# 

# Plant Disease Detection

# Réalise par : Encadré par :

**BAHRAOUI Mohamed** **Mr. MAHMOUDI Abdelhak**

1. **Introduction**

This project focuses on developing a plant disease detection system using a trained CNN model. By leveraging advanced machine learning techniques, the system aims to accurately identify and classify plant diseases. The technologies used include CNN, Django, ReactJS, and Firebase.

The CNN model is the core of the system, trained on a diverse dataset comprising images of different plants and associated diseases. The Django framework is employed to build the backend API, which receives an image link and returns disease predictions and suggestions. ReactJS is used to create an intuitive and responsive frontend interface.

Firebase storage is integrated into the system for efficient image upload and retrieval. The combination of these technologies enables seamless communication between the frontend and backend components.

Overall, this project aims to provide a comprehensive solution for plant disease detection, catering to the needs of farmers and agricultural experts. The system leverages the power of CNNs, coupled with Django, ReactJS, and Firebase, to enhance crop yield by promptly detecting and addressing plant health issues.

1. **Methodology**

**A. Training the CNN Model**

The dataset used for training the CNN model contains images of various plants affected by different diseases. After some preprocessing we got the dataset, named "img\_info," consists of image paths, labels, and label integers, covering 18 distinct plant disease categories.

The CNN model architecture follows a specific pattern for image classification. It starts with Convolutional layers, applying ReLU activation and MaxPooling for downsampling. Dropout layers are added to prevent overfitting. The feature maps are flattened and connected to Dense layers with ReLU activation. The final Dense layer uses softmax activation for classification.

The training process involves data preprocessing, including resizing the images to 64x64 pixels and normalizing pixel values. The dataset is split into training, validation, and testing sets. The model is compiled using the Adam optimizer with categorical cross-entropy loss and accuracy metric. It is trained for 100 epochs in batches, with validation data used for monitoring and the best model saved.

During training, the model learns to recognize patterns and adjust its parameters to improve accuracy. The Adam optimizer helps optimize the learning rate and minimize loss. The trained model can be saved for future predictions on new images.

**B. Creating the Django Backend API**

In the backend, the Django REST Framework is used to create the API, which receives an image link from the frontend. The image is processed and passed to the trained CNN model for disease prediction. Additionally, the backend utilizes the OpenAI API to generate suggestions for further action based on the predicted disease. The Django REST Framework handles the routes and endpoints for communication with the frontend, while the OpenAI API provides valuable insights and recommendations for plant health management.

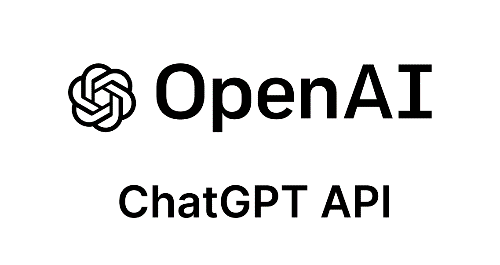
Here are the technologies used in the backend:



Django is an open-source web framework written in Python, designed for efficient web application development. It follows the MVC pattern and provides built-in features such as ORM, URL routing, forms, authentication, and an admin interface. With a templating engine and focus on security, Django simplifies development while allowing extensibility through third-party libraries.



Django Rest Framework (DRF) is a powerful extension to the Django web framework that enables the creation of Web APIs. It provides a set of tools and libraries that simplify the process of building RESTful APIs in Django. DRF includes features such as serialization, authentication, permissions, viewsets, and pagination, which help developers quickly develop robust APIs. It follows best practices for API design and integrates seamlessly with Django's ORM and authentication system. DRF also offers support for popular API authentication methods like token-based authentication and OAuth.



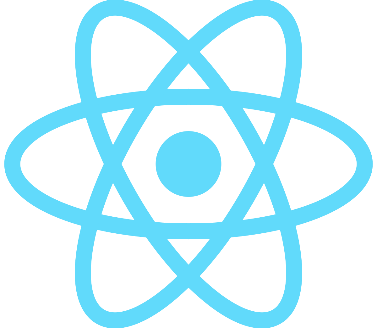
Using the OpenAI Chat API, you can build your own applications with gpt-3.5-turbo and gpt-4 to do things like:

* Draft an email or other piece of writing
* Write Python code
* Answer questions about a set of documents
* Create conversational agents
* Give your software a natural language interface
* Tutor in a range of subjects
* Translate languages
* Simulate characters for video games and much more

**C. Creating the Django Backend API**

The ReactJS frontend enables users to upload an image, receive plant disease predictions, and view suggestions for further action. It consists of components for image upload, prediction display, and suggestions. Firebase Storage is integrated for image uploading and retrieval. The frontend communicates with the backend API using HTTP requests, sending the image link as a payload. The backend processes the image, predicts the disease, and returns the results. The frontend updates the UI to display the prediction and suggestions received from the backend. Overall, the ReactJS frontend provides a seamless and interactive user experience for plant disease detection.

Here are the technologies used in the frontend:



React.js is an open-source JavaScript library for building user interfaces. It follows a component-based architecture and uses a virtual DOM for efficient updates. JSX syntax allows developers to write HTML-like code alongside JavaScript. React's unidirectional data flow simplifies state management. It has a sibling library called React Native for building native mobile apps. React has a large ecosystem of third-party libraries and tools. It is widely adopted and used for dynamic and responsive UIs.

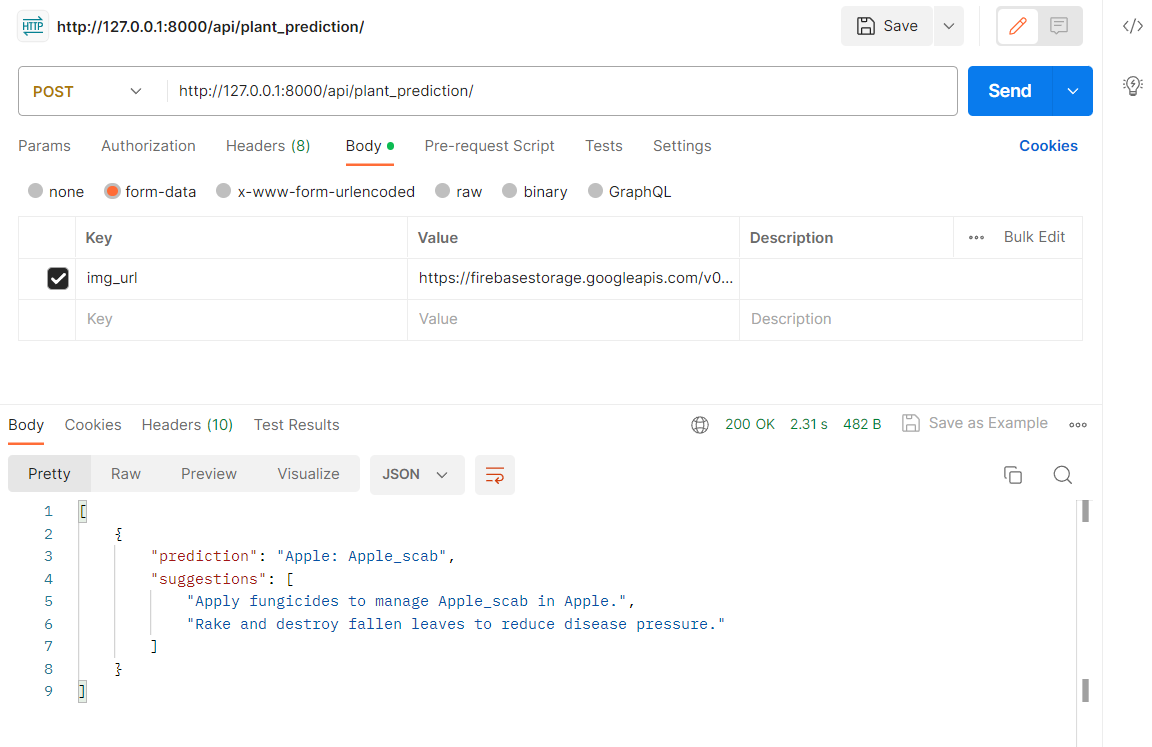


Firebase Storage: Cloud-based storage service by Firebase. Securely store and serve user-generated files. Access control rules, file metadata, and resumable uploads. Unique URLs for direct file access. Integration with Firebase ecosystem. Client SDKs for web, iOS, Android. Usage monitoring and analytics. Simplifies file management in web and mobile apps.

The frontend uses other technologies such as HTML, CSS, BOOTSTRAP AND Javascript

1. **Realization**

**A. Test the backend API using postman**

****

The Postman screen facilitates sending a POST request to the API endpoint at <http://127.0.0.1:8000/api/plant_prediction>. It consists of the following elements:

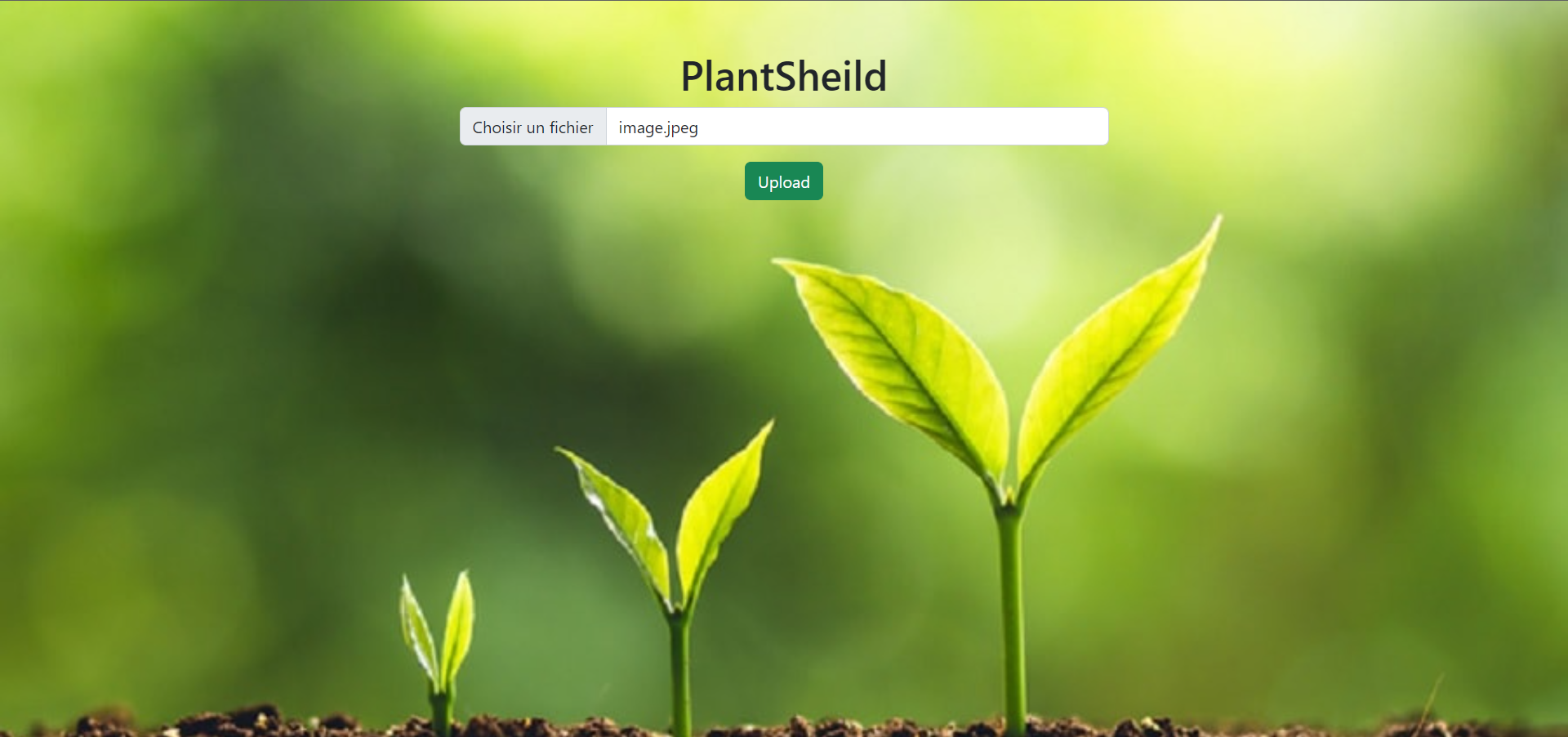
* Request Method and URL: Displays "POST" and the target API endpoint.
* Headers: Allows customization of request headers.
* Body: Defines the request payload as a JSON object with {'img\_url': img\_url}.
* Send Button: Triggers the execution of the POST request.
* Response: Displays the server's response, which is in the form {'predictions': predictions, 'suggestions': [suggestion1, suggestion1]}.

Developers can input the desired 'img\_url' in the Body section and click Send to execute the request. The server's response will include the 'predictions' and 'suggestions' based on the provided image URL. The Postman screen provides an intuitive interface for interacting with the API and viewing the response.

**B. Interfaces**



In the interface designed for detecting plant diseases, the user starts by selecting an image of a plant that they suspect may be affected by a disease. This image can be chosen from their local device.



Once the user has selected the image, an "Upload" button becomes visible, indicating that they can proceed with uploading the selected plant image for disease detection.



Upon clicking the "Upload" button, the interface begins the upload process.Then the interface generates predictions and suggestions related to the detected plant disease. The predictions typically include the name or type of disease affecting the plant, while the suggestions may offer recommendations for treatment, further examination, or preventive measures.