: 12-9-2024
TIME: 1:28 PM
MUNICIPALITY OF BUENAVISTA, ADN

Dear Ma'am,

Greetings!

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 BUENAVISTA'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.

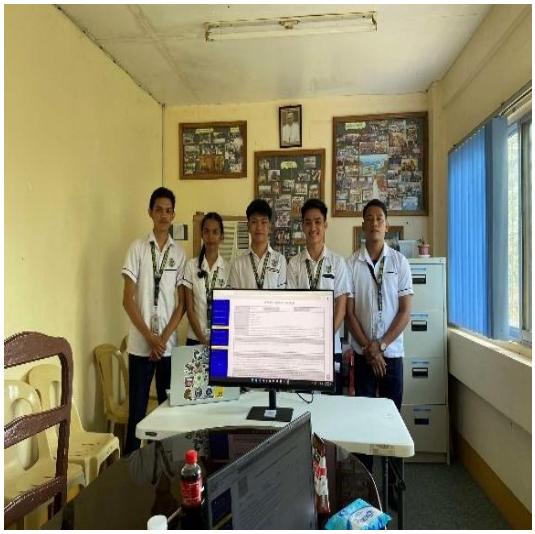
Thank you for considering our request. We look forward to your positive response and are more than willing to discuss this project further at your convenience.

Sincerely,

Winston Lloyd B. Timogan
Research Leader

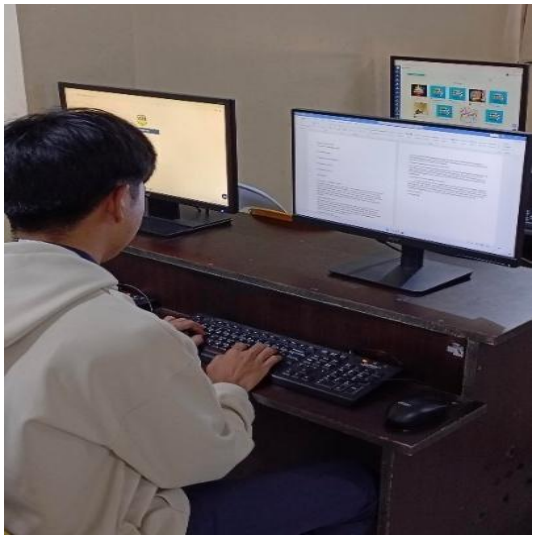
D. Documented Undertakings

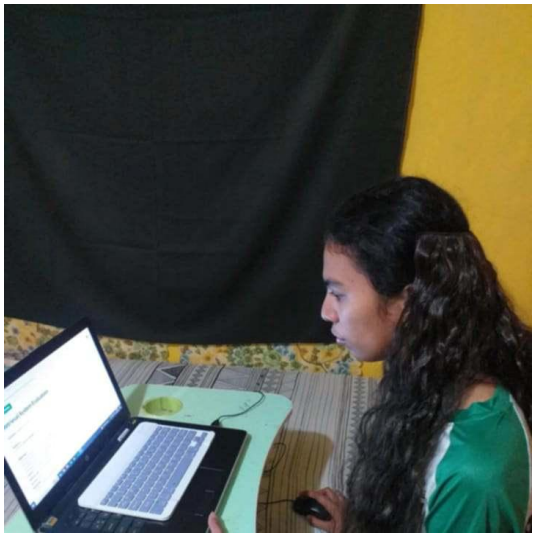
Title Hearing



Researchers successfully passed one title during the title hearing.

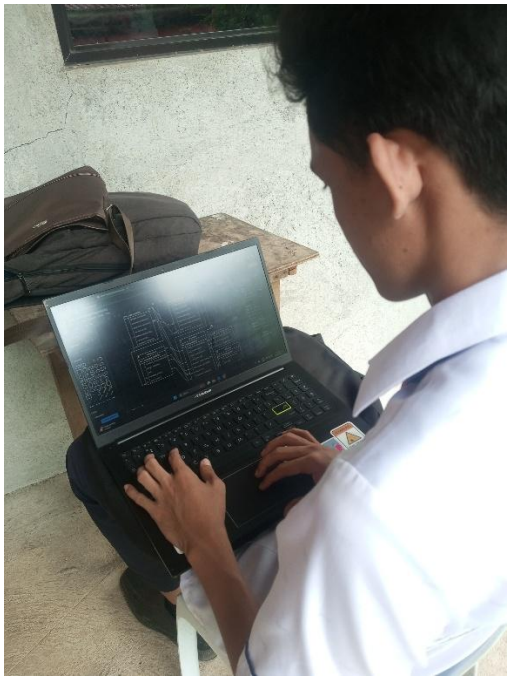
Chapter II Making





Researchers during the formulation of Chapter 2.

Diagrams Making

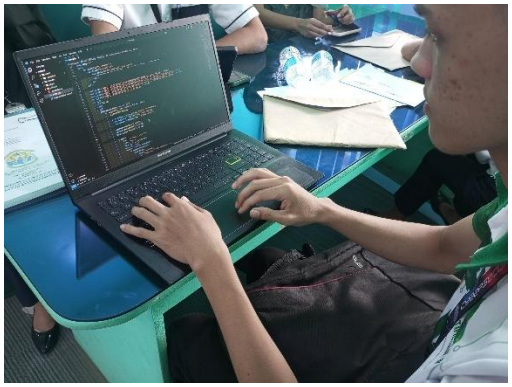
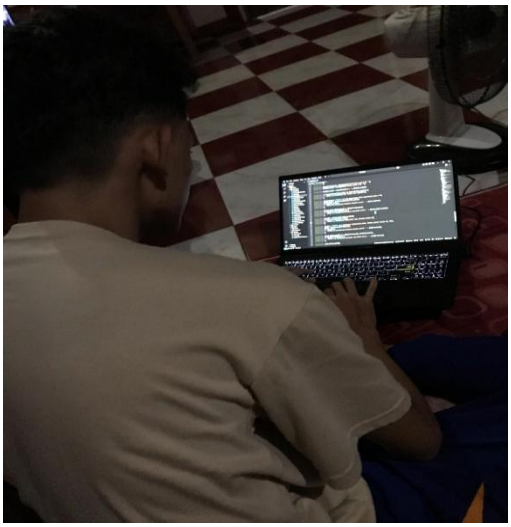


Researcher during the formulation of diagrams for Chapter 3.

Title Proposal

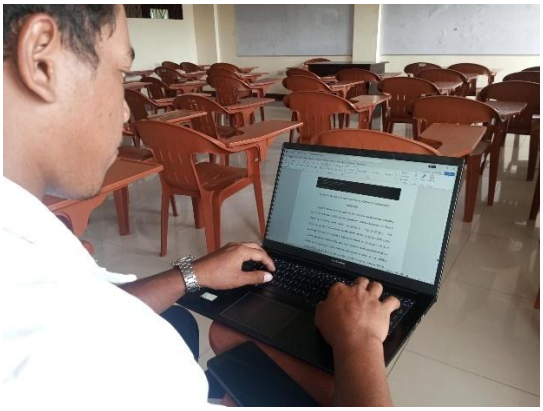
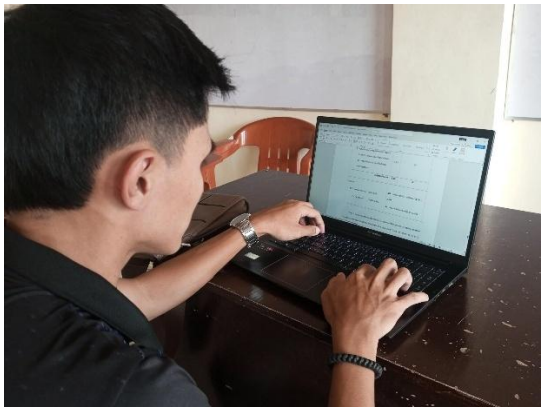
Researchers successfully defended Chapter 1 to 3 during the research proposal.

Developing a System

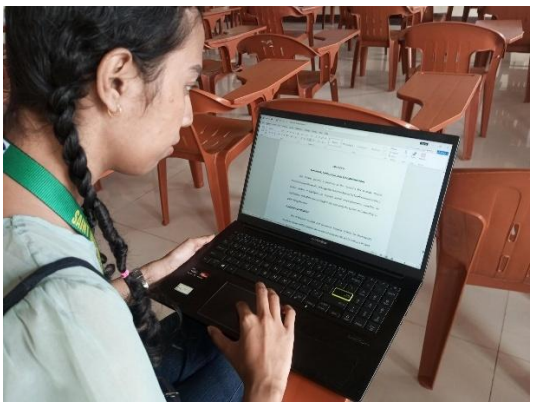

The Main Developer was responsible for designing, coding, and integrating the system's core features, managing database connections, creating technical diagrams, and ensuring the system met all functional requirements through testing and refinement.

Chapter 4 Making

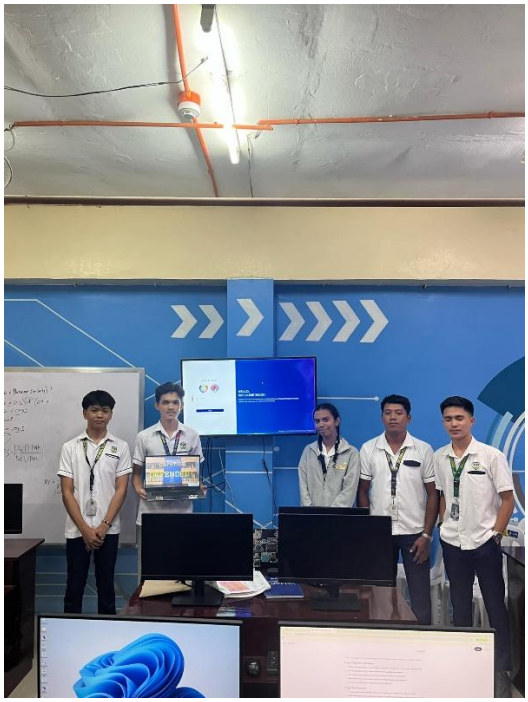
Researchers, during the formulation of Chapter 4, arranged the code, included the discussion, and tallied the results of the end users' evaluations. They focused on analyzing the code's functionality and provided insights based on the feedback collected from the users.

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.

Implementation and System Deployment

Researchers successfully deployed the system at the Municipal Social Welfare Services office in Buenavista, Agusan del Norte.

E. Certificate of Implementation/Extension

Republic of the Philippines
Province of Agusan del Norte
MUNICIPAL SOCIAL WELFARE AND DEVELOPMENT OFFICE
Municipality of Buenavista

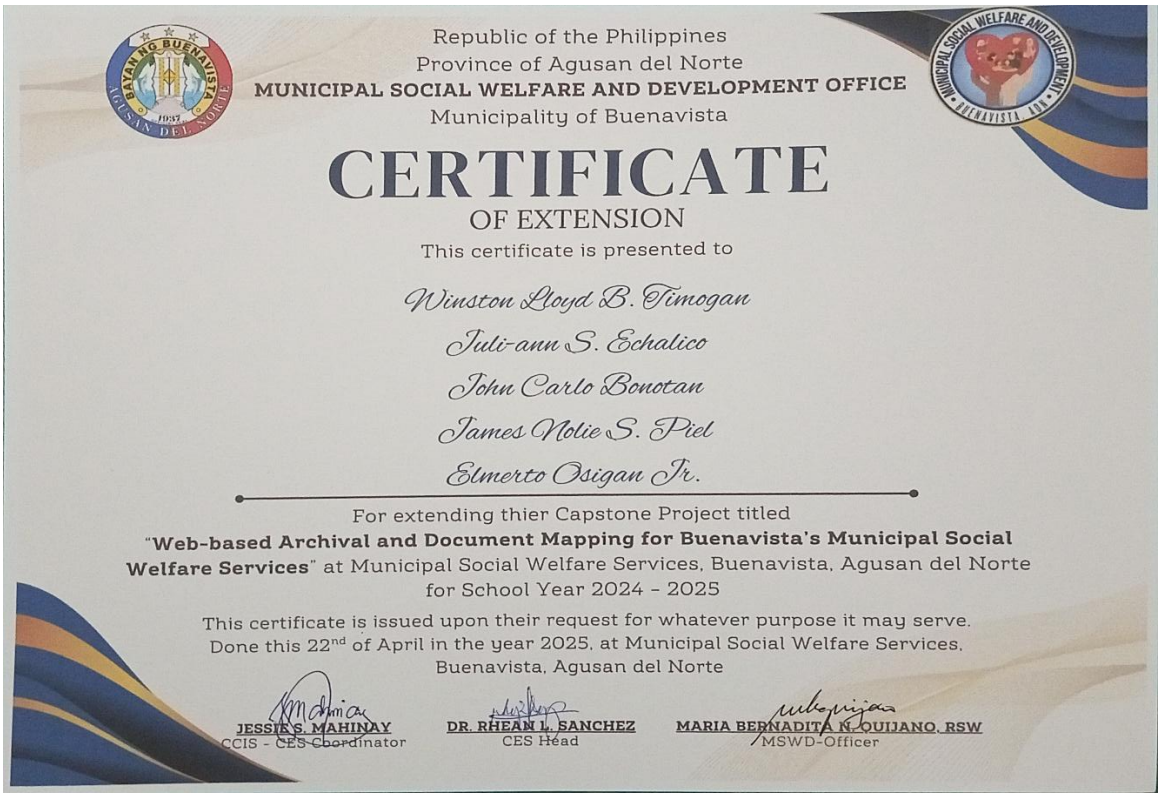
CERTIFICATE
OF IMPLEMENTATION

This certificate is presented to

Winston Lloyd B. Timogan
Juli-ann S. Echalico
John Carlo Bonotan
James Nolie S. Piel
Elmerto Osigan Jr.

Implemented thier Capstone Project titled,
"**Web-based Archival and Document Mapping for Buenavista's Municipal Social Welfare Services**"
Given this 22nd of April in the year 2025, at Municipal Social Welfare Services , Buenavista Agusan del Norte

MARIA BERNADITA N. QUIJANO, RSW
MSWD-Officer



Republic of the Philippines
Province of Agusan del Norte
MUNICIPAL SOCIAL WELFARE AND DEVELOPMENT OFFICE
Municipality of Buenavista

CERTIFICATE
OF EXTENSION

This certificate is presented to

Winston Lloyd B. Timogan
Juli-ann S. Echalico
John Carlo Bonotan
James Nolie S. Piel
Elmerto Osigan Jr.

For extending thier Capstone Project titled
"**Web-based Archival and Document Mapping for Buenavista's Municipal Social Welfare Services**" at Municipal Social Welfare Services, Buenavista, Agusan del Norte for School Year 2024 - 2025

This certificate is issued upon their request for whatever purpose it may serve.
Done this 22nd of April in the year 2025, at Municipal Social Welfare Services, Buenavista, Agusan del Norte

JESSIE S. MAHINAY
CCIS - CES Coordinator

DR. RHEAN L. SANCHEZ
CES Head

MARIA BERNADITA N. QUIJANO, RSW
MSWD-Officer

F. Map of the Research Locale

G. Curriculum Vitae

Name : Winston Lloyd B. Timogan

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

Email : timogan001@gmail.com

Personal Information

Sex : Male

Date of Birth : November 7, 2003

Height : 170cm

Weight : 49kg

Civil Status : Single

Educational Attainment

Elementary : Culit Elementary School

Secondary : Northwestern Agusan Colleges

Tertiary : Saint Michael College of Caraga

Membership/Affiliations

ORGANIZATION	POSITIONS HELDS
Collegiate Computing Society (2024 - 2025)	Vice - President
Michaelinian Security Guild (2024 - 2025)	3rd Year Representative

G. Curriculum Vitae

Name : Juli-Ann S. Echalico

Address : P-1 Guinabsan, Buenavista, ADN

Email : akrawatanab20@gmail.com

Personal Information

Sex : Female

Date of Birth : February 24, 2003

Height : 162cm

Weight : 42

Civil Status : Single

Educational Attainment

Elementary : Tanutao Elementary School

Secondary : Saint James High School

Tertiary : Saint Michael College of Caraga

Membership/Affiliations

ORGANIZATION	POSITIONS HELDS
N/A	

G. Curriculum Vitae

Name : James Nolie S. Piel

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

Email : jamesnolie18@gmail.com

Personal Details

Sex : Male

Date of Birth : April 12, 2002

Height : 165.1cm

Weight : 54kg

Civil Status : Single

Educational Attainment

Elementary : Buenavista Central Elementary School

Secondary : Saint James High School

Tertiary : Saint Michael College of Caraga

Membership/Affiliations

ORGANIZATION	POSITIONS HELDS
Michaelinian Security Guild (2024 - 2025)	Beau

G. Curriculum Vitae

Name : John Carlo M. Bonotan

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

Email : bonotanjohncarlo@gmail.com

Personal Details

Sex : Male

Date of Birth : March 1, 2003

Height : 167cm

Weight : 42kg

Civil Status : Single

Educational Attainment

Elementary : Buenavista Elementary School

Secondary : Saint James High School

Tertiary : Saint Michael College of Caraga

Membership/Affiliations

ORGANIZATION	POSITIONS HELDS
N/A	

G. Curriculum Vitae

Name : Elmerto G. Osigan Jr.

Address : Punta D-6 Nasipit, Agusan Del Norte

Email : elmertoosigan101@gmail.com

Personal Information

Sex : Male

Date of Birth : April 14, 1999

Height : 173cm

Weight : 68kg

Civil Status : Single

Educational Attainment

Elementary : Punta Elementary School

Secondary : Northwestern Agusan Colleges

Tertiary : Saint Michael College of Caraga

Membership/Affiliations

ORGANIZATION	POSITIONS HELDS
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: April 28, 2025
Location: Saint Michael College of Caraga

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