**By: Kamalesh V B**

**Kamaleshbalu8838@gmail.com**

**Lovely Professional university**

**Full Explanation of Cathago Document Scanner Project**

**Project Overview**

The Cathago Document Scanner is a web-based application designed to allow users to upload text documents, compare them against a reference set for similarity, and view detailed analytics. It features a modern, dark-themed interface with 3D particle effects, built using Node.js, Express, SQLite, Python, and Three.js. The project includes distinct user and admin dashboards, offering functionalities like document matching, credit management, and smart analytics.

**Purpose**

The goal is to create an efficient tool for document comparison with an engaging user experience. Users can scan documents to find similarities (e.g., plagiarism detection), while admins manage the system and analyse usage patterns. The sleek design enhances usability and visual appeal.

**Key Functionalities**

1. User Features:

* Upload .txt files and match them against reference files using Levenshtein and Gemini algorithms.
* View matches with similarity scores (>50%) and download matched files.
* 20 free scans per day, resetting at midnight.
* Analytics showing daily scans and top topics.

2. Admin Features:

* Manage user credits (adjust or approve requests).
* Upload reference files to the uploads/ folder.
* View analytics in a table: top users, topics, daily scans, and credit stats.

3. UI/UX Design:

* Dark theme with a night sky background and 3D particles.
* Card-based layout with glowing text, hover effects, and tooltips.
* Responsive design for all screen sizes.

**Step-by-Step Workings**

The project operates through a series of interconnected components. Below is how it works from setup to user interaction.

**Step 1: Project Structure and Dependencies**

* + The project is organized into public/ (frontend files) and src/ (backend files).
  + Dependencies:
  + Node.js: Runs the server (Express, SQLite3).
  + Python: Handles Gemini API matching (google-generativeai).
  + Three.js: Adds 3D particle effects (three.min.js).
  + SQLite: Stores user data, documents, and scan counts (database.sqlite).

**Step 2:** **Server Initialization (src/server.js)**

* + The server starts with node src/server.js.
  + It sets up Express, session management, and routes for authentication, credits, documents, and analytics.
  + The database (src/db.js) initializes tables: users, documents, credit\_requests, daily\_scans.
  + An admin user (admin/123) is pre-inserted.

**Step 3:** **User Authentication (src/auth.js)**

* + Users access http://localhost:3000 (index.html).
  + Register: Submits username/password, hashed (SHA-256), and stored in the user’s table.
  + Login: Verifies credentials, sets session (req.session.user), redirects to dashboard.html (user) or admin.html (admin).

**Step 4:** **User Document Upload (src/document.js - /upload)**

* + User logs in (e.g., user), goes to dashboard.html.
  + Uploads a .txt file via the form:
  + Checks credits (>0) and daily scan limit (<20).
  + Saves to database.sqlite (documents table) with user\_id, filename, content, timestamp.
  + Updates credits (decrements) and daily\_scans (increments).
  + Returns docId (e.g., 1).

**Step 5:** **Document Matching (src/document.js - /matches/:docId)**

* + After upload, fetches /matches/[docId] (e.g., /matches/1).
  + Retrieves the uploaded document from the database using docId and user\_id.
  + Reads reference files from uploads/ folder (e.g., s1.txt).
  + Calculates similarity:
  + Levenshtein: String distance algorithm (~81.82% for similar text).
  + Gemini: Python script (gemini\_match.py) uses Google Gemini API (~80%).
  + Returns matches in JSON format if similarity > 50%.

**Step 6:** **User Dashboard Display (public/dashboard.html, script.js)**

* + Loads profile data (username, credits, past scans).
  + Shows analytics (daily scans with progress bar, top topics).
  + Displays matches in a card with filename, similarity, and download button.
  + UI: Cards float, text glows, buttons animate on hover.

**Step 7: Admin Dashboard Management (public/admin.html, script.js)**

* + Loads analytics in a table:
  + Top Users: Username, scans, credits.
  + Top Topics: Most frequent document phrases.
  + Daily Scans: User ID, count, date.
  + Credit Stats: Username, requests, total credits.
  + Manages credits: Adjusts via form (username-based).
  + Approves/denies credit requests from users.
  + Uploads reference files to uploads/.

**Step 8:** **UI/UX Rendering (public/style.css)**

* + Background: night-sky.jpg with a dark overlay.
  + Particles: Three.js renders 5000 cyan particles rotating in 3D space.
  + Cards: Gradient backgrounds, shadows, and 3D tilt on hover.
  + Table: Analytics styled with a glowing header and hover effects.
  + Responsive: Adjusts layout for smaller screens.

**Step 9:** **Analytics and Limits (src/analytics.js, credit.js)**

* + Tracks scans in daily\_scans table, resets to 20 at midnight (credit.js).
  + Analyses document content for topics (first 5 words).
  + Aggregates user data (scans, credits) and credit requests for admin stats.

**Step 10: File Download (src/document.js - /download/:filename)**

* + User clicks "Download" on a match (e.g., s1.txt).
  + Server sends the file from uploads/ if it exists.

**Example Workflow**

1. Setup:

* + Place s1.txt ("The quick brown fox leaps over the idle dog.") in uploads/.
  + Run node src/server.js.

2. User Interaction:

* + Register user, log in.
  + Upload test.txt ("The quick brown fox jumps over the resting dog.").
  + Result: Matches s1.txt (~80% similarity), displayed in dashboard.

3. Admin Interaction:

* + Log in as admin/123.
  + View analytics table, adjust user’s credits, upload more reference files.

**Technical Highlights**

1. Backend: Node.js with Express handles routing and SQLite for data persistence.
2. Matching: Combines Levenshtein (local) and Gemini API (Python) for accuracy.
3. Frontend: HTML/CSS with Three.js for a modern, interactive UI.
4. Security: Passwords hashed, session-based authentication.

**Photos**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a video game

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Conclusion**

The Cathago Document Scanner integrates document processing with a visually stunning interface and robust analytics. Each step—from upload to matching to admin management—works seamlessly to provide a functional and attractive tool. This project demonstrates skills in full-stack development, UI/UX design, and data analysis.