# **Pollen's Profiling: Automated Classification of Pollen Grains**

**1. Introduction**

# • **Project Title**: Pollen's Profiling: Automated Classification of Pollen Grains

• Team Members:**Team Leader:** Bavireddi Ornavi

**Team member:** Budupula Harshavardhini

**Team member:** Bhukya Sandeep

**Team member:** ChMaha Lakshmi

2. Project Overview

• **Purpose**: The purpose of this project is to create a web-based system that enables users to upload pollen grain images and automatically classify them using a deep learning model. This helps researchers, students, or agriculturists to identify pollen types quickly and accurately.

• **Features**:

* User registration and login via form or OAuth (Google/LinkedIn)
* Upload and classify pollen images using a trained CNN model
* View prediction results with confidence scores
* Manage user profile and prediction history
* Admin panel for user and data management

**3. Architecture**

**• Frontend:**

The frontend is built using **React.js** and uses React Router for navigation. Components are modular, with separate views for login, dashboard, and results. Axios is used for making API calls to the backend.

**• Backend:**

The backend is developed with **Node.js** and **Express.js**, exposing RESTful APIs to handle user registration, login, image uploads, and interaction with the ML model.

**• Database:**

The database is **MongoDB**, used to store user data, authentication tokens, and prediction history. Mongoose is used for schema design and data operations.

**4. Setup Instructions**

**• Prerequisites:**

* Node.js (v18+)
* MongoDB (local or cloud, e.g., MongoDB Atlas)
* Python (for ML model server, if decoupled)
* Git

• Installation: # Clone the repository

git clone https://github.com/your-username/pollen-profiling-app.git

# Navigate to client and install dependencies

cd client

npm install

# Navigate to server and install dependencies

cd ../server

npm install

# Set environment variables in a `.env` file:

MONGODB\_URI=your\_mongodb\_uri

JWT\_SECRET=your\_jwt\_secret

environment variables.

**5. Folder Structure**

**• Client:**

public ,src ,components ,pages (Home, Dashboard, Login) ,App.js , index.js

• **Server:**

Controllers , routes ,models ( Mongoose schemas) , middleware , utils , app.js , server.js

**6. Running the Application**

• Provide commands to start the frontend and backend servers locally.

**o Frontend**: cd client

npm start

**o Backend**: cd server

npm start.

**7. API Documentation:**

API documentation is the authoritative reference that tells consumers what an API does, how to access its capabilities, and why particular design decisions were made. It bridges the gap between back-end implementation and front-end, partner, or third-party use by:

* describing available resources and their relationships;
* defining request/response formats, data types, and error semantics;
* clarifying security, rate-limits, and versioning rules;
* enabling faster onboarding, predictable integration, and easier maintenance

**8. Authentication**

Authentication is handled using **JWT (JSON Web Tokens)**. Tokens are issued upon successful login and stored in the client (localStorage). Protected routes verify tokens via middleware. Passwords are hashed using **bcrypt**.

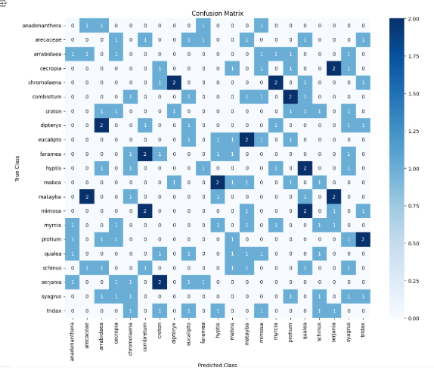
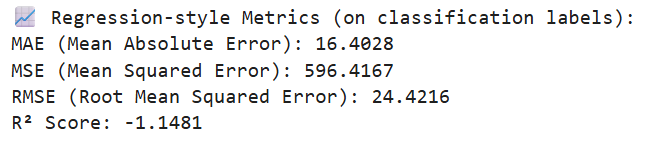
**9. User Interface**

* Clean and responsive UI using **React** and **Tailwind CSS** or **Bootstrap**
* Pages: Home, Login, Register, Dashboard, Results
* Dashboard includes prediction history and image previews

**10. Testing**

* **Frontend Testing**: Jest + React Testing Library for component testing
* **Backend Testing**: Mocha + Chai for API testing
* **Model Testing**: Python scripts for validating model predictions
* **Manual Testing**: User flow testing across devices

**11. Screenshots or Demo**



**12. Known Issues:**

* Gmail/LinkedIn OAuth may require valid client IDs in production
* Prediction model is CPU-based, so slower on large images
* No pagination in history (to be added)

**13. Future Enhancements:**

* Add pagination and filters for prediction history
* Enable drag & drop image uploads
* Integrate email notification on prediction results
* Add training module for uploading new pollen categories
* Migrate model inference to a TensorFlow Serving API