# Project Documentation format

# 1. Introduction

- **Project Title:** Pollen's Profiling: Automated Classification of Pollen Grains
- **Team Members:** Divvela Navya

# 2. Project Overview

- **Purpose:** An end-to-end AI-based pollen grain image classification system. It allows users to upload microscopic images of pollen grains and get real-time predictions powered by a trained deep learning model hosted via Google Colab.
- **Features:**
  - Upload pollen grain images for classification
  - Real-time prediction and display of pollen type
  - Clean, responsive UI built with React
  - RESTful API backend for image processing and prediction

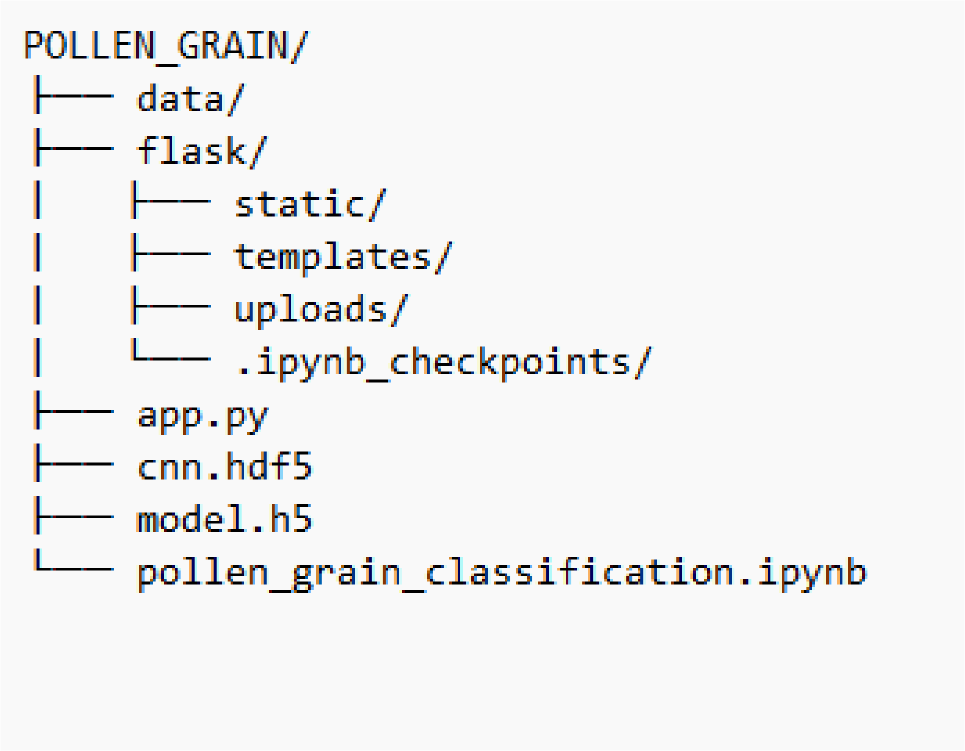
# 3. Architecture

- **Frontend:**
  - Technology Stack: HTML5, CSS3, JavaScript
  - Purpose: Provide a simple interface for users to upload pollen grain images and view predictions.
  - Pages:
    - index.html: Main upload interface
    - result.html: Displays predicted class and confidence
  - Client-side Logic: AJAX call using fetch() to send the image to backend API and route user to result page.
- **Backend:** A Flask-based web server is developed in app.py, and it includes multiple routes:
  - Home page route (index.html)
  - prediction.html': Upload and trigger classification
  - logout.html': Render final results
  - '/result': Accepts the uploaded image, preprocesses it, performs model inference using a pre-trained deep learning model, and returns the classification output.

## 4. Setup Instructions

- **Prerequisites:**
  - Python  3.8
  - Google Account (to use Colab)
  - Browser with JavaScript enabled
  - Basic web server (optional): Python’s http.server or Live Server extension
- **Installation:** Step-by-step guide to clone, install dependencies, and set up the environment variables.

## 5. Folder Structure

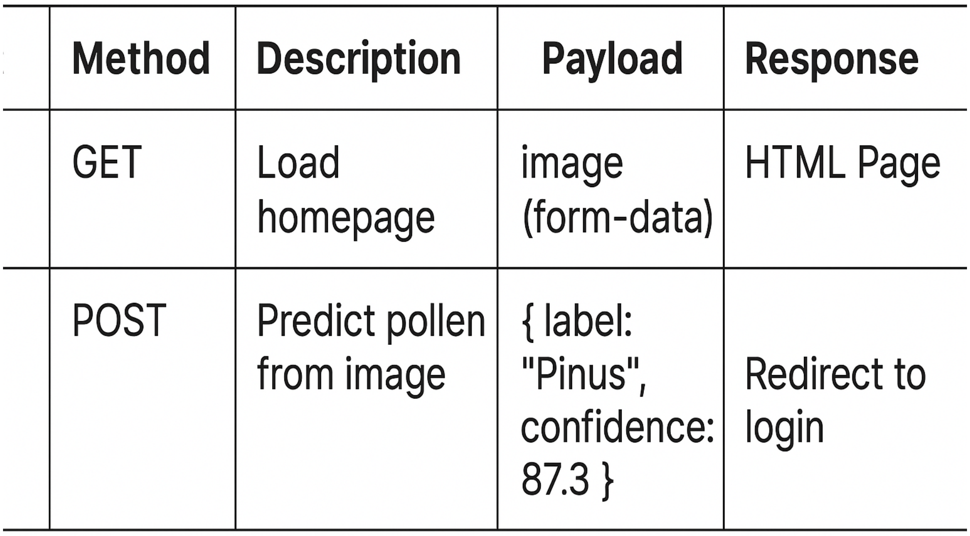


```
POLLEN_GRAIN/
├── data/
├── flask/
│    ├── static/
│    ├── templates/
│    ├── uploads/
│    └── .ipynb_checkpoints/
├── app.py
├── cnn.hdf5
├── model.h5
└── pollen_grain_classification.ipynb
```

## 6. Running the Application

- Step 1: Create and activate environment
- Step 2: Install dependencies pip install flask keras tensorflow numpy
- Step 3: Start the Flask server python app.py

## 7. API Documentation



| | Method | Description | Payload | Response |
|---|---|---|---|---|
| | GET | Load homepage | image (form-data) | HTML Page |
| | POST | Predict pollen from image | { label: "Pinus", confidence: 87.3 } | Redirect to login |

## 8. Authentication

- Likely session-based using Flask’s session object.
- User credentials can be stored in server-side sessions.
- Routes like /logout use session.clear() to end login state.

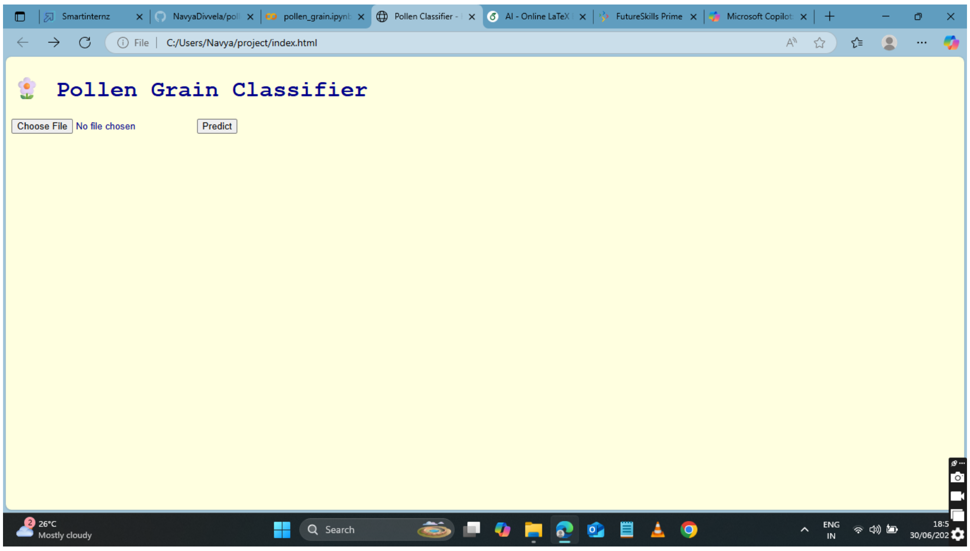
## 9. User Interface

- index.html: Likely hosts the upload form.
- prediction.html: Displays predicted pollen type with confidence.
- logout.html: Handles session termination or redirect.

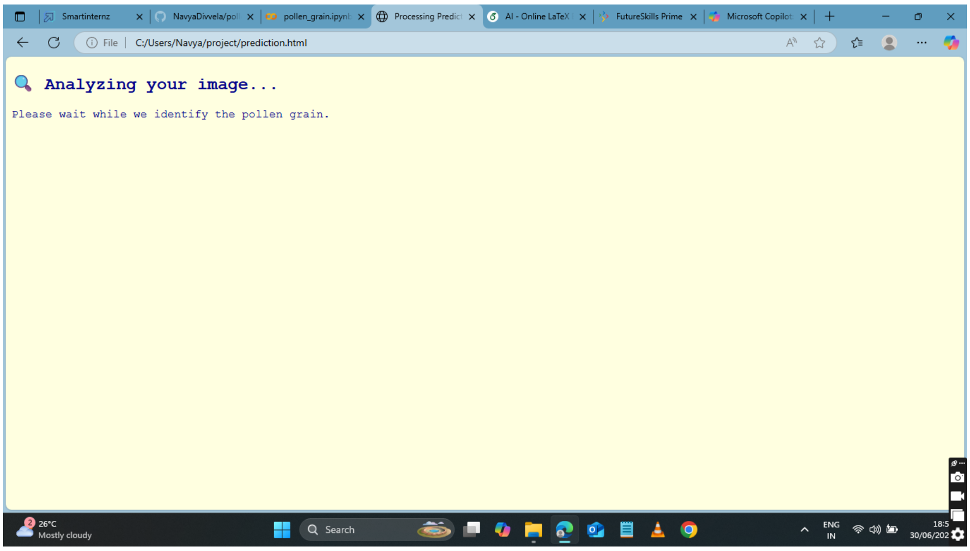
## 10. Testing

- Unit Tests: Use pytest to test utility functions, image preprocessing, model loading, etc.
- Integration Tests: Use FlaskClient to test endpoints like /predict with mock images.
- Model Tests: Evaluate model performance with a held-out validation dataset and report accuracy, precision, recall.

# 11. Screenshots or Demo



🌸 Pollen Grain Classifier



🔍 Analyzing your image...

Please wait while we identify the pollen grain.

# 12. Known Issues

- UI doesn’t show error if model fails to predict
- No image preview on result page
- Model confidence may be skewed if dataset is imbalanced
- No persistent logging of user interactions

# 13. Future Enhancements

- Add user authentication and session tracking

- Batch image prediction feature
- Class explanation tooltips (e.g., visual pollen references)
- Export prediction as PDF report
- REST API + mobile-friendly design