STORAGE AND BACKUP SYSTEM FOR KITENGELA CREATIVE STUDIO

SYSTEM DOCUMENTATION

SUBMITTED BY,

EVANS MUNENE

DIT-01-0285/2024

A SYSTEM DOCUMENTATION SUBMITTED IN PARTIAL FULFILMENT FOR THE AWARD OF DIPLOMA IN INFORMATION TECHNOLOGY BY ZETECH UNIVERSITY

OCTOBER 2025

CHAPTER ONE: PROJECT PLANNING AND ANALYSIS

(WORKPLAN)

1.1 Statement of Problem

The Kitengela Creative Studio has been experiencing frequent file loss among students and trainers, especially when working on creative projects such as animations, graphics, and video editing. This often results in wasted time, disrupted workflows, and in some cases, total loss of completed work. The situation is made worse by heavy reliance on unreliable storage devices like flash drives and memory cards, which are prone to damage and corruption (Backblaze, 2021). Most users have limited awareness of backup practices, and poor internet access further discourages the use of cloud-based solutions (UNESCO, 2023). As a result, files are rarely backed up, exposing users to risks such as accidental deletion, software crashes, or hardware failure (Norton, 2022). This persistent problem not only affects productivity but also compromises the quality and consistency of creative outputs. There is a growing need to develop affordable and practical strategies to improve digital storage and backup at the studio.

1.2 Study Justification

This study is important because it addresses the recurring issue of file loss at Kitengela Creative Studio, which negatively affects the productivity and learning outcomes of students and trainers. It aims to diagnose the root causes of the problem and explore how it impacts creative work in a training environment. By examining existing digital storage and backup solutions such as cloud services, external drives, and automated software the study will assess how these systems function, what technical skills or tools they require, and how long they may take to implement. This investigation will help simulate real-life solutions that respond directly to the identified challenges. Ultimately, the findings will offer a well-informed foundation for selecting a suitable solution or combination of solutions that can be developed

1.3System Objectives

1.3.1 General Objective

To design and implement a secure system that stores user data and performs automated backups.

1.3.2 Specific Objectives

i. To design a secure user registration and authentication module.
ii. To develop a user dashboard for uploading and accessing stored data.
iii. To implement an automated data backup process linked to user activity.
iv. To develop an admin module for managing users, files, and backup logs.

1.3.4Functional Requirements

User	User Activities	Features
Admin	Approve user registrations	Admin dashboard with signup approval panel
Admin	Monitor backup activities	Real-time backup logs display

Admin	Manage users and files	Add, Edit or delete user data and files
User	Register and login securely	Authentication with email and password
User	Upload files for storage and backup	File upload, storage and backup module with progress trucker
User	Access previously backed up files	File retrieval and download interface
User	View backup history	Backup history showing timestamp and file details
System Automated	Perform regular automatic backups	Schedule backup service
System Automated	Notify users and admin of successful backups	Email /notification alerts after backups

Table 1.4 Functional Requirements Table

1.3 Breakdown of Tools & Resources to Be Used

Category	Tools / Resources	Purpose / Description
Hardware	Computers / Laptops	Used for system development, testing, and documentation
Storage Devices	External Hard Drive / Cloud Storage (Google Drive, OneDrive)	Backup and file storage during testing
Software	Visual Studio Code / Node.js	Development environment for backend and frontend
Database	MySQL / SQLite	To store user credentials, file data, and backup logs
Web Technologies	HTML, CSS, JavaScript	Frontend design and user interface
Server Tools	Express.js / JSON Server	For running and testing APIs
Version Control	Git & GitHub	Code management and version control
Testing Tools	Postman	API endpoint testing
Documentation Tools	Microsoft Word, Draw.io, Lucidchart	For reports, diagrams, and documentation
Human Resources	Project Developer (Evans Munene)	System design, coding, and testing
Power & Internet	Electricity and Internet Access	Essential for system deployment and communication

1.6 Project Schedule Breakdown

	PROJECT MILESTONES

WEEKS	Project Planning & Analysis (System Documentation: Cover page & Chapter One)	Project Design & Modeling (System Documentation Chapter Two)	Project Development & Testing (System Documentation Chapter Three)	Project Deployment (System Documentation Chapter Three)	Final Touches of System Documentation (Preliminary Pages, Chapter Four & References)	Project Presentation
19th -26th September 2025	🟦					
27th -10th October 2025		🟦				
11th -31st October 2025			🟦			
1st - 14th November 2025				🟦		
15th -24th November 2025					🟦	
25th -28th November						🟦

Table 1.6 Project Schedule Breakdown

CHAPTER TWO: DESIGN AND MODELING

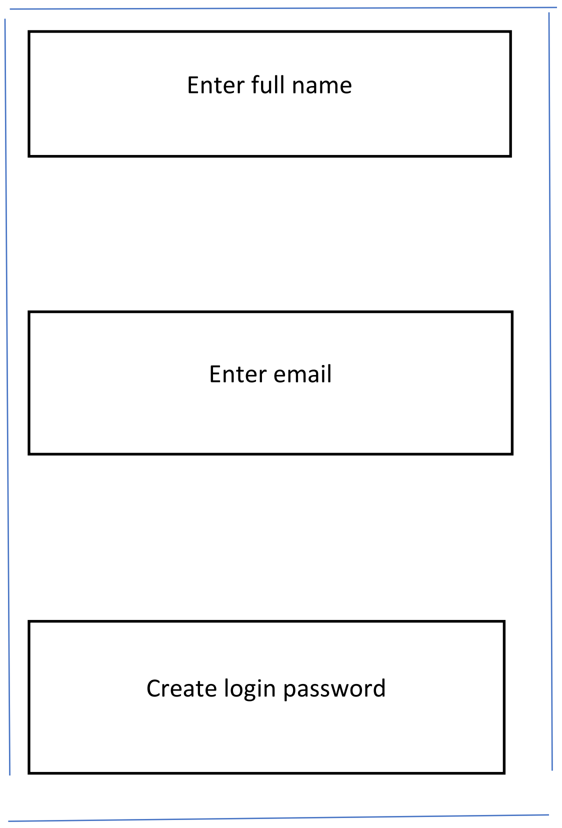
2.1 Introduction to Modelling

In this chapter, I am presenting the various design and modeling diagrams that i created during the system planning and design phase. These models helped visualize how the Backup System for Creative Studio would function before any development began. Modeling allowed me to understand the relationships between different components, plan user interactions, and structure system logic clearly. By sketching and diagramming before development, I reduced design errors, improved efficiency, and ensured the system architecture was well thought out and consistent with user needs.

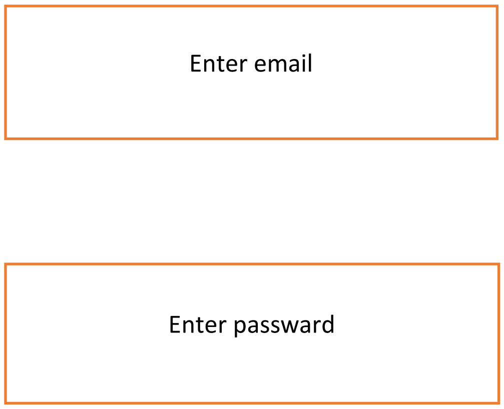
2.2 User Interface Models

This section presents the sketches and design of the system's user interfaces. Each interface was conceptualized to show how users would interact with the system. The UI models helped visualize layout, input elements, and navigation flow before implementing them in code.

2.2.1 sign up form



2.2.2 **Login page**



2.2.3 Dashboard Page Design

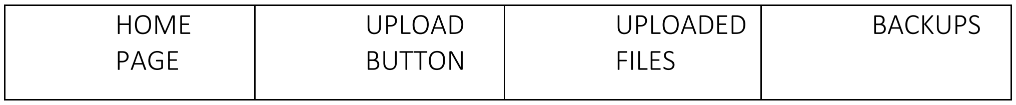
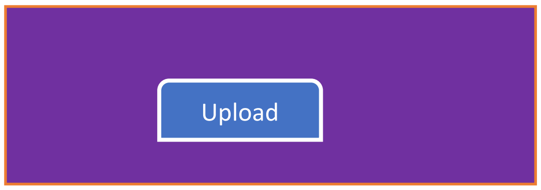


Figure Dashboard Interface Design, Hand-Sketched

2.2.4 File Upload Page Design



2.3 Logic Models

This section presents the logical and architectural diagrams developed to illustrate how the Backup System operates. These diagrams represent data flow, process logic, and interactions between users, system components, and data stores.

2.3.1 System Flowchart

The system flowchart illustrates the sequence of operations within the Backup System. It shows how users log in, upload files, and how the system validates, stores, and logs each operation.

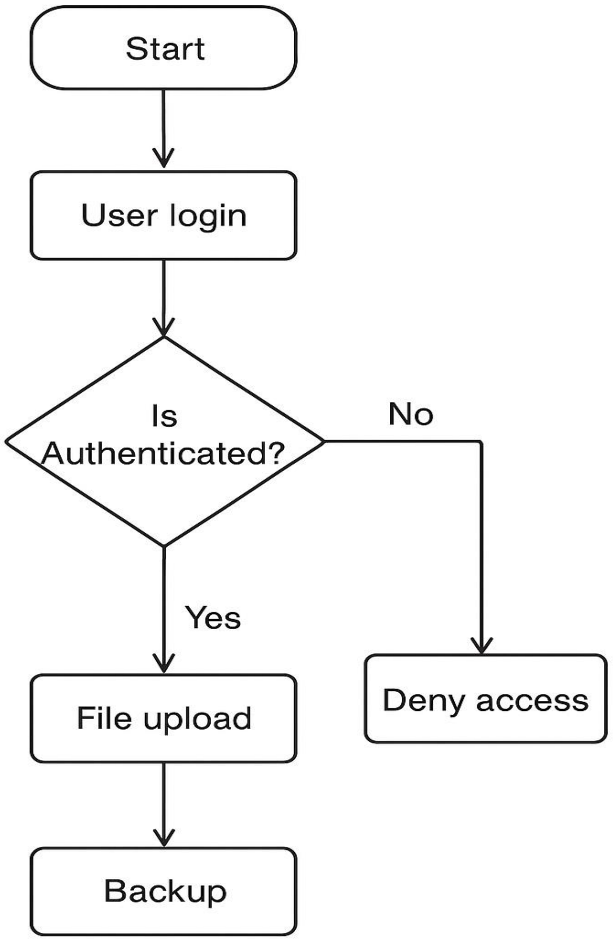


Figure 2.3.1.1System Flowchart for the Backup System

2.3.2 DFD Level 0 (Context Diagram)

This diagram shows the Backup System as a single process interacting with external entities such as users, administrators, and cloud storage providers. It defines the system boundary and the overall flow of data.

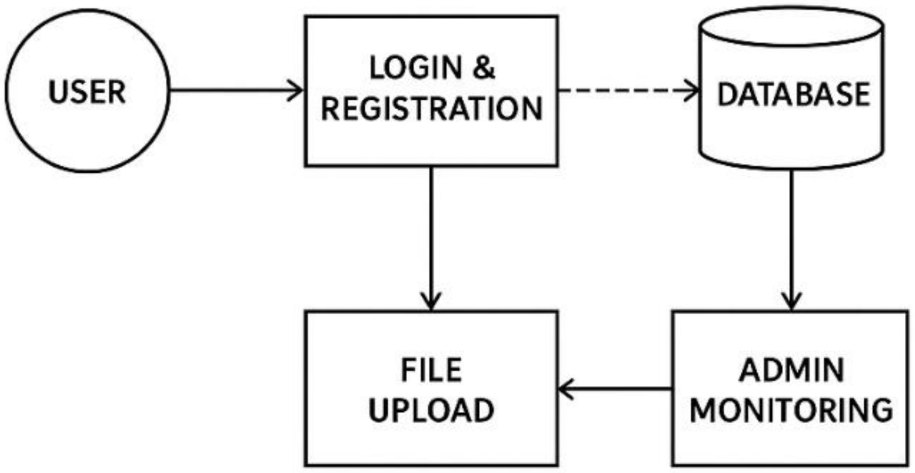


Figure 2.3.2.1 DFD Level 0 (Context Diagram)

2.3.3 DFD Level 1 (Detailed Data Flow Diagram)

The Level 1 DFD breaks down the main system into sub processes, including user authentication, file upload and validation, backup scheduling, and monitoring. It also shows how data moves between these processes and data stores.

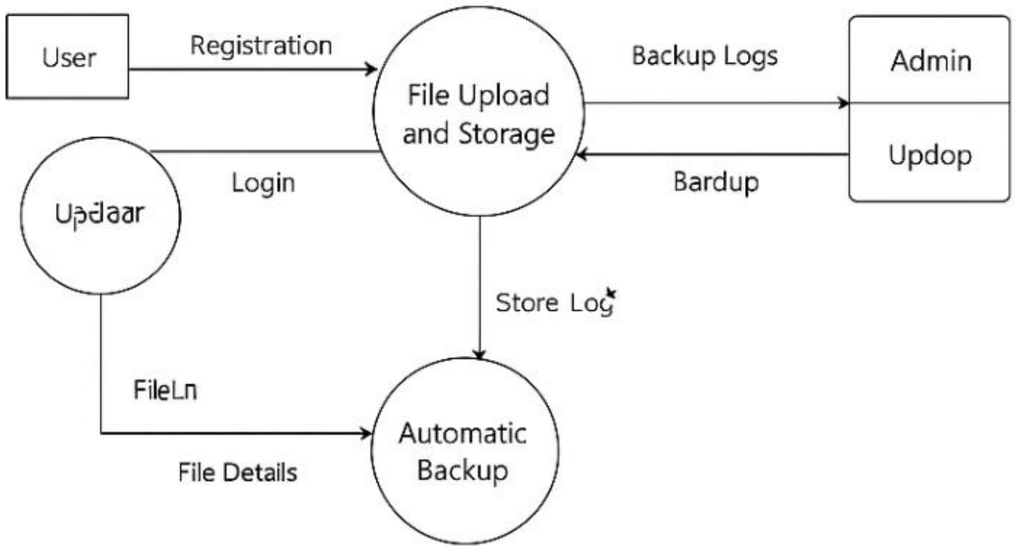


Figure 2.3.3.1 DFD Level 1 (Detailed Data Flow Diagram)

2.3.4 Entity Relationship Diagram (ERD)

The ERD defines the database structure of the Backup System. It highlights entities such as User, File, and Backup Logs, along with their relationships. The model ensures proper data organization, integrity, and efficient backup tracking.

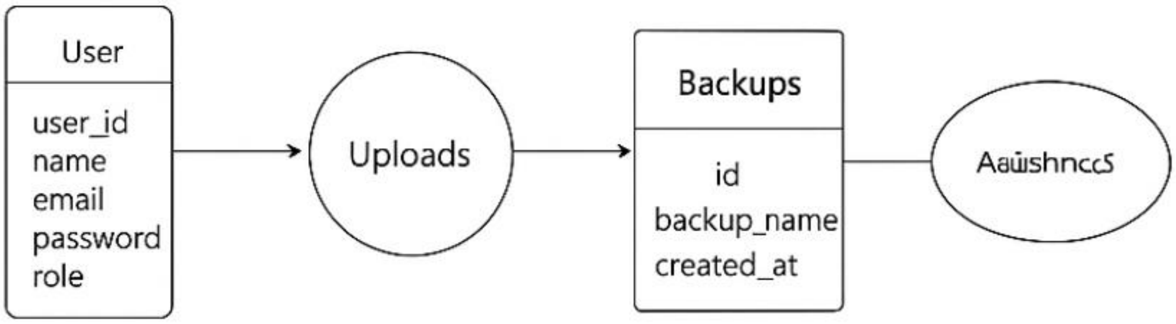
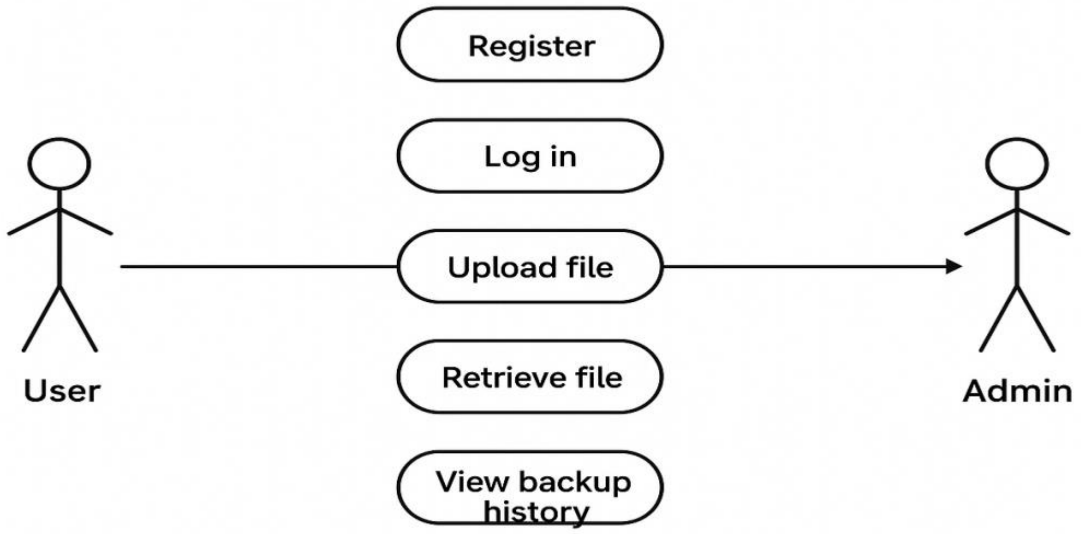


Figure 2.3.4.1 Entity Relationship Diagram (ERD)

2.3.5 Use Case Diagram

This diagram below illustrates the interactions between system actors and use cases. The main actors are the User, Administrator, and Backup Scheduler. It defines the operations each actor performs, such as uploading files, reviewing logs, and performing automated backups.


Register
Log in
Upload file
Retrieve file
View backup history
User
Admin

CHAPTER THREE: SYSTEM IMPLEMENTATION
(DEVELOPMENT, TESTING AND DEPLOYMENT)

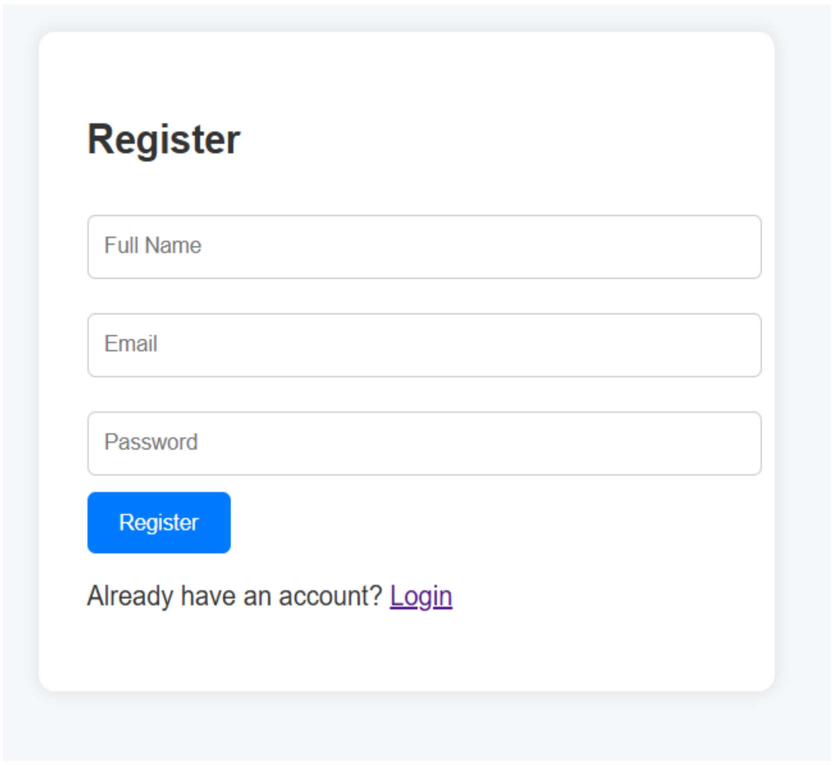
3.1 Introduction

This chapter describes the complete implementation process of the Backup Management System. It presents the development journey from setting up the development environment, coding both frontend and backend modules, configuring the database, to finally deploying and testing the system. The goal is to demonstrate how the proposed design was transformed into a fully functional application. The implementation involved using tools such as Visual Studio Code, Node.js, MySQL, and phpMyAdmin, alongside frontend technologies like HTML, CSS, and JavaScript.

3.2 User Interface Development

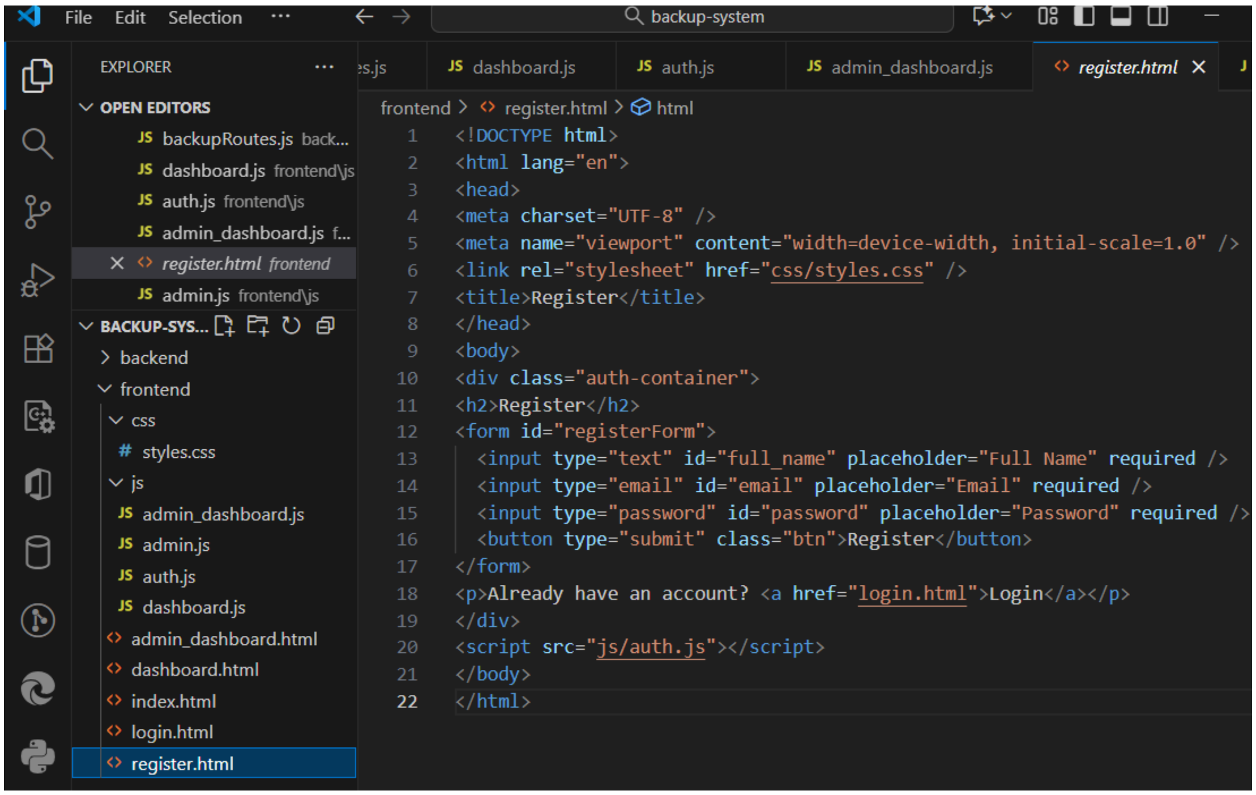
This chapter describes the complete implementation process of the Backup Management System. It presents the development journey from setting up the development environment, coding both frontend and backend modules, configuring the database, to finally deploying and testing the system. The goal is to demonstrate how the proposed design was transformed into a fully functional application. The implementation involved using tools such as Visual Studio Code, Node.js, MySQL, and phpMyAdmin, alongside frontend technologies like HTML, CSS, and JavaScript.

Figure 3.2.2.1: *User Registration Form Interface*



(Screenshot of registration page goes here)

Figure 3.2.2.2: *Registration Page Code Snippet*



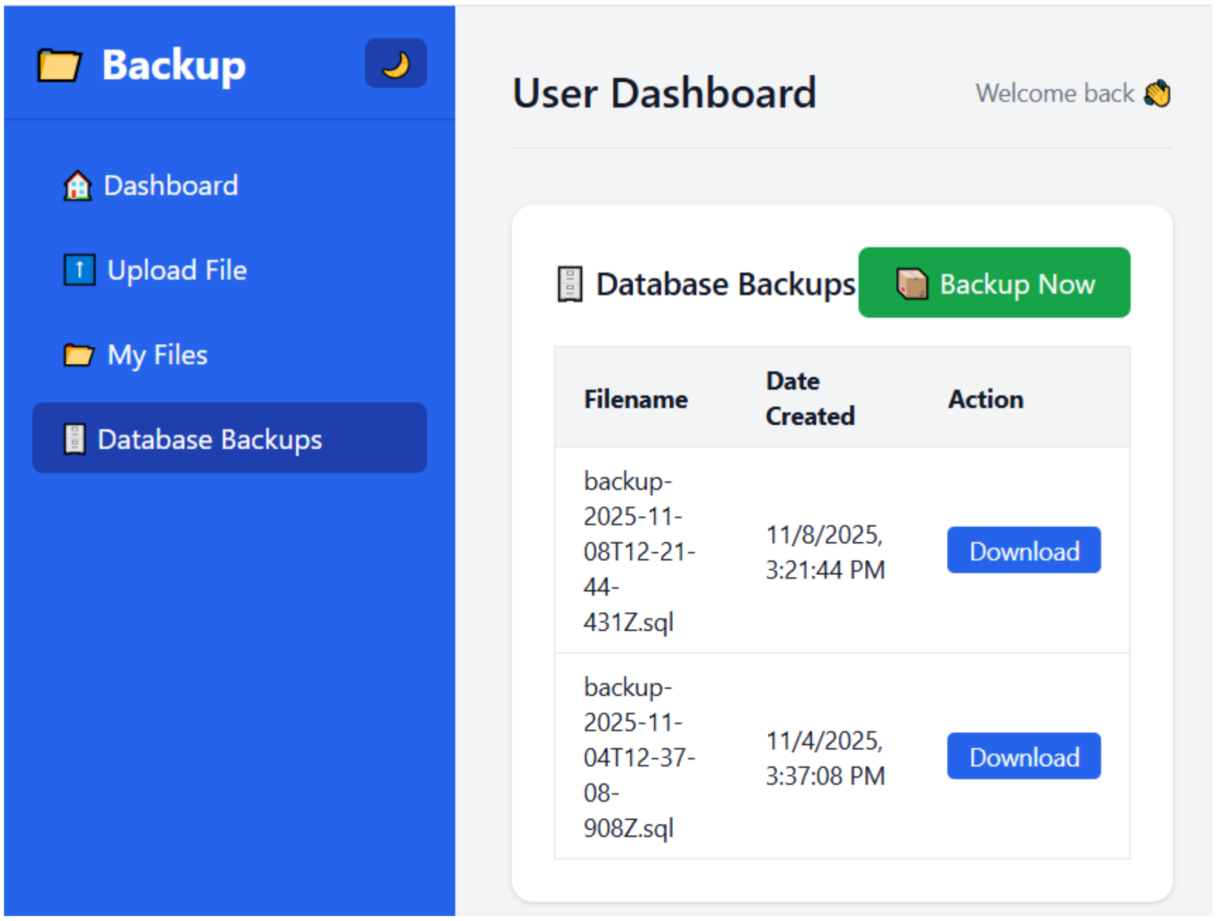
```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="css/styles.css" />
<title>Register</title>
</head>
<body>
<div class="auth-container">
<h2>Register</h2>
<form id="registerForm">
  <input type="text" id="full_name" placeholder="Full Name" required />
  <input type="email" id="email" placeholder="Email" required />
  <input type="password" id="password" placeholder="Password" required />
  <button type="submit" class="btn">Register</button>
</form>
<p>Already have an account? <a href="login.html">Login</a></p>
</div>
<script src="js/auth.js"></script>
</body>
</html>
```

(Screenshot of HTML/JS code goes here)

3.2.3 Dashboard Page Development

The dashboard provides access to system functionalities such as uploading files, viewing uploaded files, and creating or downloading backups. It dynamically displays information retrieved from the backend through RESTful API endpoints such as /api/files/list and /api/backup/database.

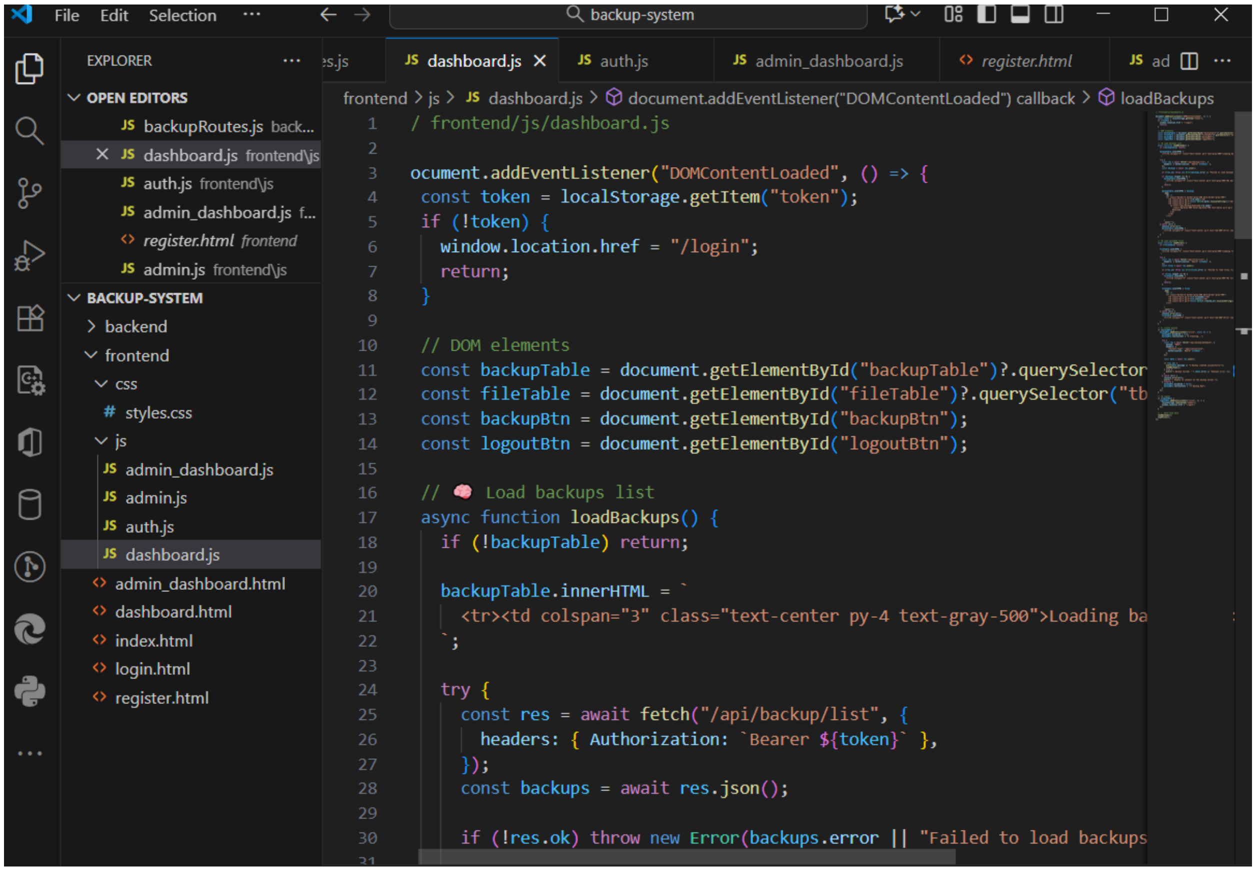
Figure 3.2.3.1: *User Dashboard Interface*



Filename	Date Created	Action
backup-2025-11-08T12-21-44-431Z.sql	11/8/2025, 3:21:44 PM	Download
backup-2025-11-04T12-37-08-908Z.sql	11/4/2025, 3:37:08 PM	Download

(Screenshot of dashboard page goes here)

Figure 3.2.3.2: *Dashboard Script Code (dashboard.js)*



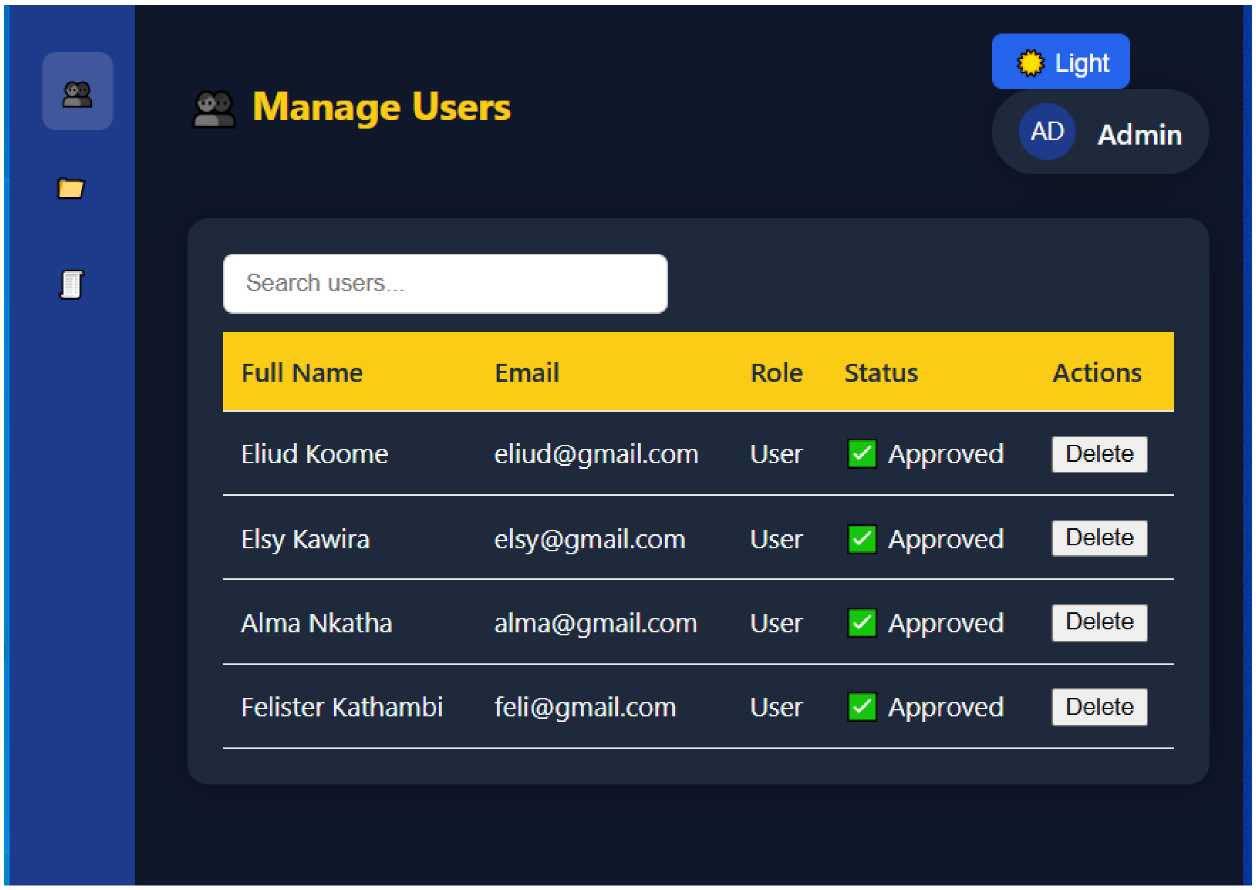
```
/ frontend/js/dashboard.js

ocument.addEventListener("DOMContentLoaded", () => {
 const token = localStorage.getItem("token");
 if (!token) {
   window.location.href = "/login";
   return;
 }

 // DOM elements
 const backupTable = document.getElementById("backupTable")?.querySelector
 const fileTable = document.getElementById("fileTable")?.querySelector("tb
 const backupBtn = document.getElementById("backupBtn");
 const logoutBtn = document.getElementById("logoutBtn");

 // 🧠 Load backups list
 async function loadBackups() {
   if (!backupTable) return;

   backupTable.innerHTML = `
     <tr><td colspan="3" class="text-center py-4 text-gray-500">Loading ba
   `;

   try {
     const res = await fetch("/api/backup/list", {
       headers: { Authorization: `Bearer ${token}` },
     });
     const backups = await res.json();

     if (!res.ok) throw new Error(backups.error || "Failed to load backups
```

(Screenshot of code snippet goes here)

3.2.4 Admin Dashboard Page

The admin dashboard allows administrators to view all user activities, approve user accounts, and monitor backup operations. It provides real-time system logs and allows access to database backup files.

Figure 3.2.4.1: *Admin Dashboard Interface*



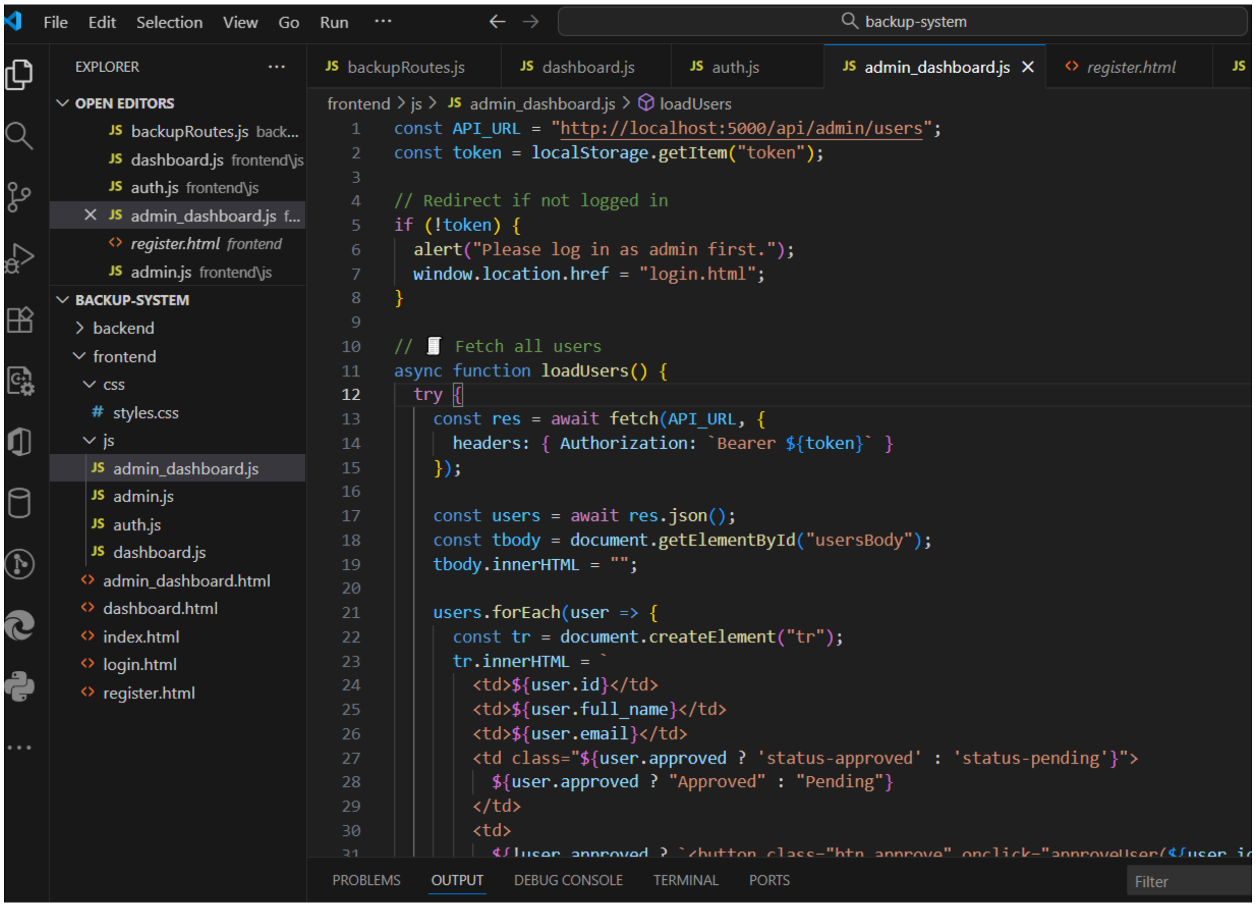
Manage Users

Search users...

Full Name	Email	Role	Status	Actions
Eliud Koome	eliud@gmail.com	User	✅ Approved	Delete
Elsy Kawira	elsy@gmail.com	User	✅ Approved	Delete
Alma Nkatha	alma@gmail.com	User	✅ Approved	Delete
Felister Kathambi	feli@gmail.com	User	✅ Approved	Delete

(Screenshot of admin_dashboard.html)

Figure 3.2.4.2: *Admin Dashboard Logic Code (admin.js)*



```javascript
const API_URL = "http://localhost:5000/api/admin/users";
const token = localStorage.getItem("token");

// Redirect if not logged in
if (!token) {
  alert("Please log in as admin first.");
  window.location.href = "login.html";
}

// 📋 Fetch all users
async function loadUsers() {
  try {
    const res = await fetch(API_URL, {
      headers: { Authorization: `Bearer ${token}` }
    });

    const users = await res.json();
    const tbody = document.getElementById("usersBody");
    tbody.innerHTML = "";

    users.forEach(user => {
      const tr = document.createElement("tr");
      tr.innerHTML = `
        <td>${user.id}</td>
        <td>${user.full_name}</td>
        <td>${user.email}</td>
        <td class="${user.approved ? 'status-approved' : 'status-pending'}">
          ${user.approved ? "Approved" : "Pending"}
        </td>
        <td>
          ${!user.approved ? `<button class="btn approve" onclick="approveUser(${user.i
```

(Screenshot of code snippet goes here)

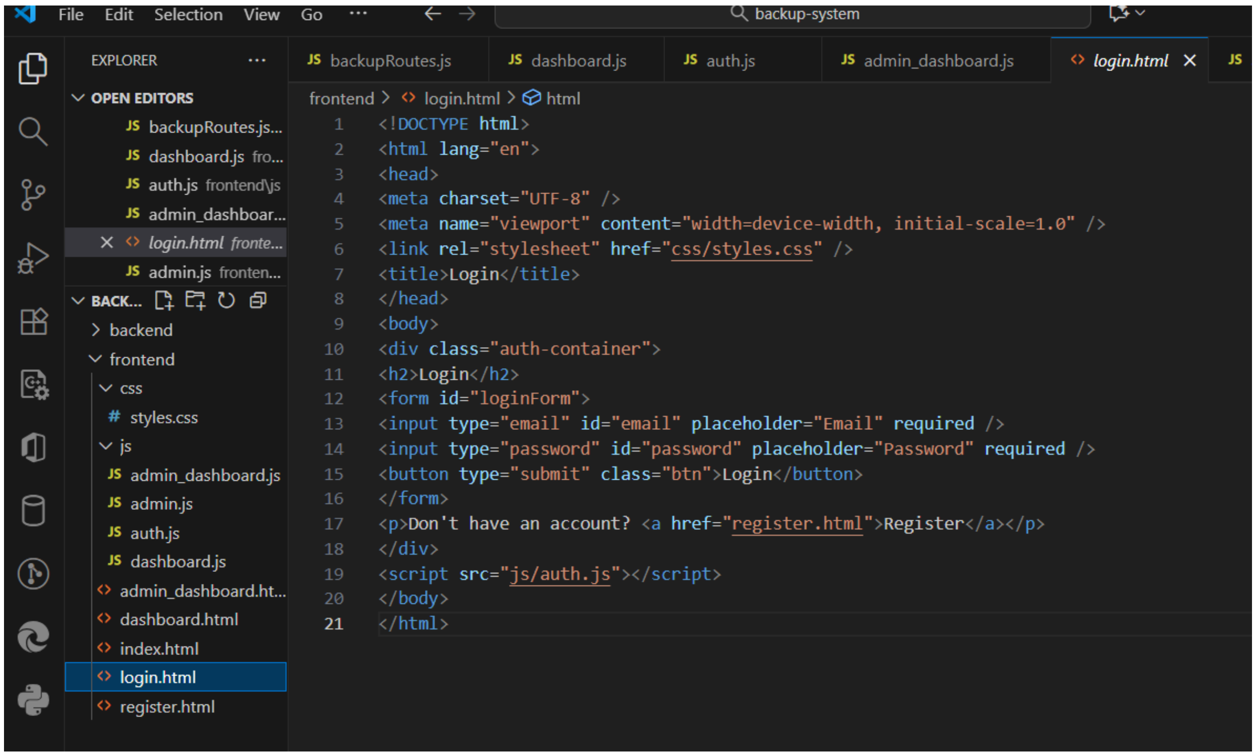
3.3 Logic Development

The logic of the system is implemented using **Node.js** and **Express.js**. This layer handles authentication, file uploads, database backup, and log management. Data is stored in a **MySQL** database, and backups are created using the mysqldump utility.

3.3.1 User Authentication Logic

The authentication logic in authRoutes.js manages registration and login. It uses hashed passwords for security and JSON Web Tokens (JWT) for session management.

Figure 3.3.1.1: *Authentication Logic Code Snippet*



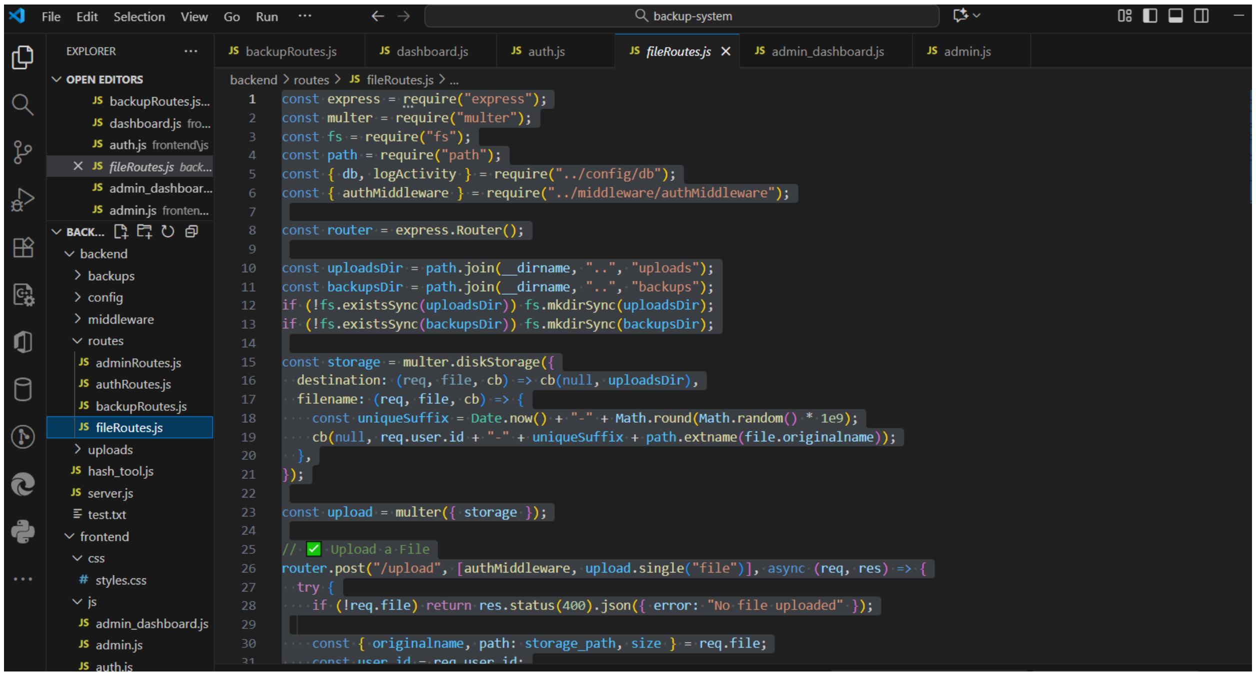
```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="css/styles.css" />
<title>Login</title>
</head>
<body>
<div class="auth-container">
<h2>Login</h2>
<form id="loginForm">
<input type="email" id="email" placeholder="Email" required />
<input type="password" id="password" placeholder="Password" required />
<button type="submit" class="btn">Login</button>
</form>
<p>Don't have an account? <a href="register.html">Register</a></p>
</div>
<script src="js/auth.js"></script>
</body>
</html>
```

(Screenshot showing login and JWT generation code)

3.3.2 File Upload and Management Logic

The file upload feature is managed by fileRoutes.js, which uses the **Multer** library to handle file uploads. Each file uploaded by a user is stored in the /uploads folder, and its details are recorded in the database.

Figure 3.3.2.1: *File Upload Logic Code*



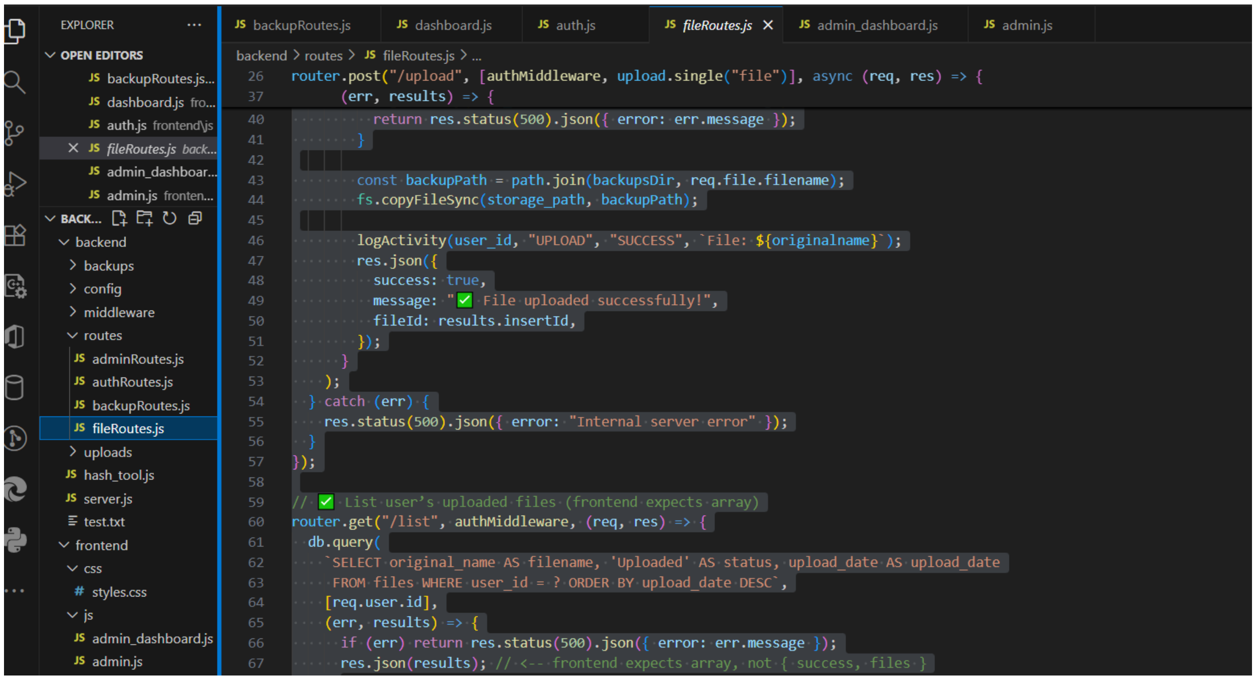
```javascript
const express = require("express");
const multer = require("multer");
const fs = require("fs");
const path = require("path");
const { db, logActivity } = require("../config/db");
const { authMiddleware } = require("../middleware/authMiddleware");

const router = express.Router();

const uploadsDir = path.join(__dirname, "..", "uploads");
const backupsDir = path.join(__dirname, "..", "backups");
if (!fs.existsSync(uploadsDir)) fs.mkdirSync(uploadsDir);
if (!fs.existsSync(backupsDir)) fs.mkdirSync(backupsDir);

const storage = multer.diskStorage({
  destination: (req, file, cb) => cb(null, uploadsDir),
  filename: (req, file, cb) => {
    const uniqueSuffix = Date.now() + "-" + Math.round(Math.random() * 1e9);
    cb(null, req.user.id + "-" + uniqueSuffix + path.extname(file.originalname));
  },
});

const upload = multer({ storage });

// ✅ Upload a File
router.post("/upload", [authMiddleware, upload.single("file")], async (req, res) => {
  try {
    if (!req.file) return res.status(400).json({ error: "No file uploaded" });

    const { originalname, path: storage_path, size } = req.file;
    const user_id = req.user.id;
```

(Screenshot showing multer setup and database query)

3.3.3 Backup Creation Logic

The database backup functionality is handled in backupRoutes.js. The system uses the mysqldump command to create .sql dump files and saves them in the /backups directory. Users can manually initiate backups or schedule them for automation.

Figure 3.3.3.1: *Database Backup Code Snippet*



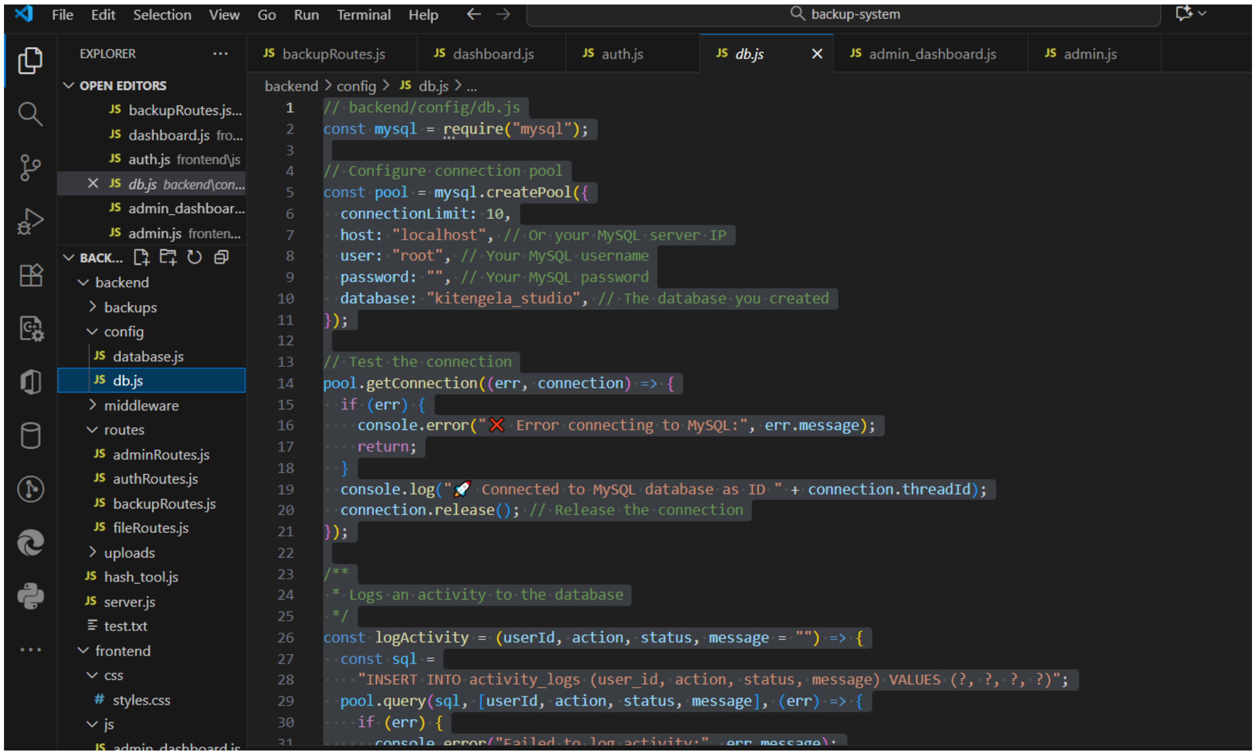
```
router.post("/upload", [authMiddleware, upload.single("file")], async (req, res) => {
      (err, results) => {
          return res.status(500).json({ error: err.message });
        }

        const backupPath = path.join(backupsDir, req.file.filename);
        fs.copyFileSync(storage_path, backupPath);

        logActivity(user_id, "UPLOAD", "SUCCESS", `File: ${originalname}`);
        res.json({
          success: true,
          message: "✅ File uploaded successfully!",
          fileId: results.insertId,
        });
      }
    );
  } catch (err) {
    res.status(500).json({ error: "Internal server error" });
  }
});

// ✅ List user's uploaded files (frontend expects array)
router.get("/list", authMiddleware, (req, res) => {
  db.query(
    `SELECT original_name AS filename, 'Uploaded' AS status, upload_date AS upload_date
     FROM files WHERE user_id = ? ORDER BY upload_date DESC`,
    [req.user.id],
    (err, results) => {
      if (err) return res.status(500).json({ error: err.message });
      res.json(results); // <-- frontend expects array, not { success, files }
```

(Screenshot showing exec(mysqldump) command)

3.3.4 Database Configuration Logic

The database connection and activity logging logic are defined in config/db.js. It establishes a secure connection pool to the MySQL database and provides a logging function for recording system activities.

Figure 3.3.4.1: *Database Configuration Code (db.js)*



```javascript
// backend/config/db.js
const mysql = require("mysql");

// Configure connection pool
const pool = mysql.createPool({
  connectionLimit: 10,
  host: "localhost", // Or your MySQL server IP
  user: "root", // Your MySQL username
  password: "", // Your MySQL password
  database: "kitengela_studio", // The database you created
});

// Test the connection
pool.getConnection((err, connection) => {
  if (err) {
    console.error("❌ Error connecting to MySQL:", err.message);
    return;
  }
  console.log("🚀 Connected to MySQL database as ID " + connection.threadId);
  connection.release(); // Release the connection
});

/**
 * Logs an activity to the database
 */
const logActivity = (userId, action, status, message = "") => {
  const sql =
    "INSERT INTO activity_logs (user_id, action, status, message) VALUES (?, ?, ?, ?)";
  pool.query(sql, [userId, action, status, message], (err) => {
    if (err) {
      console.error("Failed to log activity:", err.message);
```

(Screenshot showing mysql.createPool code)

3.4 Testing

Testing is carried out to verify that all system modules work as intended. Both frontend and backend components are tested using manual input, Postman API requests, and browser console logs.

Feature tested	Testing method	Expected output	Results
User registration	Form submission	User data store in database	passed
Login	Login form and postman	Redirects user to dashboard	Passed
File upload	Dashboard upload	File save in/uploads	Passed
Backup creation	Backup now button	.sql file saved in /backups	Passed
Admin approval	Admin panel Action	User marked as approved	Passed

Detail out what actions you took to test whether different features of your system are working. And corrections you made to different parts to ensure that they are working as planned.

3.5 Deployment

The system runs on a **local Node.js server** using port **5000**. The backend is connected to a MySQL database hosted in XAMPP. The frontend files are served using Express middleware.

Deployment steps include:

1. Installing all required dependencies using npm install.
2. Setting up the database connection in config/db.js.
3. Starting the MySQL service using XAMPP.
4. Running the backend server using node server.js.
5. Accessing the system through http://localhost:5000.

Mention on what platform have you deployed your system (e.g. if it's a website, what web hosting platform did you use – and copy paste a link to your website, if it's an android application – the process of putting up your app on Google Playstore.). Essentially, detail out the process of how you transformed your project into a form that can be run/installed by anybody.