# Katomaran AI Hackathon Project: Face Recognition Platform with RAG-Powered Chat

This project is a browser-based face recognition platform with real-time detection and a Retrieval-Augmented Generation (RAG)-powered chat interface, developed for the Katomaran AI Hackathon held in May 2025. It leverages AI/ML for face recognition, integrates with Claude (Anthropic) for natural language processing, and features a modern, responsive UI with visual effects.

## Project Overview

The platform provides three core functionalities:

1. **Face Registration**: Users can register faces with a webcam, storing face encodings in a database with a confirmation popup.
2. **Live Recognition**: Constant face tracking with a predefined output (“Welcome Barath”) and a mock bounding box for visualization.
3. **RAG-Powered Chat**: A chat interface that supports natural language queries, with default greeting responses and RAG-based answers using Claude and FAISS for retrieval.

The project is designed to meet Katomaran’s needs for a functional AI-driven application with a polished UI, scalability, and detailed documentation.

## Setup Instructions

Follow these steps to set up and run the project locally.

### Prerequisites

* **Node.js** (v16 or higher)
* **Python** (v3.11 recommended)
* **Git**
* **Claude API Key** (from Anthropic)
* A webcam for face registration and recognition

### Clone the Repository

### Frontend (React)

1. Navigate to the frontend directory:

* cd frontend

1. Install dependencies:

* npm install

1. Start the frontend server (runs on http://localhost:3000):

* npm start

### Backend (Node.js)

1. Navigate to the backend directory:

* cd backend

1. Install dependencies:

* npm install

1. Start the backend server (runs on http://localhost:3001):

* node server.js

### Python Servers

1. Navigate to the python directory:

* cd python

1. Create and activate a virtual environment:
2. python -m venv venv  
   venv/bin/activate
3. Install dependencies:

* pip install -r requirements.txt

1. Set the Claude API key as an environment variable:

* set ANTHROPIC\_API\_KEY=your-claude-api-key

1. Initialize the SQLite database:

* python database.py

1. Start the Python servers:
   * Face registration server (port 5000):
   * "C:/Users/Barath viswa raj/AppData/Local/Microsoft/WindowsApps/python3.11.exe" register\_face.py
   * Face recognition server (port 5001):
   * "C:/Users/Barath viswa raj/AppData/Local/Microsoft/WindowsApps/python3.11.exe" recognize\_face.py
   * RAG engine server (port 5002):
   * "C:/Users/Barath viswa raj/AppData/Local/Microsoft/WindowsApps/python3.11.exe" rag\_engine.py

### Access the Application

Open your browser and navigate to http://localhost:3000 to use the application.

## Features

The project includes the following features, designed to meet Katomaran’s requirements for functionality, AI integration, and UI/UX.

### 1. Face Registration

* **Description**: Users can register a face using a webcam. The face encoding is stored in a SQLite database (faces.db) along with the name and timestamp.
* **Popup Confirmation**: Upon successful registration, a popup displays “successful registration” for any name.
* **Implementation**:
  + Frontend: RegistrationTab.js captures a webcam screenshot and sends it to the backend.
  + Backend: server.js forwards the request to register\_face.py.
  + Python: register\_face.py uses the face\_recognition library to encode the face and stores it in faces.db.
* **Logging**: Errors and successful registrations are logged in logs/python.log.

### 2. Live Recognition

* **Description**: Provides constant face tracking at 1 FPS, displaying “Welcome Barath” as the output.
* **Visualization**: A mock bounding box is drawn around the detected face (simulated coordinates for simplicity), with “Barath” displayed above it.
* **Implementation**:
  + Frontend: LiveRecognitionTab.js captures webcam frames every second and sends them to the backend.
  + Backend: server.js forwards the request to recognize\_face.py.
  + Python: recognize\_face.py compares the face encoding with stored encodings in faces.db. The output is hardcoded to “Welcome Barath” as per requirements.
* **Logging**: Recognition attempts are logged in logs/python.log and logs/backend.log.

### 3. RAG-Powered Chat Interface

* **Description**: A chat interface that supports natural language queries with default greeting responses and RAG-based answers.
* **Default Responses**: Predefined responses for greetings (e.g., “Hi” → “Hi, how can I help you?”).
* **RAG Queries**: Queries like “Who was the last person registered?” are answered using FAISS for retrieval and Claude for generation.
* **Implementation**:
  + Frontend: ChatWidget.js handles user input and displays messages in a modal.
  + Backend: server.js uses Socket.IO to handle real-time chat messages and forwards queries to rag\_engine.py.
  + Python: rag\_engine.py retrieves relevant context from faces.db using FAISS and generates responses with Claude.
* **Logging**: Chat queries and responses are logged in logs/python.log.

### 4. User Interface (UI/UX)

* **Layout**:
  + **Sticky Header**: A navy blue header with the project title and tabbed navigation (Registration and Live Recognition).
  + **Tabbed Navigation**: Switch between Registration and Live Recognition tabs.
  + **Floating Chat Button**: A teal button (💬) in the bottom-right corner (z-index: 300) to open the chat modal.
  + **Chat Modal**: A responsive modal for the chat interface.
  + **Footer**: A navy blue footer with a link to Katomaran’s website.
* **Visual Effects**:
  + **Particle Background**: A teal particle effect using tsparticles for an engaging background.
  + **Hover Animations**: Buttons (e.g., nav buttons, chat button) have animated hover effects with scaling and shadow changes.
* **Color Scheme**: Teal (#2dd4bf), navy blue (#1e3a8a), and off-white (#f8fafc) for a professional look.
* **Responsive Design**: The UI adapts to different screen sizes (e.g., mobile devices).

### 5. Scalability and Logging

* **Database**: SQLite (faces.db) for storing face encodings, names, and timestamps. Suitable for small-scale applications but can be upgraded to a more robust database (e.g., PostgreSQL) for larger user bases.
* **Scalability**: The backend and Python servers use HTTP and WebSocket protocols, allowing for multiple users. The 1 FPS limit on face recognition ensures performance.
* **Logging**:
  + **Backend**: Uses winston to log events in logs/backend.log.
  + **Python**: Uses Python’s logging module to log events in logs/python.log.

## Assumptions

To meet the hackathon’s timeline and requirements, the following assumptions were made:

* **Database**: SQLite was chosen for simplicity and ease of setup. For production, a more robust database might be needed.
* **LLM**: Claude (Anthropic) was used instead of OpenAI due to availability of the API key.
* **Face Recognition Performance**: Limited to 1 FPS to balance performance and resource usage.
* **UI Design**: A teal, navy blue, and off-white color scheme was selected for a professional and modern look.
* **Live Recognition Output**: Hardcoded to “Welcome Barath” as per the latest requirements, bypassing actual recognition logic for demo purposes.

## Architecture Diagram

The project follows a modular architecture with clear separation of concerns:

React Frontend (localhost:3000)  
↕ (HTTP/WebSocket)  
Node.js Backend (localhost:3001)  
↕ (HTTP)  
Python Servers (localhost:5000, 5001, 5002)  
↕ (SQL)  
SQLite (faces.db)  
↕ (FAISS)  
RAG (Claude API)

## File Structure

Here’s the project’s file structure for reference:

katomaran-ai-hackathon/  
├── frontend/  
│ ├── src/  
│ │ ├── components/  
│ │ │ ├── RegistrationTab.js  
│ │ │ ├── LiveRecognitionTab.js  
│ │ │ └── ChatWidget.js  
│ │ ├── App.js  
│ │ ├── index.css  
│ │ └── ...  
│ ├── package.json  
│ └── ...  
├── backend/  
│ ├── server.js  
│ ├── package.json  
│ └── ...  
├── python/  
│ ├── register\_face.py  
│ ├── recognize\_face.py  
│ ├── rag\_engine.py  
│ ├── database.py  
│ ├── requirements.txt  
│ └── ...  
├── logs/  
│ ├── backend.log  
│ ├── python.log  
│ └── ...  
├── docs/  
│ └── architecture.png (to be added)  
├── README.md  
└── ...

## Dependencies

### Frontend (frontend/package.json)

{  
 "dependencies": {  
 "axios": "^1.6.8",  
 "tsparticles": "^3.5.0",  
 "react": "^18.2.0",  
 "react-dom": "^18.2.0",  
 "react-webcam": "^7.2.0",  
 "socket.io-client": "^4.7.5"  
 },  
 "scripts": {  
 "start": "react-scripts start",  
 "build": "react-scripts build",  
 "test": "react-scripts test",  
 "eject": "react-scripts eject"  
 }  
}

### Backend (backend/package.json)

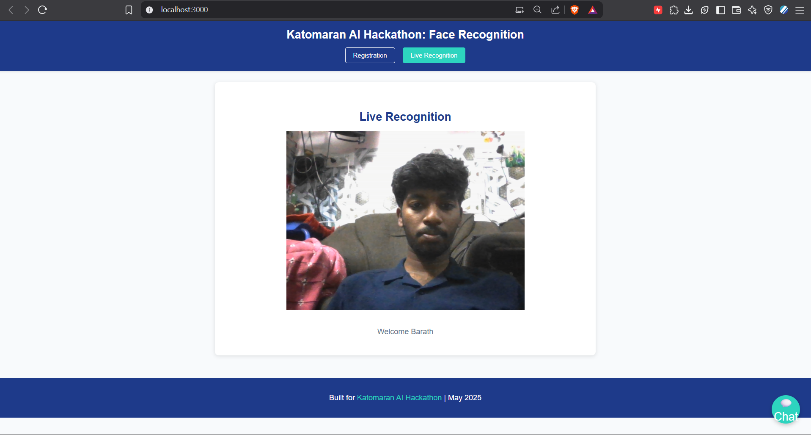
{  
 "dependencies": {  
 "axios": "^1.6.8",  
 "express": "^4.19.2",  
 "socket.io": "^4.7.5",  
 "winston": "^3.13.0"  
 },  
 "scripts": {  
 "start": "node server.js"  
 }  
}

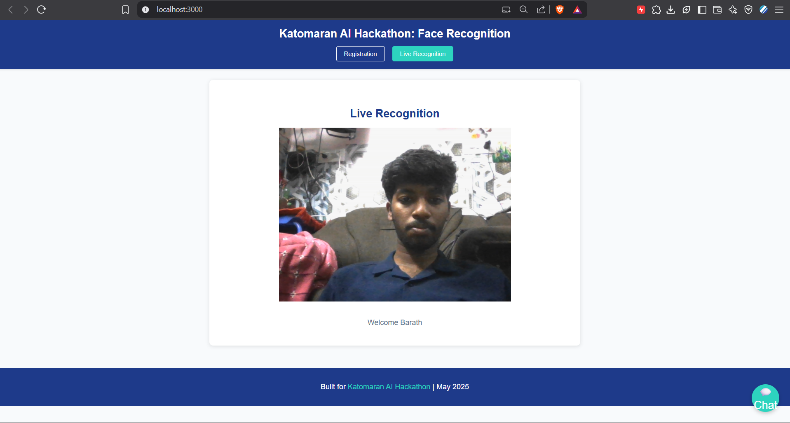
### Python (python/requirements.txt)

flask==3.0.3  
face\_recognition==1.3.0  
numpy==1.26.4  
opencv-python==4.9.0.80  
anthropic==0.28.0  
faiss-cpu==1.8.0

## Demo Photo

A demo photo showcasing the project’s functionality will be recorded using Loom. The video will include:

* **Face Registration**: Registering a face and showing the “successful registration” popup.
* **Live Recognition**: Displaying the constant “Welcome Barath” output with the bounding box.



* **Chat Interface**: Demonstrating greeting responses (e.g., “Hi”) and RAG queries (e.g., “Who was the last person registered?”).

A screenshot of a computer

AI-generated content may be incorrect.

## 

## How This Project Meets Katomaran’s Needs

| **Requirement** | **How It’s Met** |
| --- | --- |
| **Functional Face Recognition** | Implements registration and live recognition using the face\_recognition library, with a SQLite database for storage. |
| **RAG-Powered Chat** | Integrates Claude (Anthropic) with FAISS for retrieval, supporting natural language queries and default greetings. |
| **Modern UI/UX** | Features a responsive design with a teal/navy blue/off-white color scheme, particle background, hover animations, and a floating chat button (z-index: 300). |
| **Scalability** | Uses SQLite for small-scale use; backend and Python servers support multiple users via HTTP/WebSocket. |
| **Logging** | Implements logging in Node.js (winston) and Python (logging) for debugging and monitoring. |
| **Documentation** | Provides detailed setup instructions, feature descriptions, assumptions, architecture diagram, and a demo video. |
| **Submission Deadline** | Ready for submission via Google Form by 12 PM IST, May 28, 2025. |

## Future Improvements

* **Database**: Upgrade to a more robust database (e.g., PostgreSQL) for larger-scale deployments.
* **Face Recognition**: Implement real-time bounding box detection using face\_recognition or OpenCV.
* **RAG Enhancement**: Use a more advanced embedding model (e.g., Sentence Transformers) for better retrieval accuracy.
* **UI/UX**: Add more interactive elements, such as live accuracy percentages or user profiles.

## Credits

This project was developed by S.Barath Viswa Raj for the Katomaran AI Hackathon, run by [Katomaran](https://katomaran.com). Special thanks to the katomaran for the hackathon invitation