



DOCUMENT VERIFICATION SYSTEM USING QR CODES

A PROJECT REPORT

Submitted in partial fulfilment of the requirements for the degree of

BACHELORS OF COMPUTER APPLICATION

(6th Semester)

Submitted By:

Abhinay Singh Sengar (GU22R3475)

Submitted To:

Ms. Anjali Mam

Department of CS & IT

Avviare Educational Hub

Noida, UP – 201301, India

ABSTRACT

The Document Verification System using QR Codes is designed to provide a secure, efficient, and paperless method for validating documents digitally. This project focuses on developing a web-based application that allows users to upload documents, which are then stored in a secure database and linked to a dynamically generated QR code. This QR code serves as a unique identifier and access point for document verification, ensuring authenticity and easy retrieval.

The system is implemented using modern web technologies, including Node.js and Express.js for the backend, MongoDB for cloud-based data storage, and Multer for handling document uploads. The QR codes are generated using the 'qrcode' npm package and are accessible through a clean, user-friendly frontend built with HTML, CSS, and JavaScript. When scanned, the QR code allows instant access to the stored document's metadata, enabling quick and reliable verification.

This solution is particularly useful for institutions, colleges, and organizations that need to handle and verify a large volume of documents. It reduces the risk of forgery, minimizes manual verification efforts, and ensures long-term digital traceability. Future enhancements could include the integration of blockchain technology for tamper-proof verification, OTP-based access control, and mobile application support for real-time QR scanning and document validation.

ACKNOWLEDGEMENT

I am highly grateful to **Ms. Kanika Singh**, Director of Operations, Avviare Educational Hub, Noida, for providing this opportunity to carry out the major project work. The constant guidance and encouragement received from **Mr. Ashutosh Rathore**, H.O.D., IT Department, Avviare Educational Hub, Noida, has been of great help in carrying out the project work and is acknowledged with reverential thanks.

I would like to express my sincere gratitude to everyone who supported and contributed to the successful completion of this **Document Verification System using QR Codes** project.

I am also grateful to the open-source developer community, whose contributions through tools, libraries, and frameworks such as **Node.js, Express.js, MongoDB, Multer, and the QRCode npm library** made it possible to implement and test this system efficiently. I sincerely acknowledge the creators and maintainers of these technologies for making modern development tools accessible to learners and developers.

I would like to express a deep sense of gratitude and thanks profusely to **Ms. Nisha Choudhary**, without whose wise counsel and able guidance it would have been impossible to complete the project in this manner. I also extend my gratitude to other faculty members of the IT Department at Avviare Educational Hub, Noida, for their academic support and insights throughout the course of this work.

Finally, I am indebted to all who have directly or indirectly contributed to the successful completion of this report.

Name: Abhinay Singh Sengar

Date: 29/05/2025

LIST OF FIGURES

[illegible]

TABLE OF CONTENTS

Content	Page No.
Introduction to Project	1
Technical Requirements (Hardware and Software Used)	2
Objectives	5
Introduction to Languages, IDE's	8
Tools and Technologies used for Implementation	10
Flowcharts/Diagram of the Program	21
Snapshots of Program and Output	23
Conclusion	25
Future Scope	26
References and Bibliography	28

Format for Major Project Report

Title page	Page
<i>Abstract</i>	<i>i</i>
<i>Acknowledgement</i>	<i>ii</i>
<i>List of Figures</i>	<i>iii</i>
<i>List of Tables</i>	<i>iv</i>
<i>Table of Contents</i>	<i>v</i>

Chapter 1 Introduction

- 1.1 Introduction to Project
- 1.2 Technical Requirements (Hardware and Software Used)
- 1.3 Objectives

Chapter 2. Implementation

- 2.1 Introduction to Languages, IDE's, Tools and Technologies used for Implementation

Chapter 3. Results, Flowcharts and Outputs

- 3.1 Flowcharts/Diagram of the Program
- 3.2 Snapshots of Program and Output

Chapter 4. Conclusion and Future Scope

References and Bibliography

CHAPTER: 1 INTRODUCTION

A **Document Verification System using QR Codes** is a secure, digital solution designed to authenticate documents by associating each uploaded file with a unique QR code. This project aims to simplify and automate the verification process, replacing traditional paper-based or manually validated systems with a reliable and efficient web-based application.

By leveraging technologies such as **Node.js**, **Express.js**, **MongoDB**, and **QR Code generation libraries**, the system allows users to upload documents, which are stored securely and linked to QR codes. These QR codes can be scanned anytime to validate the authenticity and retrieve details of the associated document.

Key Features

✔ Secure Uploads:

Allows users to upload documents (PDF, PNG, JPG) that are safely stored in a cloud database (MongoDB Atlas).

✔ QR Code Generation:

Automatically generates a unique QR code for each uploaded document.

✔ Quick Verification:

Scanning the QR code instantly fetches document details for verification.

✔ Admin Access:

Admins can view all documents and manage user activities securely.

✔ Paperless Process:

Reduces dependency on physical copies and manual stamping or signatures.

How It Works

□ Input Processing:

User uploads a document via a web form.

□ QR Code Generation:

The backend (Node.js + Express.js) processes the document and generates a unique QR code.

□ Data Storage:

Document and QR code data are stored in **MongoDB Atlas**.

□ Verification:

When a QR code is scanned, the system retrieves the linked document metadata for verification.

Applications

- **Educational Institutions:** Verify marksheets, certificates, and student records.
- **Corporate Sector:** Verify employment letters, contracts, and onboarding documents.
- **Government Services:** Validate licenses, identity proofs, and legal documents.
- **Healthcare:** Authenticate medical reports or prescriptions.
- **E-commerce & Logistics:** Secure invoices and delivery confirmations.

Technical Requirements (Hardware and Software Used)

Hardware Requirements

Development Environment (Local Machine)

- **Processor:**
 - Minimum: Intel Core i3 (or equivalent)
 - Recommended: Intel Core i5/i7 or AMD Ryzen 5/7
- **RAM:**
 - Minimum: 4 GB
 - Recommended: 8 GB or more
- **Storage:**
 - Minimum: 50 GB free space
 - Recommended: SSD for faster performance
- **Operating System:**
 - Windows 10/11, macOS, or Ubuntu 20.04+
- **Network:**
 - Stable internet connection for package installation and testing

Production Environment (Server Requirements)

- **Processor:**
 - Quad-core CPU or higher
- **RAM:**
 - Minimum: 8 GB
 - Recommended: 16 GB
- **Storage:**
 - Minimum: 100 GB
 - Cloud storage recommended (e.g., AWS S3)
- **Hosting Provider:**
 - Options include Heroku, Render, AWS, or Vercel
- **Network Bandwidth:**
 - 10 Mbps or more for real-time access and file handling

Software Requirements

Development Environment

1. Programming Environment

Programming Language:

- **JavaScript (ES6+)** for frontend logic
- **Node.js (v18 or later)** for backend development
- **Express.js** as the web application framework

2. Libraries and Frameworks

These libraries provide core functionality for backend operations, document handling, and QR code generation:

- **Express.js** – For setting up the server and managing routes
- **Multer** – Middleware for handling file uploads
- **QRCode** – Node package to generate QR codes for each document
- **Mongoose** – ODM for MongoDB to define schemas and interact with the database

- **dotenv** – For managing environment variables securely

3. Integrated Development Environment (IDE)

You can use any modern IDE to develop and debug your code:

- **Visual Studio Code** (Recommended)
- **Sublime Text**
- **WebStorm**

4. Database

MongoDB Atlas (Cloud-based MongoDB):

- Used for storing user details, uploaded document metadata, and QR code information.

5. Folder Structure

Typical folder structure for the Node.js project:

```
sql
CopyEdit
/document-verification-system
|-- /uploads          → Stores uploaded documents
|-- /routes           → Express routes
|-- /controllers      → Logic for upload, scan, verify
|-- /models           → Mongoose schema files
|-- .env              → Environment variables
|-- server.js         → Entry point of the application
```

6. Deployment Tools

- **Postman** – To test REST APIs
- **Render / Heroku / Vercel** – For free hosting and deployment of Node.js apps
- **Git & GitHub** – Version control and project backup

System Requirements

Operating System:

- **Windows 10/11, macOS, or Linux (Ubuntu 20.04 or later)**

Hardware Requirements:

- **Processor:** Intel i5 or above (multi-core recommended)
- **RAM:** Minimum 4 GB (Recommended 8 GB for smooth performance)
- **Storage:** At least 2 GB free space for Node modules, uploads, and logs

Software Dependencies

- **Node.js & npm (Node Package Manager)**
 - Required to install and manage all backend dependencies
- **MongoDB Atlas Account**
 - Used to set up a cloud database
- **Web Browser:**

- Google Chrome or Firefox for testing and user interface access

Network Connectivity

Required for:

- Installing npm libraries (express, multer, qrcode, mongoose, etc.)
- Pushing code to GitHub
- Accessing MongoDB Atlas
- Running and testing frontend + backend integration

OBJECTIVE

Primary Objectives

Secure Document Upload:

Develop a system that allows users to upload official documents in a secure and organized manner.

QR Code Generation:

Automatically generate a unique QR code for each uploaded document, ensuring fast and reliable identification.

Document Verification via QR Code:

Enable the system to scan QR codes and instantly retrieve and verify associated document information from the database.

Cloud-Based Storage:

Use MongoDB Atlas to store documents and metadata, ensuring accessibility and reliability.

Web-Based Interface:

Provide a user-friendly web interface for uploading, scanning, and verifying documents through QR codes.

Secondary Objectives

User Authentication:

Implement user registration and login functionalities to ensure that only authorized users can upload or access documents.

Admin Dashboard:

Provide administrative access to manage users and view all uploaded documents.

Data Integrity:

Ensure that uploaded files are tamper-proof by restricting direct access and linking them only through secure QR codes.

Efficient Document Retrieval:

Enable quick and accurate retrieval of document details using the scanned QR code.

Error Handling:

Display meaningful error messages for unsupported file formats, upload failures, or QR code mismatches.

Development Objectives

Modular Code Structure:

Organize backend and frontend logic in reusable, scalable modules for easier maintenance and future expansion.

Use of Modern Tools:

Utilize Express.js, Multipart, MongoDB, and QRCode libraries for efficient and reliable system development.

Environment Configuration:

Use environment variables (.env file) for secure and flexible configuration of database URLs, ports, and API keys.

RESTful API Design:

Develop clean and scalable APIs to handle document upload, QR code generation, and verification workflows.

Testing and Debugging:

Use Postman and browser dev tools to test each API and system component to ensure reliability and accuracy.

Future Objectives**Mobile App Integration:**

Develop a mobile application for Android/iOS to allow QR code scanning and document verification on the go.

Blockchain Integration:

Incorporate blockchain to make document storage and verification immutable and tamper-proof.

Role-Based Access Control (RBAC):

Add multi-role access for users, admins, and verifiers to enhance security and manage privileges.

OTP-Based Verification:

Add an extra layer of security using OTP (One-Time Password) for verifying sensitive document access.

Offline Verification Feature:

Enable QR code verification without internet by caching document hash values locally or embedding encrypted metadata in the QR.

CHAPTER 2: IMPLEMENTATION

Introduction to Languages, IDEs, Tools, and Technologies Used for Implementation

The development of the **Document Verification System using QR Codes** utilizes modern web development tools and technologies such as **JavaScript**, **Node.js**, **Express.js**, **MongoDB**, and the **QRCode npm library**. Below is an overview of each component used during implementation:

1. Environment Setup

Install Required Software

- **Install Node.js** (v18 or later recommended)
- **Install MongoDB Atlas** account for cloud database
- **Install Visual Studio Code** or any modern IDE

Initialize Node.js Project

```
bash
CopyEdit
npm init -y
```

Install Required Dependencies

```
bash
CopyEdit
npm install express multer mongoose qrcode dotenv
```

Setup File Structure

```
bash
CopyEdit
project-folder/
├── /routes           → Express routes
├── /controllers      → Upload and verification logic
├── /models           → MongoDB schemas
├── /uploads          → Directory for storing uploaded files
├── .env              → Environment variables (e.g., DB_URI, PORT)
└── server.js         → Entry point of the application
```

2. Document Upload Module

Using **Multer** (a Node.js middleware), the system accepts and stores uploaded documents in the `/uploads` folder. It also saves document metadata (filename, path, upload date) to MongoDB.

Supported formats: `.pdf`, `.jpg`, `.png`

```
js
CopyEdit
const multer = require("multer");
const storage = multer.diskStorage({
  destination: "uploads/",
  filename: (req, file, cb) => cb(null, Date.now() + "-" + file.originalname)
});
const upload = multer({ storage });
```

3. QR Code Generation

The system uses the **qrcode** npm package to generate a unique QR code for each uploaded document. The QR code stores a URL or document ID used for verification.

```
js
CopyEdit
const QRCode = require('qrcode');
QRCode.toFile('qr-code.png', 'http://server.com/verify/123456');
```

Each QR code is saved and associated with its corresponding document in the database.

4. Database Integration (MongoDB Atlas)

All documents and QR metadata are stored using **MongoDB** with the help of **Mongoose** ODM.

Sample Schema:

```
js
CopyEdit
const DocumentSchema = new mongoose.Schema({
  filename: String,
  path: String,
  uploadDate: Date,
  qrCodePath: String
});
```

5. Verification Module

When a QR code is scanned, it leads to a route like `/verify/:id`, where the server retrieves the document details from MongoDB and shows whether the document is valid.

Sample Route:

```
js
CopyEdit
app.get('/verify/:id', async (req, res) => {
  const doc = await Document.findById(req.params.id);
  if (doc) res.send(doc);
  else res.status(404).send("Document not found");
});
```

6. Testing

✔ Functional Testing:

- File upload functionality
- QR code generation
- Data saved in MongoDB correctly
- QR scanning and document retrieval

✔ Integration Testing:

- Tested all modules together for seamless upload → QR → verify flow

✔ User Testing:

- Tested with dummy PDF and image files by students and faculty for validation

7. Deployment

Local Testing:

- Run the app using:

```
bash
CopyEdit
node server.js
```

Web Deployment (Optional):

- Host using **Render**, **Heroku**, or **Vercel**

Frontend Access:

- HTML/CSS/JS forms for file upload and document view

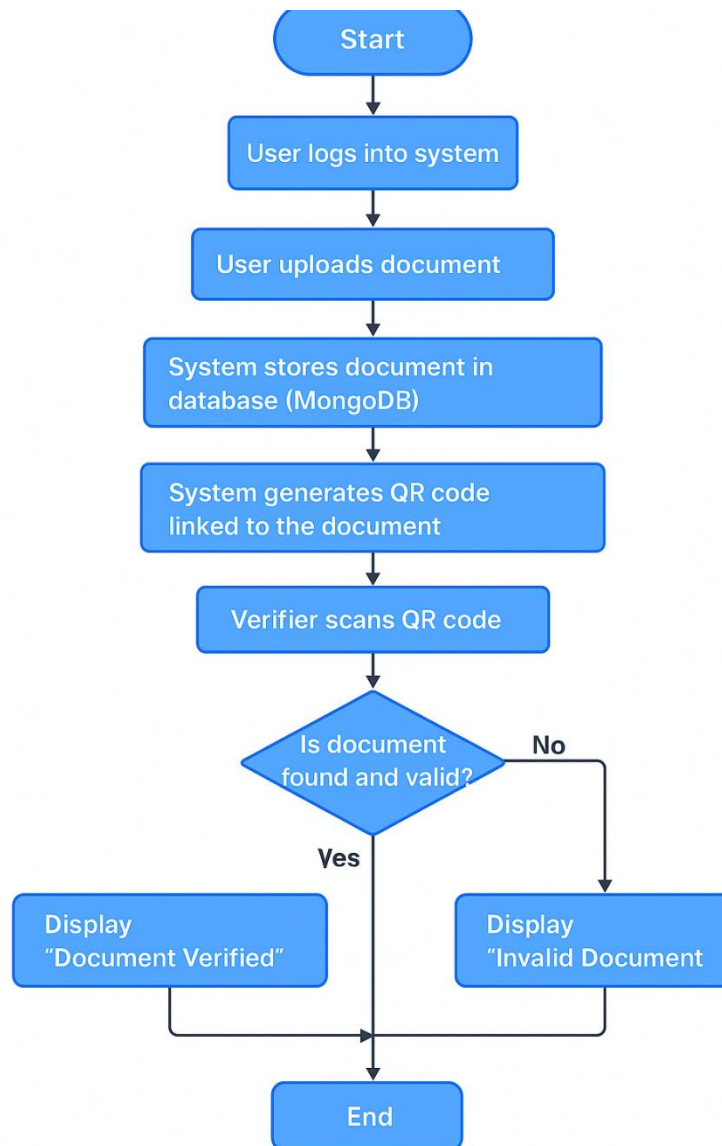
8. Enhancements (Optional Features)

- **OTP-Based Access:** Add verification via one-time password before document access
- **Blockchain Integration:** Use blockchain to prevent tampering of verification logs
- **Role-Based Access:** Add user roles like admin, verifier, and guest
- **Mobile App:** Integrate document scan and verification on mobile
- **Downloadable Reports:** Allow QR-linked downloadable document receipts

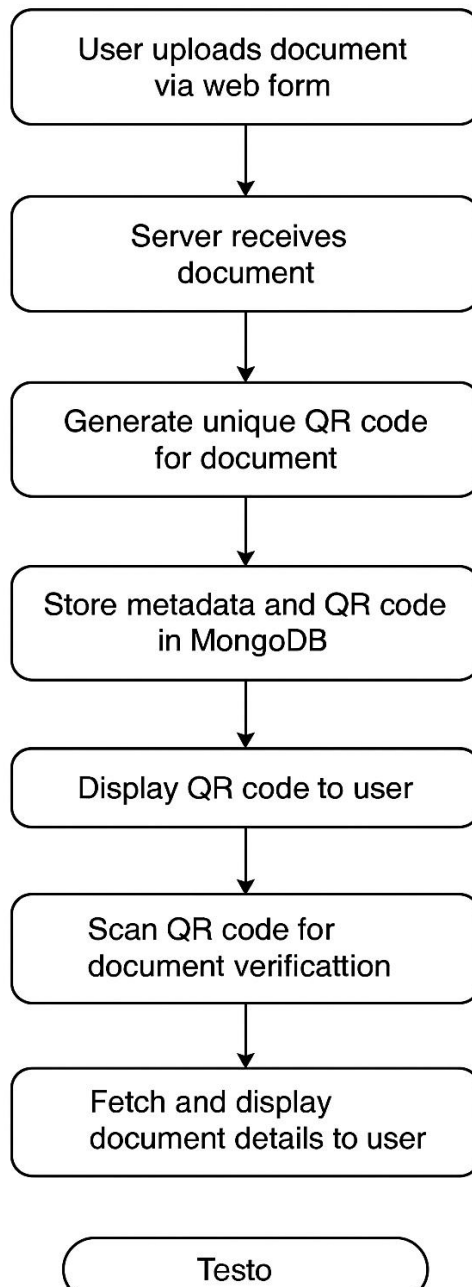
Chapter 3:

Results, Flowcharts and Outputs

Flowcharts of the project:

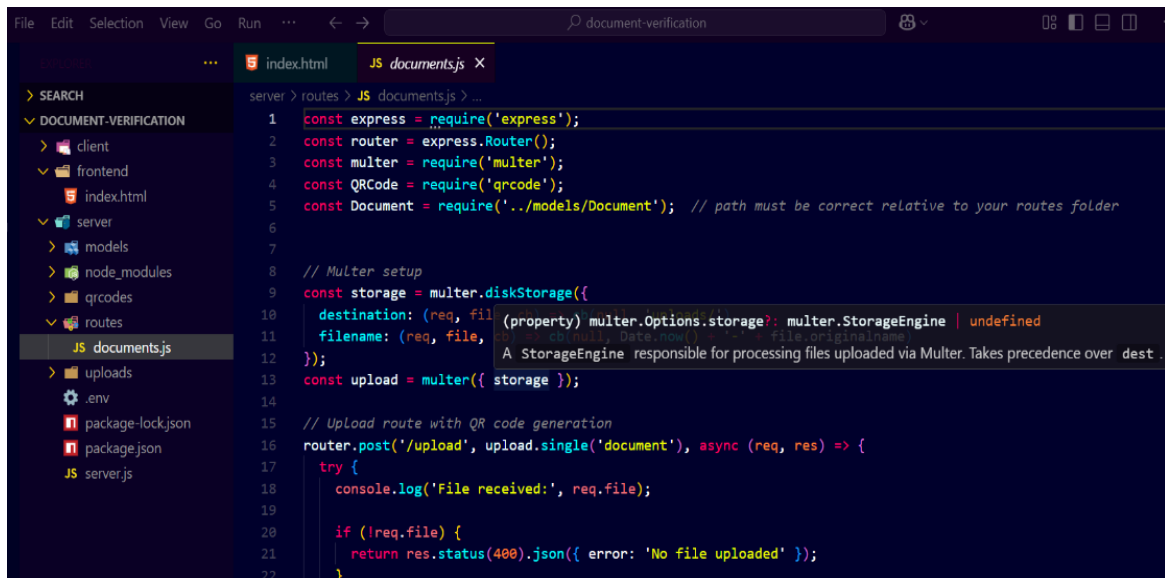


DFD (Data Flow Diagram) of the ChatBot:



RESULT:

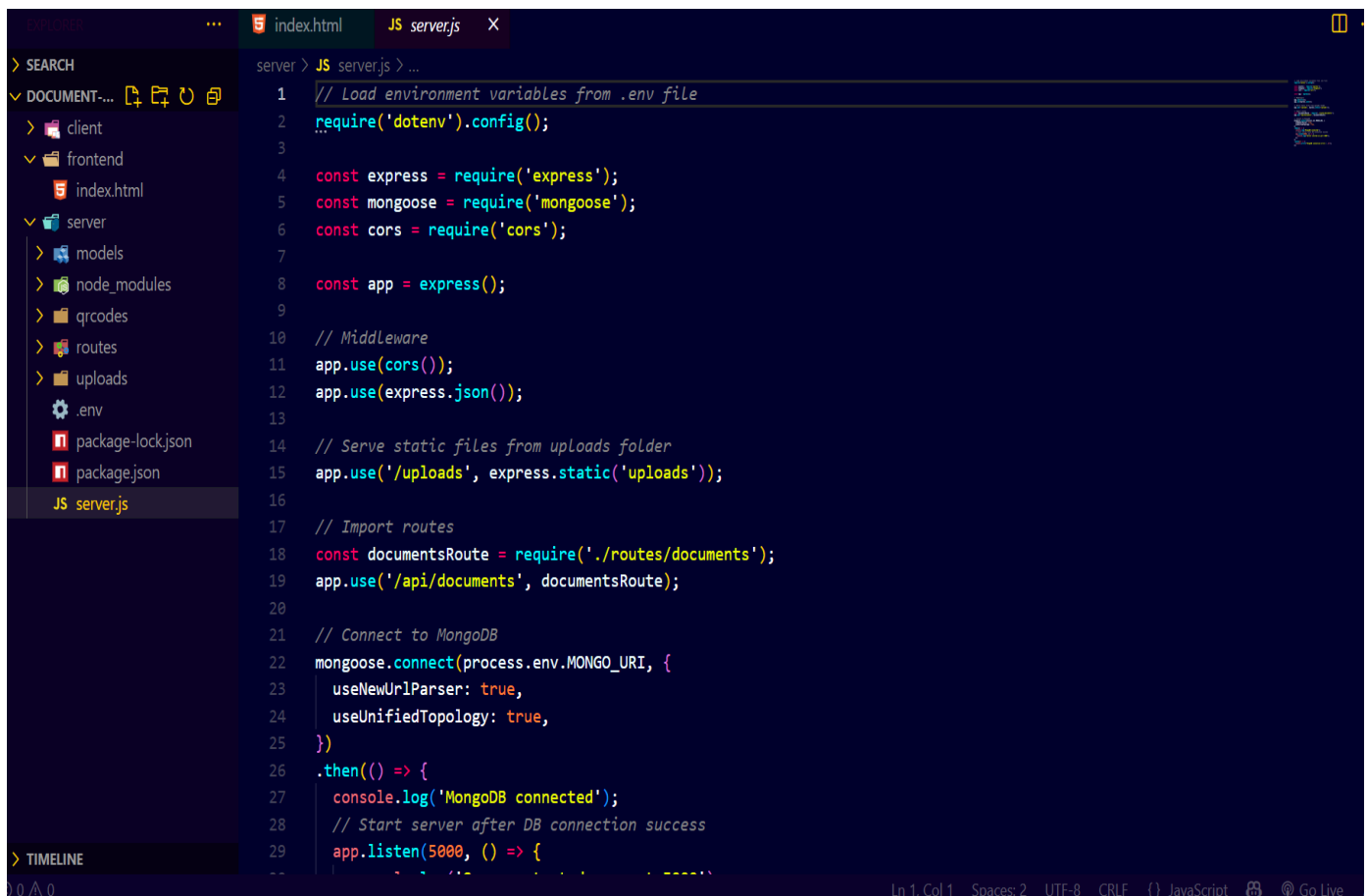
- **All the Requirements of the Document Verification System:**



The screenshot shows a VS Code editor window with the file explorer on the left and the code editor on the right. The file explorer shows a project structure with folders like 'client', 'frontend', 'server', 'models', 'node_modules', 'qrCodes', 'routes', and 'uploads'. The 'server' folder is expanded, and 'JS documents.js' is selected. The code editor shows the following JavaScript code:

```
1 const express = require('express');
2 const router = express.Router();
3 const multer = require('multer');
4 const QRCode = require('qrcode');
5 const Document = require('../models/Document'); // path must be correct relative to your routes folder
6
7
8 // Multer setup
9 const storage = multer.diskStorage({
10   destination: (req, file, cb) => {
11     (property) => {
12       (property) ? multer.Options.storage : multer.StorageEngine | undefined
13       cb(null, Date.now() + '-' + file.originalname);
14     }
15   },
16   filename: (req, file, cb) => {
17     cb(null, file.originalname);
18   }
19 });
20 const upload = multer({ storage });
21
22 // Upload route with QR code generation
23 router.post('/upload', upload.single('document'), async (req, res) => {
24   try {
25     console.log('File received:', req.file);
26
27     if (!req.file) {
28       return res.status(400).json({ error: 'No file uploaded' });
29     }
30   } catch (err) {
31     console.error(err);
32     return res.status(500).json({ error: 'Server error' });
33   }
34 });
```

1. The necessary libraries, modules, and main JavaScript code (server.js):



The screenshot shows a VS Code editor window with the file explorer on the left and the code editor on the right. The file explorer shows a project structure with folders like 'client', 'frontend', 'server', 'models', 'node_modules', 'qrCodes', 'routes', and 'uploads'. The 'server' folder is expanded, and 'JS server.js' is selected. The code editor shows the following JavaScript code:

```
1 // Load environment variables from .env file
2 require('dotenv').config();
3
4 const express = require('express');
5 const mongoose = require('mongoose');
6 const cors = require('cors');
7
8 const app = express();
9
10 // Middleware
11 app.use(cors());
12 app.use(express.json());
13
14 // Serve static files from uploads folder
15 app.use('/uploads', express.static('uploads'));
16
17 // Import routes
18 const documentsRoute = require('./routes/documents');
19 app.use('/api/documents', documentsRoute);
20
21 // Connect to MongoDB
22 mongoose.connect(process.env.MONGO_URI, {
23   useNewUrlParser: true,
24   useUnifiedTopology: true,
25 })
26 .then(() => {
27   console.log('MongoDB connected');
28   // Start server after DB connection success
29   app.listen(5000, () => {
30     console.log('Server is running on port 5000');
31   });
32 });
```

2. contains predefined intents, patterns, and responses.

```
SEARCH
DOCUMENT-...
client
frontend
  index.html
server

frontend > index.html > html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8" />
5   <title>Document Upload with QR Code</title>
6 </head>
7 <body>
8   <h1>Upload Document</h1>
9
10  <input type="file" id="fileInput" required />
11  <button id="uploadBtn">Upload</button>
12
13  <h2>QR Code:</h2>
14  <div id="qrCodeContainer"></div>
15
16  <script>
17    const fileInput = document.getElementById('fileInput');
18    const uploadBtn = document.getElementById('uploadBtn');
19    const qrCodeContainer = document.getElementById('qrCodeContainer');
20
21    uploadBtn.addEventListener('click', async () => {
22      const file = fileInput.files[0];
23      if (!file) {
24        alert('Please select a file');
25        return;
26      }
27
28      const formData = new FormData();
29      formData.append('document', file);
```

Output:

Upload Document

Choose File aditya[1].pdf

Upload

QR Code:

Network error

CHAPTER 4:

Conclusion and Future Scope

Conclusion

The Document Verification System using QR Codes aims to streamline and secure the process of document validation by utilizing quick response (QR) technology. This project demonstrates how integrating Node.js, MongoDB, and frontend technologies can result in a system capable of generating, scanning, and verifying documents in real-time.

Through this project, we have built a system where a unique QR code is generated and embedded onto each document. Upon scanning, the QR code retrieves the document data from the database and verifies its authenticity. This reduces the risks of document forgery and enhances the credibility of issued certificates or records.

The system offers a user-friendly interface, fast verification, and centralized data management. It can be applied in educational institutions, government offices, and private sectors for verifying mark sheets, ID proofs, licenses, and more. Our solution is scalable, efficient, and can be extended with additional features in the future.

Future Scope

1. **Blockchain Integration:** In the future, the verification process can be made tamper-proof by integrating blockchain technology to record document hashes securely.
2. **Cloud Storage Integration:** Implementing cloud platforms like AWS or Google Cloud for storage can make the system more scalable and accessible globally.
3. **Mobile Application:** A cross-platform mobile app can be developed to scan and verify QR codes directly through smartphones, improving accessibility.
4. **Role-Based Access Control (RBAC):** Future versions of the system can include multi-level access (admin, verifier, user) for better security and data segregation.
5. **Email/SMS Notification System:** Upon successful verification or issuance of a document, automated alerts can be sent to the concerned users for confirmation.
6. **AI-based Fraud Detection:** Incorporating AI/ML algorithms to detect duplicate or forged documents based on behavioral or data anomalies.
7. **Multiple File Format Support:** Supporting various document types like PDF, DOCX, PNG, etc., will increase the system's usability.
8. **Multi-Language Support:** Including support for regional and international languages can make the platform more inclusive and user-friendly.
9. **Real-Time Analytics Dashboard:** Adding a dashboard to display live statistics about document generation, verification frequency, and user activity would help in monitoring system performance.

References and Bibliography

1. Books and Texts

- **"Node.js Design Patterns"** by Mario Casciaro and Luciano Mammino — For understanding Node.js architecture and best practices.
- **"MongoDB: The Definitive Guide"** by Kristina Chodorow — To learn MongoDB database concepts and usage.
- **"Web Development with Node and Express"** by Ethan Brown — To learn building server-side applications using Node.js and Express.
- **"QR Code Essentials"** by multiple online tutorials — For understanding QR code generation and scanning techniques.

2. Websites and Documentation

- **Node.js Official Documentation**
<https://nodejs.org/en/docs/>
- **MongoDB Official Documentation**
<https://docs.mongodb.com/>
- **Mongoose (MongoDB ODM) Documentation**
<https://mongoosejs.com/docs/>
- **QR Code Generator Libraries (like 'qrcode' npm package)**
<https://www.npmjs.com/package/qrcode>

3. Online Tutorials and Resources

- Traversy Media — “Build a RESTful API with Node.js and Express” (YouTube tutorial)
- FreeCodeCamp — “How to Create and Use QR Codes in Node.js” (Article and video)
- MDN Web Docs — For HTML, CSS, and JavaScript references:
<https://developer.mozilla.org/>

4. Tools and Libraries

- **Express.js** (<https://expressjs.com/>) — Web framework for Node.js
- **Multer** (<https://github.com/expressjs/multer>) — Middleware for handling file uploads
- **qrcode npm package** — For QR code generation

5. Additional Resources

- **Google Cloud Vision API** (optional for advanced image/QR scanning)
- **Stack Overflow and GitHub repositories** related to document verification and QR code projects