

# **PHOTOCOPY FILE SHARING SYSTEM**

**22CDP61 Project Work I**

**Submitted by**

**NAGULAN D**

**22CDR059**

**PRAVEEN KUMAR C**

**22CDR073**

**SATHEESH K**

**22CDR089**

*in partial fulfilment of the requirements for the award of the degree  
of*

**BACHELOR OF ENGINEERING IN COMPUTER SCIENCE  
AND DESIGN**

**DEPARTMENT OF COMPUTER SCIENCE AND DESIGN**



**KONGU ENGINEERING COLLEGE**

**(Autonomous) PERUNDURAI, ERODE – 638 060**

**MAY 2025**

**DEPARTMENT OF COMPUTER SCIENCE AND DESIGN**  
**KONGU ENGINEERING COLLEGE**  
(AUTONOMOUS)  
**PERUNDURAI, ERODE 638060**  
**MAY 2025**

**BONAFIDE CERTIFICATE**

This is to certify that the Project report entitled **PHOTOCOPY FILE SHARING SYSTEM** is the bonafide record of project work done by **NAGULAN D (Reg no: 22CDR059)**, **PRAVEEN KUMAR C (Reg no: 22CDR073)** and **SATHEESH K (Reg no: 22CDR089)** in partial fulfilment of the requirements for the award of the Degree of Bachelor of Engineering in **Computer Science and Design** of Anna University, Chennai during the year 2024 - 2025.

**SUPERVISOR**

**HEAD OF THE DEPARTMENT**  
(Signature with seal)

**Date:**

Submitted for the end semester viva-voice examination held on \_\_\_\_\_

**INTERNAL EXAMINER**

**EXTERNAL EXAMINER**

**DEPARTMENT OF COMPUTER SCIENCE AND DESIGN**  
**KONGU ENGINEERING COLLEGE**  
**(Autonomous)**

**PERUNDURAL, ERODE 638060**  
**MAY 2025**

**DECLARATION**

We affirm that the Project report titled **PHOTOCOPY FILE SHARING SYSTEM** being submitted in partial fulfillment of the requirements for the award of Degree of Bachelor of Engineering is the original work carried out by us. It has not formed part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

**Date:**

**NAGULAN D**  
**(Reg No:22CDR059)**

**PRAVEENKUMAR C**  
**(Reg No:22CDR073)**

**SATEESH K**  
**(Reg No:22CDR089)**

I certify that the declaration made by the above candidates is true to the best of my knowledge.

**Date:**

**Name & Signature of the Supervisor with seal**

## **ABSTRACT**

This project introduces a QR code-based file-sharing system tailored for photocopy shops to improve file transfer efficiency and security. Traditional methods like email, WhatsApp, or pen drives are often slow, internet-dependent, and raise privacy concerns. To overcome these limitations, the proposed system enables fast and secure file transfers over a local network, eliminating the need for internet connectivity. By scanning a QR code, customers can instantly send their documents to the shop's computer. The system uses React.js for the frontend and Node.js with Express.js on the backend, while WebSocket technology handles real-time file transfers. This ensures a seamless and responsive user experience. Additionally, the system includes features for tracking daily income, making it a comprehensive solution for shop operations. It modernizes the customer experience and reduces delays associated with conventional methods. Overall, the project aims to increase efficiency, enhance security, and streamline shop management through a simple yet powerful tool.

## ACKNOWLEDGMENT

We express our sincere thanks and gratitude to our beloved Correspondent, **Thiru.A.K.ILANGO, B.Com., MBA., L.L.B.**, for giving us the opportunity to pursue this course.

We are extremely thankful with no words of formal nature to the dynamic Principal, **Dr.V.BALUSAMY, MTech, Ph.D.**, for providing the necessary facilities to complete our work.

We would like to express our sincere gratitude to our respected Head of the Department of Computer Science and Design **Dr.R.THANGARAJAN, M.E., Ph.D.**, for providing necessary facilities.

We also express our sincere thanks to our project coordinator, **Dr.P.GOWSIKRAJA, M.E, Assistant Professor** for her constant encouragement for the development of our project.

We thank our guide **Dr.K.R Prasanna Kumar M.E., Ph.D, Associate Professor** for her encouragement and valuable advice that made us carry out the project work successfully. Her valuable ideas and suggestions have been very helpful in the project. We are grateful to all the faculty members of the Computer Science and Design Department for their support.

## TABLE OF CONTENTS

CHAPTER	TITLE	PAGE NO
	ABSTRACT	iv
1	INTRODUCTION	1
	1.1 OVERVIEW OF DIGITAL RESTAURANT SYSTEM	1
	1.2 EXISTING SYSTEM	1
	1.2.1 MANUAL INVENTORY AND BILLING	1
	1.3 PROPOSED SYSTEM	2
	1.3.1 USER MODULE	3
	1.3.2 ADMIN MODULE	3
	1.3.3 SYSTEM ARCHITECTURE	4
2	LITERATURE REVIEW	5
	2.1 OVERVIEW	5
	2.2 REVIEW OF RELATED WORKS	5
3	REQUIREMENTS SPECIFICATION	7
	3.1 HARDWARE REQUIREMENTS	7
	3.2 SOFTWARE REQUIREMENTS	7
	3.3 SOFTWARE DESCRIPTION	8
	3.3.1 VISUAL STUDIO CODE (VS CODE)	8
	3.3.2 REACT JS	9
	3.3.3 NODE.JS AND EXPRESS.JS	9
	3.3.4 MySQL DB	9
	3.3.5 GIT AND GITHUB	9

<b>CHAPTER</b>	<b>TITLE</b>	<b>PAGE NO</b>
4	SYSTEM DESIGN AND IMPLEMENTATION	10
	4.1 SYSTEM ARCHITECTURE	10
	4.2 MODULES OF THE SYSTEM	11
	4.2.1 INVENTORY MANAGEMENT MODULE	11
	4.2.2 BILLING SYSTEM MODULE	12
	4.2.3 ALERTS AND NOTIFICATION MODULE	13
	4.3 WORKFLOW DIAGRAMS	14
5	RESULTS AND DISCUSSION	16
	5.1 TESTING STRATEGIES	16
	5.2 PERFORMANCE ANALYSIS	17
	5.3 SYSTEM ADVANTAGES	18
6	CONCLUSION AND FUTURE ENHANCEMENT	20
	APPENDIX I	22
	APPENDIX II	33
	REFERENCES	35

# **CHAPTER 1**

## **INTRODUCTION**

### **1.1 OVERVIEW OF PHOTOCOPY FILE SHARING SYSTEM**

The photocopy shop file-sharing system is designed to replace traditional file transfer methods like Gmail or WhatsApp with a faster and more secure alternative. By scanning a QR code at the shop's counter, customers are redirected to a local web application where they can upload their documents. These files are instantly transferred to the shop's computer without relying on the internet. The system offers a seamless and offline method of sharing files, reducing delays and privacy risks. It simplifies the process for both customers and shop staff, making operations more efficient and user-friendly.

### **1.2 EXISTING SYSTEM**

The existing system in photocopy shops involves manual file transfers through email, WhatsApp, or USB drives. These methods depend on internet connectivity or physical handling, which can be slow and inconvenient. They often result in delays, especially when the internet is unreliable or devices are incompatible. There's also a higher risk of file loss, errors, and security breaches. This approach lacks efficiency and can compromise customer privacy. Overall, it creates unnecessary complications for both customers and shop operators.

#### **1.2.1 EXISTING FILE SHARING METHODS IN PHOTOCOPY SHOPS**

Currently, most photocopy shops rely on third-party applications and manual methods for receiving files from customers. Common practices include:

- Customers sending files via email, WhatsApp, or other messaging apps.
- Using physical storage devices like USB drives or memory cards.
- Transferring files via Bluetooth or other device-to-device sharing methods.



### **Limitations of Existing System:**

- **Dependency on Internet Connectivity:** Apps like WhatsApp and Gmail require a stable internet connection.
- **Privacy Risks:** Files shared via personal apps may compromise customer privacy.
- **Slow Transfer Speeds:** File transfer over the internet can be slow, especially with large documents.
- **Device Compatibility Issues:** Not all devices support the same file-sharing apps or methods.
- **Manual Effort:** Shop operators need to manually check for incoming files from various apps and devices.

### **1.3 PROPOSED SYSTEM**

The proposed system introduces a QR code-based method for file sharing in photocopy shops, aimed at improving efficiency, speed, and data security. Instead of relying on internet-based services like email or WhatsApp, this system operates within a local network. A unique QR code is displayed at the shop's counter, which customers can scan using their smartphones. Upon scanning, they are redirected to a locally hosted web application where they can easily upload the required files. These files are then instantly received by the shop's computer without needing internet connectivity. This eliminates common issues like slow transfers, compatibility problems, and privacy risks associated with third-party platforms. The system ensures a smoother, offline workflow and minimizes wait times for customers. It also reduces the manual effort for shop operators, offering a modern, user-friendly solution. By focusing on speed, convenience, and data privacy, the system greatly enhances the overall file-sharing process in photocopy shops.

### 1.3.1 USER MODULE

The User Module allows customers to scan the QR code and upload files through a web interface. It ensures a simple and quick file transfer process without needing internet access.

Key features of the user module include:

- **QR Code Scanning:**

Customers scan a unique QR code displayed at the shop's counter to access the file-sharing system.

- **File Upload:**

The module allows users to upload files directly from their devices to the shop's server through a simple web interface.

- **No Internet Required:**

The system operates over a local network, ensuring faster transfers without the need for internet connectivity.

- **User-Friendly Interface:**

The interface is designed for easy navigation, making it accessible even for those with minimal technical knowledge.

### 1.3.2 ADMIN MODULE

The Admin Module allows shop operators to manage incoming files, monitor system performance, and track daily transactions for efficient operations.

Key features of the admin module include:

- **File Management:**

Admins can view, organize, and manage files uploaded by customers for printing or photocopying.

- **System Monitoring:**

Admins can monitor the status of file transfers and ensure smooth operation of the system.

- **Income Tracking:**

The module allows for tracking daily income, helping to record and manage financial transactions.

- **User Access Control:**

Admins can manage user permissions and access to various system features to ensure secure operations.

### 1.3.3 SYSTEM ARCHITECTURE

The system architecture consists of a client-server model where users upload files via a web application, and the server processes and stores the files locally for printing or photocopying.

- **Frontend:**

The frontend is a web-based interface built using React.js, allowing users to interact with the system by scanning QR codes and uploading files through a simple and responsive interface.

- **Backend:**

The backend is developed using Node.js and Express.js, handling file uploads, managing server requests, and ensuring secure and efficient file transfers over the local network.

- **Database:**

The database stores customer file upload details, transaction records, and daily income data. It ensures efficient data management and retrieval, typically using MySQL for storage.

- **Security and Authentication:**

The system uses secure local network protocols for file transfer and implements authentication to restrict access to the admin panel, ensuring data privacy and preventing unauthorized actions.

## CHAPTER 2

### LITERATURE REVIEW

#### 2.1 Overview

In recent years, several file-sharing applications and platforms such as Gmail, WhatsApp, and SHAREit have been used for transferring documents in photocopy shops. However, these systems are either dependent on internet connectivity or require software installations on the customer's device. Moreover, they do not offer features tailored for shop environments, such as real-time receipt of files or integrated income management. Studies have shown that file-sharing systems using local networks and QR codes offer faster and more reliable transfer rates. This project aims to fill the gap by providing a dedicated, shop-oriented, offline file-sharing solution.

#### 2.2 Review of Related Works

J. Singh et al., (2023) in their paper titled "Comprehensive Review on Web-based Photocopy Shop File Sharing System" published in the *International Journal of Engineering Research & Technology* (IJERT), Volume 12, Issue 11, discussed the evolution, architecture, and benefits of web-based file-sharing systems for small businesses, specifically photocopy shops. The review explored key modules such as file upload handling, customer records, transaction management, and real-time file transfers. It highlighted the use of web development frameworks like HTML, CSS, JavaScript, React.js, and Node.js for creating efficient and scalable systems. Their study emphasized how these technologies support offline operations, security, and seamless user experience for photocopy shops, providing a significant improvement over traditional file transfer methods.

W. S. Gray and S. C. Liguori (2003), in their book titled "*Photocopy Shop Management and Operations*" (4th ed.), published by Prentice Hall PTR, offered a comprehensive guide to managing photocopy shop operations, including customer service, inventory, and financial controls. While the book is rooted in traditional management practices, its insights remain relevant for modern systems that now leverage web technologies. In the context of digital transformation, these operations are increasingly being implemented using web technologies like the **MERN stack** (MongoDB, Express.js, React.js, Node.js), enabling efficient, scalable, and interactive solutions in photocopy shop management.

## **CHAPTER 3**

### **REQUIREMENTS SPECIFICATION**

#### **3.1 HARDWARE REQUIREMENTS**

Processor Type : Intel i5

Speed : 3.4GHZ

RAM : 8GB DDR5 RAM

Hard disk : 500 GB

#### **3.2 SOFTWARE REQUIREMENTS**

Operating System : Windows 11 (64 bit)

Front end :HTML,CSS, React.js,javascript,Electron

Backend : Node js,Express js

Database : MySQL DB

Deployment : Offline system

Version Control : Github

## **3.3 SOFTWARE DESCRIPTION**

### **3.3.1 Visual Studio Code (VS Code)**

Visual Studio Code is a lightweight yet powerful source code editor that runs on your desktop. It comes with built-in support for JavaScript, Node.js, and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python) and runtimes.

#### **Features of VS Code**

- IntelliSense for smart code completion
- Built-in Git commands
- Extension marketplace for thousands of tools
- Debugging directly from the editor
- Lightweight and fast performance

### **3.3.2 ReactJS**

ReactJS is a popular JavaScript library for building user interfaces, especially single-page applications, where a fast and interactive user experience is essential. React makes it painless to create interactive UIs by efficiently updating and rendering components based on data changes.

#### **Features of ReactJS**

- Component-based architecture
- Virtual DOM for fast rendering
- Reusable components
- One-way data binding
- Strong community support

### **3.3.3 Node.js and Express.js**

Node.js is an open-source, cross-platform JavaScript runtime environment that executes JavaScript code outside of a browser. Express.js is a web application framework for Node.js, designed for building web applications and APIs.

#### **Features of Node.js and Express.js**

- Asynchronous and event-driven
- Fast execution using the V8 JavaScript engine
- Robust set of features for web and mobile applications

### **3.3.4 MySQL**

MySQL is an open-source, relational database management system (RDBMS) that stores data in structured tables using SQL (Structured Query Language). It provides a reliable, efficient, and widely-used database solution for web and enterprise applications.

#### **Features of MySQL**

- Structured, Schema-based Database Design
- High Performance and Reliability
- Rich query language
- Supports Vertical and Horizontal Scaling

### **3.3.5 Git and GitHub**

Git is a distributed version control system designed to handle everything from small to very large projects. GitHub is a hosting service for Git repositories, offering collaboration features like bug tracking, feature requests, task management, and wikis for projects.

#### **Features**

- Branching and merging
- Pull requests and code reviews

- Integrated issue tracking

### **3.3.6 ELECTRON**

Electron is an open-source framework that allows developers to build cross-platform desktop applications using web technologies like HTML, CSS, and JavaScript. In this project, it is used to create a dedicated, user-friendly desktop application for the photocopy shop's system, ensuring easy deployment and operation without complex setups.

#### **Features:**

- Cross-Platform Compatibility
- Runs Web Apps as Desktop Applications
- Simplified Application Deployment
- Improved Security and Control



## **CHAPTER 4**

### **SYSTEM DESIGN AND IMPLEMENTATION**

The system architecture is based on a client-server model operating over a local network. Customers scan a QR code using their devices, which redirects them to a React-based web interface. This interface allows them to upload files easily without internet access. The uploaded files are sent in real-time to a Node.js and Express.js server using WebSocket. The shop operator instantly receives these files on their system for printing or photocopying. Additionally, transaction details can be logged into a MySQL database for record-keeping and management.

#### **4.1 SYSTEM ARCHITECTURE**

The system uses a client-server model where customers scan a QR code to access a React.js frontend hosted locally. Files uploaded through the interface are transferred in real-time to a Node.js and Express.js backend using WebSocket. The shop operator instantly receives the files for processing. Transaction data can be optionally stored in a MySQL database.

##### **Components of the Architecture:**

##### **1. Frontend (Client-side):**

- React.js is used to build an intuitive and responsive interface where users can easily upload files after scanning the QR code.
- It ensures smooth communication with the backend, providing real-time feedback and a hassle-free file-sharing experience.

##### **2. Backend (Server-side):**

- Node.js and Express.js manage incoming file uploads from the frontend, storing them securely on the local server.
- They handle routing, request processing, and communication with the frontend and database for efficient system operation.

### 3. Database:

- MySQL stores important information such as transaction logs, file upload records, and user activity for future reference.
- It enables easy retrieval, organization, and analysis of records to support reporting and shop management tasks.

### 4. Communication:

- Real-time communication between the frontend and backend is achieved using WebSocket, enabling instant file transfers without page reloads.
- All communication occurs within a secure local network, ensuring fast, offline data exchange between customer devices and the shop system.

### 5. Deployment:

- The entire application is deployed on a local machine within the shop using Node.js, allowing it to run without internet access.
- Customers access the application via a QR code that points to the local IP address, enabling quick and easy deployment for daily use.

## 4.2 MODULES OF THE SYSTEM

The system consists of the **User Module**, allowing customers to upload files via QR code, and the **Admin Module**, enabling operators to manage files and transactions. It also includes a **File Transfer Module** for real-time uploads and a **Database Module** for storing records and data.

### 4.2.1 File Upload & Transfer Module

This module manages the seamless and instant transfer of documents from the customer's device to the shop's system over a local network.

#### Key Features:

- **QR Code Access:** Customers scan a QR code to access the file upload portal.
- **Instant File Transfer:** Documents uploaded by customers are instantly transferred

to the shop's computer.

- **Upload History:** Maintains a record of all uploaded files for processing and future reference.

**Workflow:**

1. Customer scans QR code → Uploads file via local web app.
2. File is instantly transferred to the server.
3. Operator receives file notification and processes it for printing.

#### **4.2.2 Transaction & Billing Module**

Handles recording of photocopy service transactions, daily income summaries, and bill generation for customers.

**Key Features:**

- **Transaction Recording:** Captures the amount collected for each print job.
- **Invoice Generation:** Automatically generates bills detailing number of copies, price, and total amount.
- **Daily Income Summary:** Provides shop operators with a daily report of income generated.

**Workflow:**

1. Operator selects the uploaded file and enters print details.
2. System calculates total cost and generates an invoice.
3. Payment is recorded; daily income report is updated.

### 4.2.3 Database Management Module

Stores all file records, transaction data, and system logs securely in the database.

#### Key Features:

- **File Metadata Storage:** Keeps track of uploaded files, customer details (if required), and print status.
- **Transaction Data Management:** Maintains records of income, invoices, and daily summaries.
- **System Log Management:** Logs events like file uploads, errors, and alerts.

#### Workflow:

1. File uploads and transactions are stored in the database.
2. System retrieves data for reporting and processing.
3. Records are maintained for analysis and future auditing.

## 4.3 WORKFLOW DIAGRAMS

To further clarify the system's operations, we present the following workflow diagrams for key processes within the system:

### 1. File Upload & Transfer Module

- Customer scans QR code → Uploads file via local web app → File instantly transferred to shop system → Operator receives notification → Operator processes file for printing

### 2. Transaction & Billing Module

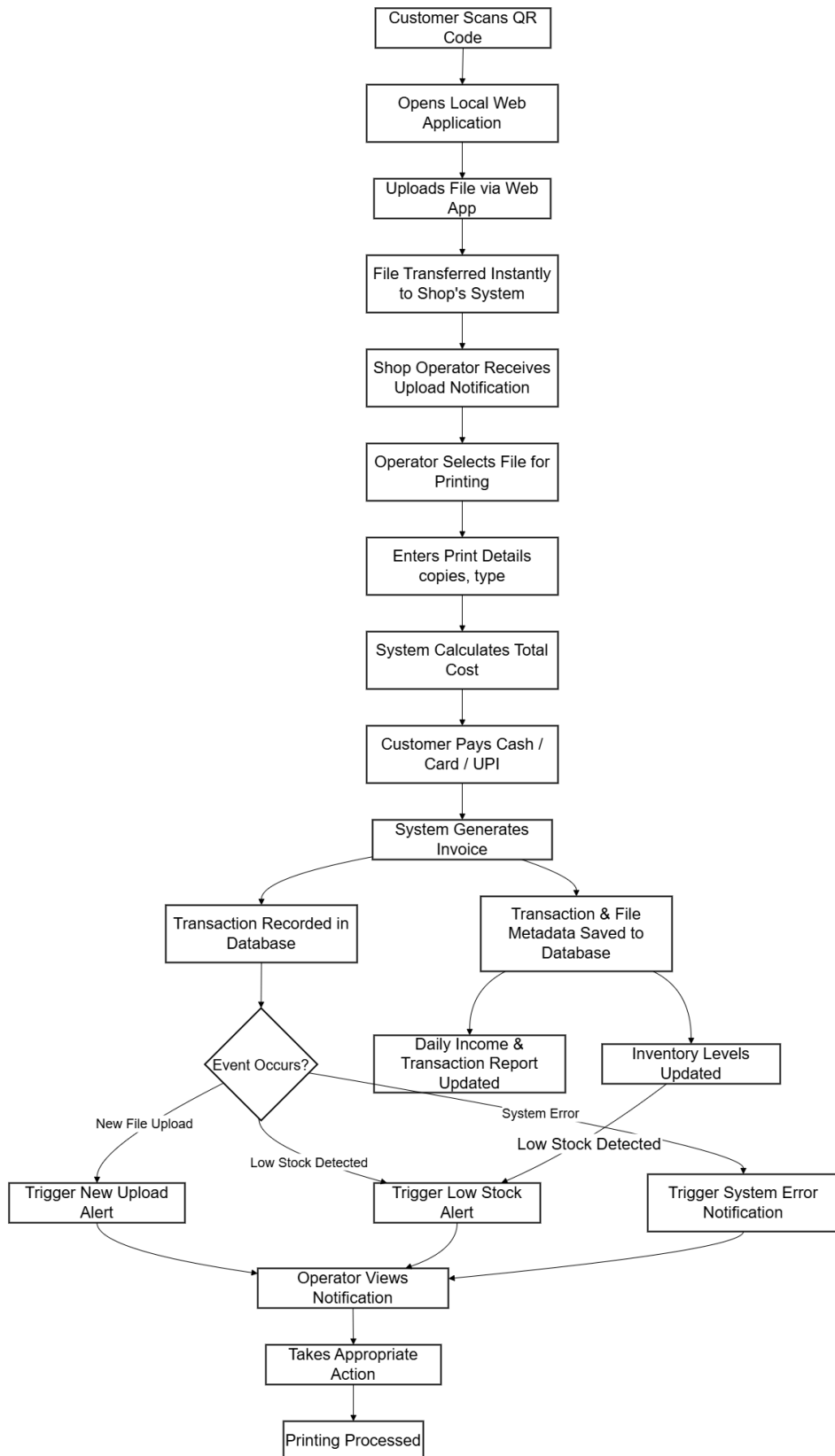
- Operator selects uploaded file → Enters print details → System calculates total

cost → Invoice generated → Payment recorded → Daily income report updated

### **3. Database Management Module**

- File uploads and transactions stored in database → System retrieves data for reporting/processing → Records maintained for analysis and future audit

# Flowchart:



## **CHAPTER 5**

### **RESULTS AND DISCUSSION**

This chapter discusses the results of the **Photoshop File Sharing System**, implementing the QR-based file-sharing system, highlighting its efficiency in improving speed, security, and ease of use in photocopy shops. It also discusses user feedback, system performance, and areas for future enhancement.

#### **5.1 TESTING STRATEGIES**

The system was tested using functional, performance, and user acceptance testing to ensure all modules work correctly. Real-time scenarios were simulated to check file transfers, notifications, and system stability.

##### **Types of Testing Conducted:**

##### **1. Unit Testing:**

- Each individual component, such as file upload, billing, and alerts, was tested separately to ensure correct functionality.
- Helped identify and fix errors early in development, improving the reliability of each module.

##### **2. Integration Testing:**

- Tested how different modules (e.g., frontend, backend, database) work together as a complete system.
- Ensured smooth and accurate data transfer between components like file upload, billing, and inventory.
- Checked the system's ability to handle integration errors and respond appropriately.

### **3. UI/UX Testing:**

- Assessed the design and layout to ensure the system is intuitive and easy to navigate for customers and shop operators.
- Collected feedback from actual users to identify areas for improvement in terms of responsiveness, accessibility, and overall user satisfaction.

### **4. Performance Testing:**

- Tested the system's ability to handle file uploads, transactions, and notifications under various load conditions to ensure fast performance.
- Evaluated how the system behaves under high traffic or simultaneous requests, identifying potential bottlenecks or areas for optimization.

### **5. Security Testing:**

- Tested the system for vulnerabilities such as unauthorized access, ensuring that user data and files are securely handled and stored.
- Verified that file transfers, user login processes, and sensitive information are encrypted and protected against potential cyber threats.

## **5.2 PERFORMANCE ANALYSIS**

The performance of the Photoshop File Sharing System system assesses the system's efficiency, responsiveness, and stability under varying conditions, ensuring optimal performance for smooth operation.



## **Key Performance Metrics:**

### **1. Response Time:**

- Measures the time taken for a file to be uploaded from the customer's device to the server, ensuring quick and efficient transfers.
- Evaluates the time required to process customer transactions and generate invoices, ensuring smooth and timely billing operations.

### **2. Scalability:**

- Evaluates the system's ability to handle a growing number of simultaneous file uploads, transactions, and users without performance degradation.
- Assesses how well the system utilizes resources (e.g., server capacity and bandwidth) as the workload increases, ensuring smooth scaling as business demands grow.

### **3. Scalability:**

- Assesses the system's ability to accommodate increased user load, file transfers, and transactions as the photocopy shop grows.
- Ensures the system can scale without compromising performance by efficiently managing server resources and bandwidth under higher demand.

### **4. Load Testing:**

- Tests the system's performance by simulating a large number of simultaneous users or file uploads to assess its stability under heavy load.
- Identifies potential performance issues or bottlenecks that could arise when the system is under stress, allowing for optimization before actual deployment.

### **5. Resource Utilization:**

- Monitors the system's use of CPU and memory during normal and peak load conditions to ensure efficient performance without overloading the server.

### 5.3 SYSTEM ADVANTAGES

The Photocopy File Sharing System offers several advantages that enhance the customer and administrative experience.

#### **Key Advantages:**

**1. Faster File Transfer:**

Customers can quickly upload files by scanning a QR code. No need for email or pen drives. It saves time and speeds up shop service.

**2. Better Security:**

Files move safely within the shop's local network. No internet needed, keeping customer data private and safe.

**3. Works Without Internet:**

The system runs offline using a local server. No worries about poor or no internet — everything still works fine.

**4. Instant Processing:**

Files are received and ready for printing right away. No waiting, faster service for customers.

**5. Simple to Use:**

Customers just scan, upload, and they're done. No technical knowledge needed. Easy for operators too.

**6. Saves Costs:**

No need for Gmail, cloud storage, or pen drives. It cuts down internet bills and extra device costs for the shop.

**7. Ready to Grow:**

The system can handle more customers, more files, and extra features as the shop grows — no major changes needed.

## CHAPTER 6

### CONCLUSION AND FUTURE WORKS

This chapter presents the conclusion of the Photocopy File Sharing System and outlines potential enhancements for future versions of the platform.

#### 3.1 CONCLUSION

The **Photocopy File Sharing System** offers an efficient and modern solution for photocopy shops by allowing customers to upload files directly via QR codes, eliminating the need for third-party services like email or WhatsApp. This ensures faster, secure, and hassle-free file transfers, improving overall service speed and customer satisfaction. Operating offline, it works seamlessly even in areas with poor internet connectivity, making it reliable for all locations.

With real-time file processing, the system enables immediate access to uploaded files, allowing operators to quickly print or photocopy documents. It also features integrated billing and transaction tracking, making income management more efficient. The system ensures the security of data through local network transfers, reducing risks associated with online platforms.

In conclusion, this system significantly enhances the operational efficiency and customer experience in photocopy shops. It is adaptable to future growth, offering additional features like reporting tools or integrations with other business systems, making it a comprehensive solution for the evolving demands of modern businesses.

#### 3.2 FUTURE ENHANCEMENTS

While the Digital Restaurant Experience is effective in its current form, there are several areas where the system can be enhanced in future iterations.

1. **Cloud Integration:**

Enabling the system to back up files to the cloud, providing a secure, offsite storage option and making files accessible from anywhere.

2. **Mobile Application:**

Developing a dedicated mobile app for customers, allowing them to upload files more conveniently and track their transaction status in real-time.

3. **Advanced Analytics:**

Incorporating data analytics for detailed reports on customer behavior, sales trends, and operational efficiency to help shop owners make informed business decisions.

4. **Payment Gateway Integration:**

Adding multiple payment gateway options (e.g., credit/debit cards, mobile wallets) for seamless and secure online payment processing directly through the system.

5. **Customer Accounts:**

Allowing customers to create accounts to store past transactions, manage preferences, and access loyalty programs, enhancing their overall experience.

6. **AI-Based Recommendations:**

Implementing AI to analyze customer preferences and suggest additional services, such as document formatting or design enhancements, to increase shop revenue.

7. **Multilingual Support:**

Expanding the system to support multiple languages, making it more accessible to a broader range of customers in diverse regions.

## APPENDIX I

### CODING:

App.jsx

```
import React, { useEffect, useState } from "react";
import FileUploadApp from "../components/FileUploadApp";
import FileViewPage from "../components/FileViewPage";
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import Home from "../components/Home";
import "../App.css"
function App() {
  const [ipAddress, setIpAddress] = useState({
    ip: "",
    status: false,
  });

  useEffect(() => {
    const ip = window.location.hostname; // Get the hostname (IP address or domain name)
    setIpAddress({ ip: ip, status: true });
  }, []);
  console.log(ipAddress);
  return (
    <
      <Router>
        <Routes>

          {ipAddress.status && (
            <
              <Route path="/" element={<FileUploadApp ip={ipAddress.ip}/>} />
              <Route path="/admin" element={<FileViewPage ip={ipAddress.ip}/>} />
            </>
          )}
        </Routes>
      </Router>
    </>
  );
}

export default App;
```

Home.jsx

```
import { useNavigate } from "react-router-dom";
import "../App.css"
const Home = () => {

  const navigate = useNavigate();
```

```

const handleAdmin = () => {
  const password = window.prompt("Enter admin password");
  if (password === "admin")
    navigate("/admin");
  return;
}
return (
  <div className="home">
    <div>Choose the type of User</div>
    <div className="home-buttons">
      <div>
        <button onClick={() => navigate("/upload")}>Public User</button>
      </div>
      <div>
        <button onClick={handleAdmin}>Admin</button>
      </div>
    </div>
  </div>
)
}
export default Home;

```

FileViewPage.jsx

```

import React, { useState, useEffect } from "react";
import { useNavigate } from "react-router-dom";
import io from "socket.io-client";
import Swal from "sweetalert2"; // Import SweetAlert2
import PriceCalculator from "../PriceCalculator";
import "../style.css";
import NavBar from "../Navbar";
import SettingsPage from "../SettingsPage";

const FileViewPage = ({ ip }) => {
  const [files, setFiles] = useState([]);
  const [price, setPrice] = useState([]);
  const [error, setError] = useState(null);
  const [toggle, setToggle] = useState(false);
  const [validUser, setValidUser] = useState(false);
  const [navbarData, setNavbarData] = useState("dashboard");
  const navigate = useNavigate();

  useEffect(() => {
    if (!validUser) {
      Swal.fire({
        title: "Admin Login",
        text: "Enter the admin password to continue:",
        input: "password",
        inputAttributes: {
          autocapitalize: "off",

```

```

    },
    showCancelButton: false,
    confirmButtonText: "Submit",
    allowOutsideClick: false,
    allowEscapeKey: false,
    preConfirm: (password) => {
      if (password !== "admin") {
        Swal.fire({
          icon: "error",
          title: "Access Denied",
          text: "Incorrect password. Redirecting...",
        }).then(() => {
          window.location.href = "/";
        });
      } else {
        setValidUser(true);
      }
    },
  });
}

if (!ip) {
  console.warn("IP address not provided");
  setError("Server IP not provided");
  return;
}

const socket = io(`http://${ip}:3000`);

socket.on("fileList", (fileMetadata) => {
  console.log("Received file metadata:", fileMetadata);
  setFiles(fileMetadata);
});

socket.on("dataList", (dataList) => {
  console.log("Received data list:", dataList);
  setPrice(dataList);
});

socket.on("connect_error", (err) => {
  console.error("WebSocket error:", err);
  Swal.fire({
    icon: "error",
    title: "Connection Error",
    text: "Error connecting to the server. Please check your connection.",
  });
  setError("Error connecting to the server");
});

```

```

return () => {
  socket.off("fileList");
  socket.off("dataList");
  socket.disconnect();
};
}, [ip]);

const handlePrint = (file) => {
  Swal.fire({
    title: "Print Confirmation",
    text: `Are you sure you want to print the file "${file.fileName}"?`,
    icon: "question",
    showCancelButton: true,
    confirmButtonText: "Yes, Print",
    cancelButtonText: "Cancel",
  }).then((result) => {
    if (result.isConfirmed) {
      const fileURL = `http://${ip}:3000/uploads/${file.fileName}`;

const printWindow = window.open("", '_blank');
if (printWindow) {
  printWindow.document.write(`
    <html>
      <head>
        <title>Print</title>
      </head>
      <body style="margin:0">
        <iframe src="${fileURL}" style="border:none; width:100%; height:100vh;"
onload="this.contentWindow.focus(); this.contentWindow.print();"></iframe>
      </body>
    </html>
  `);
  printWindow.document.close();
} else {
  console.error("Failed to open print window");
}

    Swal.fire({
      icon: "success",
      title: "Print Started",
      text: `The file "${file.fileName}" is being printed.`,
    });
  });
};

let totalRevenue = 0;
if (price?.length > 0) {
  for (let i = 0; i < price.length; i++) {
    if (price[i].price !== "Unknown") totalRevenue += Number(price[i].price);
  }
}

```



```

    console.log(price[i].price);
  }
}

```

```

console.log(navbarData);

```

```

return (

```

```

  <

```

```

    <NavBar setNavbarData={setNavbarData} />

```

```

    {validUser &&

```

```

      (navbarData === "dashboard" ? (

```

```

        <

```

```

          <div className="container">

```

```

            <h1>List of Files to be Printed</h1>

```

```

            {files.length === 0 ? (

```

```

              <p>No files found</p>

```

```

            ) : (

```

```

              <table className="table">

```

```

                <thead>

```

```

                  <tr>

```

```

                    <th>Name</th>

```

```

                    <th>File Name</th>

```

```

                    <th>Pages</th>

```

```

                    <th>Layout</th>

```

```

                    <th>Color</th>

```

```

                    <th>Pages Per Sheet</th>

```

```

                    <th>Price</th>

```

```

                    <th>Uploaded Time</th>

```

```

                    <th>Print</th>

```

```

                  </tr>

```

```

                </thead>

```

```

                <tbody>

```

```

                  {files.map((file, index) => (

```

```

                    <tr key={file.uniqueFileName || index}>

```

```

                      <td>{file?.fileType?.name || "N/A"}</td>

```

```

                      <td>{file.fileName}</td>

```

```

                      <td>{file?.fileType?.pages || "N/A"}</td>

```

```

                      <td>{file?.fileType?.layout || "N/A"}</td>

```

```

                      <td>{file?.fileType?.color || "N/A"}</td>

```

```

                      <td>{file?.fileType?.pagePerSheet || "N/A"}</td>

```

```

                      <td>

```

```

                        {file?.fileType?.price === "Unknown"

```

```

                          ? "1.5"

```

```

                          : file?.fileType?.price || "0"}

```

```

                      </td>

```

```

                      <td>{new Date(file.uploadTime).toLocaleString()}</td>

```

```

                      <td>

```

```

                        <button onClick={() => handlePrint(file)}>

```

```

                          Print

```

```

        </button>
      </td>
    </tr>
  )}
</tbody>
</table>
)}
<div className="total-revenue">
  Total Revenue: <span>{totalRevenue.toFixed(2)}</span>
</div>
<div onClick={() => setToggle(!toggle)} className="toggle-button">
  View Revenue at range
</div>
{toggle && (
  <div className="price-calculator">
    <PriceCalculator data={price} />
  </div>
)}
</div>
</>
): navbarData === "revenue" ? (
  <
    <PriceCalculator data={price} />
  </>
): (
  <SettingsPage ip={ip} />
)}
</>
);
};

```

export default FileViewPage;

FileUploadApp.jsx

```

import React, { useState, useEffect, useMemo } from "react";
import { io } from "socket.io-client";
import "./style.css";
import axios from "axios";

```

```

function FileUploadApp({ ip }) {
  const [socket, setSocket] = useState(null);
  const [price, setPrice] = useState({
    blackPrice: 1,
    colorPrice: 5,
  });
  const [userInfo, setUserInfo] = useState({ ipAddress: "" });
  const [file, setFile] = useState(null);
  const [status, setStatus] = useState("upload");
  const [fileType, setFileType] = useState({

```

```

name: "",
pages: "",
pagePerSheet: 1,
layout: "portrait",
color: "black",
price: 0,
flip: false,
});
useEffect(() => {
  if (ip) {
    const newSocket = io(`http://${ip}:3000`);
    setSocket(newSocket);

    return () => newSocket.close();
  }
}, [ip]);
useEffect(() => {
  const fetchPricing = async () => {
    try {
      const response = await axios.get(`http://${ip}:3000/pricing`);
      setPrice({
        blackPrice: response.data.blackPrice,
        colorPrice: response.data.colorPrice,
      });
    } catch (error) {
      console.log(error);
    }
  };
  if (ip) {
    fetchPricing();
  }
}, []);

const calculatePrice = () => {
  let temp_pages = fileType.pages;
  let numPages = 1;

  if (temp_pages.includes("-")) {
    const arr = temp_pages.split("-");
    console.log(arr);
    numPages = Number(arr[1]) - Number(arr[0]) + 1;
  } else if (!isNaN(Number(temp_pages)) && temp_pages.trim() !== "") {
    numPages = Number(temp_pages);
  }
  console.log(price);
  console.log(numPages);
  const copyPrice =
    (numPages / fileType.pagePerSheet) *
    (fileType.color === "black" ? price.blackPrice || 1 : price.colorPrice || 5);

```

```

console.log("copyPrice"+copyPrice);
if (fileType.flip) {
  return Math.round(copyPrice) + copyPrice - copyPrice / 2;
}
if (!isFinite(copyPrice)) {
  return 1;
}
return copyPrice;
};

```

```

const calculatedPrice = useMemo(() => calculatePrice(), [fileType, file]);
console.log(calculatedPrice);
const handleFileChange = (e) => {
  setFile(e.target.files[0]);
};

```

```

const handleUpload = () => {
  const valuePrice = calculatePrice();
  if (!file) {
    alert("Please select a file!");
    return;
  }
}

```

```

setFileType((prev) => ({ ...prev, price: valuePrice + fileType.price }));
setStatus("uploading");

```

```

const reader = new FileReader();

```

```

reader.onload = () => {
  const arrayBuffer = reader.result;
  socket.emit("uploadFile", {
    fileName: file.name,
    fileData: arrayBuffer,
    fileType,
    userInfo,
  });
  alert("File uploaded successfully!");
  socket.on("uploadSuccess", () => {
    setStatus("Done");
  });
  setStatus("Done");
};

```

```

reader.readAsArrayBuffer(file);
};

```

```

const handleInputChange = (field, value) => {
  setFileType((prev) => ({ ...prev, [field]: value }));
}

```

```
};
```

```
return (  
  <div className="file-upload-container">  
    <div className="file-upload-app">  
      <div className="header">  
        <h1>File Upload</h1>  
      </div>  
      <div className="input-group">  
        <input  
          type="text"  
          className="input-field"  
          placeholder="Enter name"  
          onChange={(e) => handleInputChange("name", e.target.value)}  
        />  
        <input  
          type="text"  
          className="input-field"  
          placeholder="Enter pages"  
          onChange={(e) => handleInputChange("pages", e.target.value)}  
        />  
        <input  
          type="number"  
          className="input-field"  
          placeholder="Enter pages per sheet"  
          onChange={(e) => handleInputChange("pagePerSheet", e.target.value)}  
        />  
      </div>  
      <div className="radio-group">  
        <label>Layout:</label>  
        <div className="radio-options">  
          {" "  
        </div>  
        <label>  
          <input  
            type="radio"  
            name="layout"  
            value="portrait"  
            checked={fileType.layout === "portrait"}  
            onChange={(e) => handleInputChange("layout", e.target.value)}  
          />  
          Portrait  
        </label>  
        <label>  
          <input  
            type="radio"  
            name="layout"  
            value="landscape"  
            checked={fileType.layout === "landscape"}  
            onChange={(e) => handleInputChange("layout", e.target.value)}  
          />  
          Landscape  
        </label>  
      </div>  
    </div>  
  </div>  
)
```

```

    />
    Landscape
  </label>
</div>
</div>
<div className="radio-group">
  <label>Color:</label>
  <div className="radio-options">
    <label>
      <input
        type="radio"
        name="color"
        value="black"
        checked={fileType.color === "black"}
        onChange={(e) => handleInputChange("color", e.target.value)}
      />
      Black
    </label>
    <label>
      <input
        type="radio"
        name="color"
        value="color"
        checked={fileType.color === "color"}
        onChange={(e) => handleInputChange("color", e.target.value)}
      />
      Color
    </label>
  </div>
</div>
<div className="checkbox-group">
  <input
    type="checkbox"
    className="checkbox-field"
    checked={fileType.flip}
    onChange={(e) => handleInputChange("flip", e.target.checked)}
  />
  <label>Print on Both sides</label>
</div>
<input type="file" className="file-input" onChange={handleFileChange} />
<button className="upload-button" onClick={handleUpload}>
  Upload File
</button>
<div className="price">Price: {calculatedPrice}</div>
</div>
</div>
);
}

```

```

export default FileUploadApp;
Navbar.jsx
import React, { useState } from "react";
import "./Navbar.css";

function NavBar({setNavbarData}) {

  return (
    <nav className="navbar-container">
      <div className="navbar">
        Photocopy Shop
      </div>
      <div className="home-menu">
        <div onClick={() => setNavbarData("dashboard")}>Dashboard</div>
        <div onClick={() => setNavbarData("revenue")}>Revenue History</div>
        <div onClick={() => setNavbarData("settings")}>Settings</div>
      </div>
    </nav>
  );
}

```

```

export default NavBar;
PriceCalculator.jsx
import React, { useState } from "react";
import "./style.css"; // Import the custom CSS file

```

```

const PriceCalculator = ({ data }) => {
  const [startDate, setStartDate] = useState("");
  const [endDate, setEndDate] = useState("");
  const [totalPrice, setTotalPrice] = useState(0);

  const calculateTotalPrice = () => {
    const start = new Date(startDate);
    const end = new Date(endDate);

    if (isNaN(start.getTime()) || isNaN(end.getTime())) {
      alert("Please select valid start and end dates!");
      return;
    }

    const filteredData = data.filter((item) => {
      const itemDate = new Date(item.uploadTime);
      return itemDate >= start && itemDate <= end && typeof item.price === "number";
    });

    const total = filteredData.reduce((acc, curr) => acc + curr.price, 0);
    setTotalPrice(total);
  };
}

```

```

return (
  <div className="price-calculator-container">
    <h2 className="price-calculator-header">Revenue Calculator</h2>
    <div className="input-group">
      <label>
        Start Date:
        <input
          type="date"
          value={startDate}
          onChange={(e) => setStartDate(e.target.value)}
          className="date-input"
        />
      </label>
    </div>
    <div className="input-group">
      <label>
        End Date:
        <input
          type="date"
          value={endDate}
          onChange={(e) => setEndDate(e.target.value)}
          className="date-input"
        />
      </label>
    </div>
    <button
      onClick={calculateTotalPrice}
      className="calculate-btn"
    >
      Calculate Total Revenue
    </button>
    <div className="total-price">
      <strong>Total Price:</strong> {totalPrice.toFixed(2)}
    </div>
  </div>
);
};

```

```

export default PriceCalculator;
SettingPage.jsx
import React, { useState } from "react";
import QRCode from "react-qr-code";
import axios from "axios";
import "../SettingsPage.css"; // Plain CSS file for styling

```

```

const SettingsPage = ({ ip }) => {
  const [wifiDetails, setWifiDetails] = useState({ ssid: "", password: "" });
  const [webDetails, setWebDetails] = useState({ ip: "", port: "" });
  const [qrCodeData, setQrCodeData] = useState("");

```



```

const [ipqrCodeData, setIpQrCodeData] = useState("");
const [pricing, setPricing] = useState({ blackPrice: 1, colorPrice: 5 });

// Handle input changes
const handleInputChange = (e, setState) => {
  const { name, value } = e.target;
  setState((prev) => ({ ...prev, [name]: value }));
};

// Generate WiFi QR Code
const generateWifiQr = () => {
  const qrData = `WIFI:S:${wifiDetails.ssid};T:WPA;P:${wifiDetails.password};;`;
  setQrCodeData(qrData);
};

// Generate Web Page QR Code
const generateWebQr = () => {
  const qrData = `http://${webDetails.ip}:${webDetails.port}`;
  setIpQrCodeData(qrData);
};

// Update Pricing
const updatePricing = async () => {
  console.log(ip);
  try {
    await axios.post(`http://${ip}:3000/update-pricing`, pricing);
    alert("Pricing updated successfully!");
  } catch (error) {
    console.error("Error updating pricing:", error);
    alert("Failed to update pricing!");
  }
};

return (
  <div className="settings-container">
    <h1>Photocopy Shop Settings</h1>

    {/* QR Code Section */}
    <div className="section">
      <h2>QR Code Generation</h2>
      <div className="form-group">
        <h3>WiFi QR Code</h3>
        <input
          type="text"
          name="ssid"
          placeholder="WiFi SSID"
          value={wifiDetails.ssid}
          onChange={(e) => handleInputChange(e, setWifiDetails)}
        />
        <input

```

```

        type="password"
        name="password"
        placeholder="WiFi Password"
        value={wifiDetails.password}
        onChange={(e) => handleInputChange(e, setWifiDetails)}
      />
      <button onClick={generateWifiQr}>Generate WiFi QR Code</button>
    </div>
    {qrCodeData && (
      <div className="qr-code">
        <h3>Generated QR Code:</h3>
        <QRCode value={qrCodeData} size={150} />
      </div>
    )}

    <div className="form-group">
      <h3>Web Page QR Code</h3>
      <input
        type="text"
        name="ip"
        placeholder="IP Address"
        value={webDetails.ip}
        onChange={(e) => handleInputChange(e, setWebDetails)}
      />
      <input
        type="text"
        name="port"
        placeholder="Port"
        value={webDetails.port}
        onChange={(e) => handleInputChange(e, setWebDetails)}
      />
      <button onClick={generateWebQr}>Generate Web QR Code</button>
    </div>
    {ipqrCodeData && (
      <div className="qr-code">
        <h3>Generated QR Code:</h3>
        <QRCode value={ipqrCodeData} size={150} />
      </div>
    )}
  </div>

  { /* Pricing Section */ }
  <div className="section">
    <h2>Pricing</h2>
    <div className="form-group">
      <label>Black & White Price (₹):</label>
      <input
        type="number"
        name="blackPrice"

```

```

        value={pricing.black}
        onChange={(e) => handleInputChange(e, setPricing)}
      />
    </div>
    <div className="form-group">
      <label>Color Price (₹):</label>
      <input
        type="number"
        name="colorPrice"
        value={pricing.color}
        onChange={(e) => handleInputChange(e, setPricing)}
      />
    </div>
    <button onClick={updatePricing}>Update Pricing</button>
  </div>
</div>
);
};

```

```
export default SettingsPage;
```

Server.jsx

```

const express = require("express");
const http = require("http");
const socketIo = require("socket.io");
const fs = require("fs");
const path = require("path");
const QRCode = require("qrcode");
const app = express();
const cors = require("cors");
const cron = require("node-cron");
const { PDFDocument } = require('pdf-lib');
const db = require("./db");
const server = http.createServer(app);
const ip = "0.0.0.0";
const metadataFilePath = path.join(__dirname, "file_metadata.json");
const priceDataFilePath = path.join(__dirname, "file_price_data.json");
const shopDataFilePath = path.join(__dirname, "shop_data.json");

```

```

// Allow CORS for your frontend (localhost:5173)
app.use(cors());
app.use(express.json());
app.use('/uploads', express.static(path.join(__dirname, 'uploads')));
app.get("/", (req, res) => {
  res.json({ message: "Hello from the server!" });
});

```

```

app.get("/pricing", (req, res) => {
  try {

```

```

    const data = fs.readFileSync(shopDataFilePath, "utf-8");
    const pricing = JSON.parse(data);
    res.status(200).json(pricing);
  } catch (error) {
    console.error("Error reading shop data:", error);
    res.status(500).json({ error: "Failed to read pricing data" });
  }
});

app.post("/update-pricing", (req, res) => {
  let { blackPrice, colorPrice } = req.body;
  console.log(req.body);
  blackPrice = parseInt(blackPrice);
  colorPrice = parseInt(colorPrice);
  if (typeof blackPrice !== "number" || typeof colorPrice !== "number") {
    return res.status(400).json({ error: "Invalid data format" });
  }
  console.log(blackPrice, colorPrice);
  try {
    // Read the current data
    const data = fs.readFileSync(shopDataFilePath, "utf-8");
    const shopData = JSON.parse(data);

    // Update the pricing
    shopData.blackPrice = blackPrice;
    shopData.colorPrice = colorPrice;

    // Write the updated data back to the file
    console.log(shopData);
    fs.writeFileSync(
      shopDataFilePath,
      JSON.stringify(shopData, null, 2),
      "utf-8"
    );

    res.status(200).json({ message: "Pricing updated successfully" });
  } catch (error) {
    console.error("Error updating shop data:", error);
    res.status(500).json({ error: "Failed to update pricing data" });
  }
});

const io = socketIo(server, {
  cors: {
    origin: "*", // Replace with your frontend's URL
    methods: ["GET", "POST"],
    allowedHeaders: ["Content-Type"],
  },
});

const port = 3000;

```

```

const uploadDir = path.join(__dirname, "uploads");

// Ensure the upload directory exists
if (!fs.existsSync(uploadDir)) {
  fs.mkdirSync(uploadDir);
}

// File metadata storage paths

// Utility function to read file metadata
function readMetadata() {
  if (fs.existsSync(metadataFilePath)) {
    return JSON.parse(fs.readFileSync(metadataFilePath, "utf-8"));
  }
  return [];
}

// Utility function to write file metadata
function writeMetadata(data) {
  fs.writeFileSync(metadataFilePath, JSON.stringify(data, null, 2), "utf-8");
}

// Utility function to read price data
function readPriceData() {
  if (fs.existsSync(priceDataFilePath)) {
    return JSON.parse(fs.readFileSync(priceDataFilePath, "utf-8"));
  }
  return [];
}

// Utility function to write price data
function writePriceData(priceData) {
  const existingData = readPriceData();
  existingData.push(priceData);
  fs.writeFileSync(
    priceDataFilePath,
    JSON.stringify(existingData, null, 2),
    "utf-8"
  );
}

// Utility function to generate a unique file name if a file with the same name already exists
function getUniqueFilePath(fileName) {
  const extname = path.extname(fileName);
  const basename = path.basename(fileName, extname);
  let uniqueFilePath = path.join(uploadDir, fileName);
  let counter = 1;

  while (fs.existsSync(uniqueFilePath)) {

```

```

    uniqueFilePath = path.join(uploadDir, `${basename}_${counter}${extname}`);
    counter++;
  }

  return uniqueFilePath;
}

// WebSocket handling
io.on("connection", (socket) => {
  console.log("Client connected");
  io.emit("fileList", readMetadata());
  io.emit("dataList", readPriceData());
  socket.on("uploadFile", (data) => {
    const { fileName, fileData, fileType, userInfo } = data;

    if (!fileName || !fileData || !fileType) {
      socket.emit("uploadStatus", {
        success: false,
        message: "Invalid file data!",
      });
      return;
    }

    const uniqueFileName = getUniqueFilePath(fileName);

    // Save the file
    fs.writeFile(uniqueFileName, Buffer.from(fileData), (err) => {
      if (err) {
        console.error("Error saving file:", err);
        socket.emit("uploadStatus", {
          success: false,
          message: "File upload failed!",
        });
        return;
      }

      console.log("File uploaded successfully:", fileName);

      // Update and save metadata
      const metadata = readMetadata();
      const fileMetadata = {
        fileName,
        uniqueFileName,
        fileType,
        userInfo,
        filePath: uniqueFileName,
        uploadTime: new Date(),
      };
      metadata.push(fileMetadata);
    });
  });
});

```

```

writeMetadata(metadata);

// Save price data
const priceData = {
  price: fileType.price || 1,
  uploadTime: new Date(),
};
writePriceData(priceData);
(async () => {
  try {
    await db.query(
      `INSERT INTO uploaded_files (
        file_name, unique_file_name, file_path, upload_time,
        file_type_name, file_type_pages, file_type_page_per_sheet,
        file_type_layout, file_type_color, file_type_price, file_type_flip,
        user_ip_address
      ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)`,
      [
        fileMetadata.fileName,
        fileMetadata.uniqueFileName,
        fileMetadata.filePath,
        new Date(fileMetadata.uploadTime), // ensure it's a valid Date
        fileType.name || null,
        fileType.pages || null,
        fileType.pagePerSheet || null,
        fileType.layout || null,
        fileType.color || null,
        fileType.price || 0,
        fileType.flip ? 1 : 0,
        userInfo?.ipAddress || null
      ]
    );
    console.log("Inserted file metadata into MySQL successfully");
  } catch (err) {
    console.error("Error inserting file metadata into MySQL:", err);
  }
})();
socket.emit("uploadStatus", {
  success: true,
  message: `File uploaded successfully: ${fileName}`,
});
io.emit("fileList", metadata);
io.emit("dataList", readPriceData());

// Schedule file deletion after 30 seconds

});
});
cron.schedule("*/30 * * * *", () => {

```

```

console.log("Running cleanup task...");

const metadata = readMetadata();
const updatedMetadata = [];

metadata.forEach((file) => {
  const fileAge = (Date.now() - new Date(file.uploadTime).getTime()) / 1000;

  if (fileAge > 30) {
    try {
      fs.unlinkSync(file.uniqueFileName);
      console.log(`Deleted file: ${file.uniqueFileName}`);
    } catch (err) {
      console.error(`Error deleting file ${file.uniqueFileName}:`, err);
    }
  } else {
    updatedMetadata.push(file);
  }
});

// Save updated metadata (only files not deleted)
writeMetadata(updatedMetadata);
io.emit("fileList", updatedMetadata);
});

socket.on("disconnect", () => {
  console.log("Client disconnected");
});

// Start the server
server.listen(port, ip, () => {
  console.log(`Server is running at http://localhost:${port}`);
});

```



# APPENDIX II

## SCREENSHOTS

File Upload

Enter name

Enter pages

Enter pages per sheet

Layout:

☒ Portrait

☐ Landscape

Color:

☒ Black

☐ Color

☐ Print on Both sides

Choose File

No file chosen

Upload File

Price: 1

### A.2.1 User module

PHOTOCOPY SHOP

Dashboard

Revenue History

Settings

Revenue Calculator

Start Date:

14-05-2025

End Date:

01-03-2025

March, 2025

Su

Mo

Tu

We

Th

Fr

Sa

23

24

25

26

27

28

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

1

2

3

4

5

Clear

Today

Total Revenue

Price: 0.00

### A.2.2 Revenue Management

PHOTOCOPY SHOP

DashboardRevenue HistorySettings

List of Files to be Printed

Name	File Name	Pages	Layout	Color	Pages Per Sheet	Price	Uploaded Time	Print
1	ml-screen.pdf	1-5	portrait	black	1	0	5/5/2025, 4:02:28 pm	<button>Print</button>


Total Revenue: 777.50

View Revenue at range

### A.2.3 File management

Generate WiFi QR Code

Generated QR Code:



Web Page QR Code

Generate Web QR Code

Pricing

Black & White Price (₹):

Color Price (₹):

Update Pricing

### A.2.4 Settings Page

## REFERENCES

- [1] **Singh, J., & Others. (2023).** Comprehensive Review on Web-based Photocopy Shop File Sharing System. Published a detailed review on file-sharing systems for photocopy shops, covering modules like file uploads, transaction management, and real-time transfers using web technologies like React.js and Node.js. International Journal of Engineering Research & Technology (IJERT), 12(11).
- [2] **Gray, W. S., & Liguori, S. C. (2003).** Photocopy Shop Management and Operations. Offered a complete guide on traditional photocopy shop management practices, customer service, and financial controls, now adaptable to modern web-based systems. Prentice Hall PTR (4th ed.).