SOFTWARE REQUIREMENTS SPECIFICATION
for
PLANT DISEASE CLASSIFICATION

Version 1.0

Prepared by :

MOHAMMAD NASIF SADIQUE KHAN

1 Introduction

1.1 INTRODUCTION TO PLANT DISEASE CLASSIFICATION:

A plant disease classification system is a web application which can run on any local server. It can detect plant disease by comparing a sample plant leaf and a diseased plant leaf. It will contain all the necessary information about diseased plant leaves so that it can detect any diseased plant accurately. This system is built for the farmers so that they can find out whether their potato or bell pepper plant is diseased or not, easily by clicking a picture and uploading it to the system. And if the plant is diseased, they can take necessary steps to prevent it.

1.2 PURPOSE:

The document is about the Software Requirements Specification (SRS) for the "Potato and Bell pepper plant Disease Classification System". The purpose of this document is to give a detailed description of the requirements for the "Potato and Bell pepper plant Disease Classification System". The software requirements are collected from the farmers which is clear enough for the project development. To derive this project, we had to collect all the necessary information from the experienced farmer, collect dataset of all the healthy and diseased plants and understand how and what type of disease occurs in the plant and how it can be detected.

1.3 OVERVIEW:

Frontend: React Application
The frontend of the project is built using the React framework. It includes various UI components from the Material-UI library to create an appealing and user-friendly interface. The primary purpose of the frontend is to provide users with a way to upload images, visualize uploaded images, display disease classification results, and clear/reset the process. Major Components and Features:

- App-Bar: A Material-UI App-Bar is used to display the application's title and logo.
- Image Upload: Users can use the "Drop zone Area" component to upload leaf images.
- Card Display: The uploaded image is displayed in a Material-UI Card component.
- Processing Status: While processing the uploaded image, a loading spinner and text are displayed to indicate ongoing processing.
- Classification Results: After processing, the front end displays the predicted disease class along with the confidence level.
- Clear Button: A "Clear" button allows users to reset the application to its initial state.

Backend: FastAPI Server
The backend of the project is powered by FastAPI, a modern Python web framework. It provides an API endpoint that receives image files, processes them using a pre-trained machine learning model, and returns disease classification results. Major Components and Features:

- CORS Middleware: CORS (Cross-Origin Resource Sharing) middleware is added to enable the frontend application hosted on a different domain to communicate with the backend server.
- Model Loading: The backend loads a pre-trained TensorFlow model that is capable of classifying images into different disease categories.
- Prediction Endpoint: The "/predict" POST endpoint accepts image uploads, processes them using the loaded model, and returns the predicted disease class along with the confidence score.
- Class Names: The backend uses a list of class names corresponding to different disease categories.

Workflow: User Interaction and Data Flow

- A user accesses the web application's frontend interface.
- The user uploads an image of a potato or bell pepper leaf using the "DropzoneArea."
- The uploaded image is displayed in the frontend's Card component.
- The frontend sends the image to the backend's prediction endpoint via an API request.
- The backend receives the image, processes it using the pre-trained model, and returns the disease classification.
- The frontend displays the predicted disease class and the confidence level to the user.
- If the user wants to clear the results or upload a new image, they can use the "Clear" button, which resets the frontend interface.

1.4 ELICITATION TO PLANT DISEASE CLASSIFICATION:

The task that helps the developer to understand and define what is required to build a software and collect everything that is required is Elicitation. In this step we had to face many difficulties. We had to study plant behavior. If a disease of the plant occurs, what changes can be noticed in the plant leaf. Also we struggled to find all the reliable dataset as we had used image processing to identify the disease. To get maximum accuracy the data set available was not enough. So we have created some of our own samples. While we were doing that, we had a hard time finding the leaves of different plants of different diseases. We also had to communicate with farmers to understand how to find out and figure out a disease of a plant. And how can we make the software easy for them to use.

2 Development Stage Overview:

2.1 Image Classification Model's Methodology:

As shown in Fig. 1. The proposed methodology in this project includes the following four main steps: data acquisition, data prepossessing, data augmentation, and image classification.

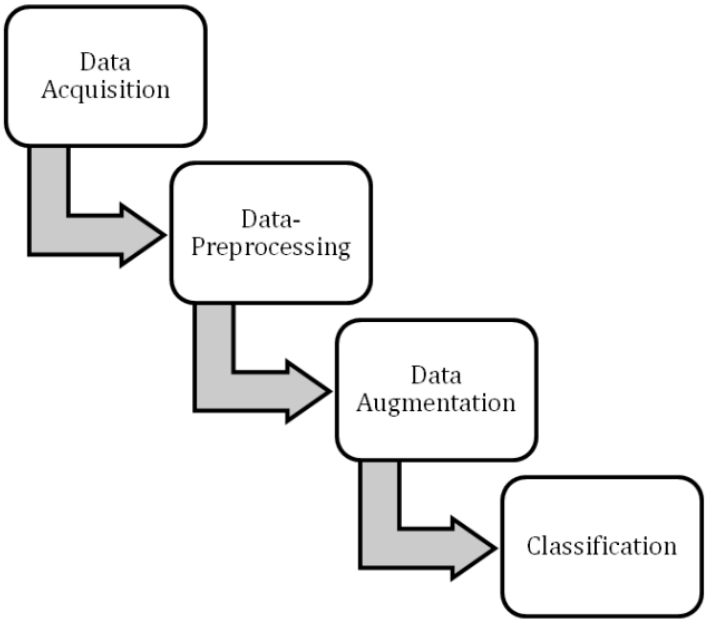

Fig. 1. Proposed methodology

Figure 2.1: methodology

2.2 Used Dataset Description:

We obtained different image resolutions and sizes from both a potato plantation and a bell pepper plantation in Patuakhali, using an open-access image database from Kaggle. In our research, we used a total of 4,627 images. These images were divided into five classes, which include Early Blight, Late Blight, and Healthy for potato, as well as Bacterial spots and Healthy for bell pepper.

2.3 Early Blight:

Early Blight, caused by the bacterium Alternaria solani, manifests as tiny black dots that develop into large, brown-to-black, round-to-ovoid lesions. These lesions are sometimes confined by leaf veins but can also be associated with lenticels. The undersides of the leaves then develop a black fungus. Early Blight can lead to tuber wilt in potatoes and typically spreads when temperatures exceed 26°C. It often occurs when potato plants are stressed due to factors like high-temperature drying or insufficient fertilization.

Fig. 2. Early blight disease of potato leaf

Figure 2.2: Early blight

2.4 Late Blight:

Late Blight, caused by the bacterium Phytophthora infestans, can inflict significant damage on potato crops, especially in years with low temperatures and ample rainfall.

Figure 2.3: Late Blight

2.5 Bacterial Spot

Bacterial Spot can devastate a pepper crop by causing early defoliation of infected leaves and disfiguring fruit. In severe cases, plants may die, and finding a cure becomes extremely challenging. However, there are various measures that growers can take to prevent the disease from occurring and spreading.

Figure 2.4: Bacterial spot

2.6 Healthy Leaf:

Healthy Leaf indicates that the leaves are fresh and not infected with any disease, applicable to both pepper and potato plants.

2.7 Data pre-processing:

Regarding data pre-processing, we ensure that the images are cropped to focus on the region of interest and reduce image noise. Images with excessive noise are discarded. Furthermore, input photos are scaled to a consistent size of 256x256 pixels after being collected from various sources with varying sizes.

2.8 Data augmentation :

Data augmentation is an essential technique employed in this study to modify data without altering its original meaning. We apply various geometric transformations, such as translations, rotations, scale changes, shearing, and vertical and horizontal flips, to augment the dataset. The data augmentation pipeline is created using "layers.experimental.preprocessing.RandomFlip" and "layers.experimental.preprocessing.RandomRotation," and these transformations are applied to the training dataset to enhance the images before feeding them into the model.

2.9 Image Classification

Image Classification involves assigning labels or classes to images based on their content. In our code, we utilize the VGG16 model, a pre-trained deep learning model designed for image classification. Here's an overview of the steps involved:

- Loading the Pre-trained VGG16 Model: We employ the VGG16 model pre-trained on the ImageNet dataset, which has learned a hierarchy of features from a vast number of images.
- Freezing Pre-trained Layers: To preserve the knowledge gained from the ImageNet dataset, we freeze the layers of the pre-trained VGG16 model, preventing them from being updated during training.
- Adding Custom Layers: On top of the VGG16 model, we add a Global Average Pooling layer to reduce feature dimensionality and a Dense layer with the number of classes in our dataset. The final layer uses a softmax activation function for multi-class classification.
- Compiling and Training the Model: We compile the model with the specified optimizer and loss function, then train it using the training dataset, validation dataset, and callbacks. The ModelCheckpoint callback saves the best model based on validation accuracy.

2.10 Selecting the Best Epoch:

To pick the best state of this fine tuned model, we plotted the accuracy and loss and as per the plot and epochs history.

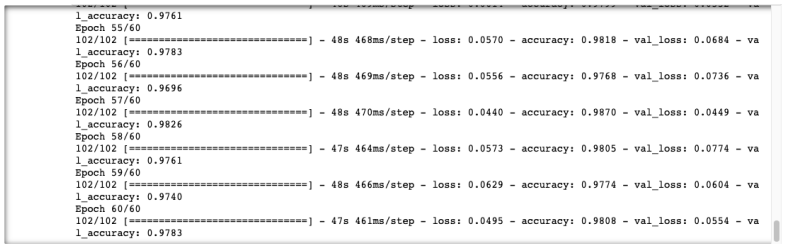

Figure 2.5: Glimps of Epoch History

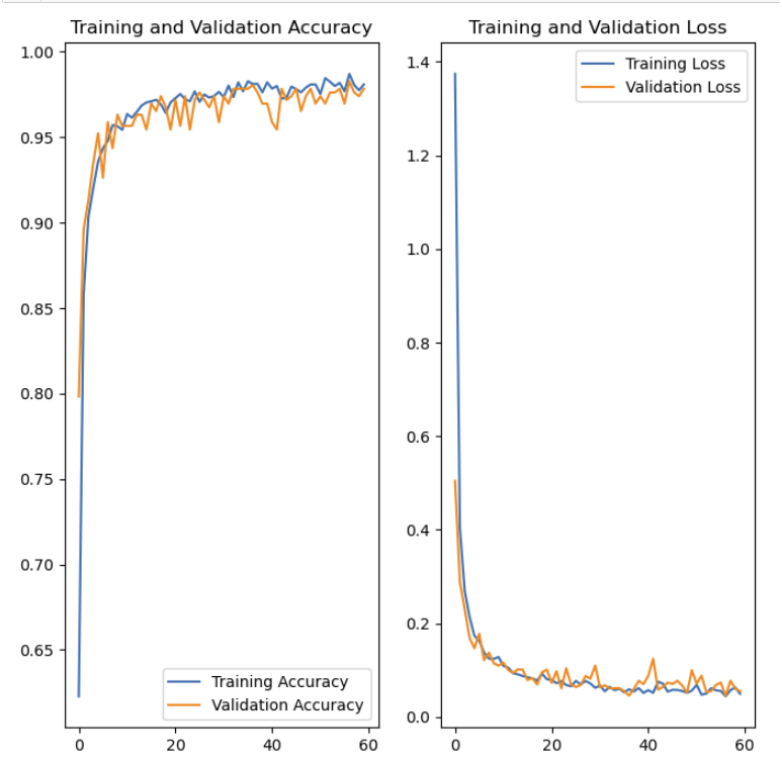

Figure 2.6: Plots of Epochs

According to both plots and history, the 57th from this model had the best classification results on Training and Validation. Hence, we picked the model from 57th epoch.

We seamlessly integrated our image classification model with our user-friendly frontend using FastAPI. Our frontend is designed with a clean and intuitive user interface, featuring HTML, CSS, JavaScript, and Node.js to provide a smooth user experience. We've incorporated a simplified interaction method that includes a drag-and-drop feature and a click-to-open functionality.

3 FastAPI Integration:

FastAPI serves as the bridge between our deep learning model and the frontend. It allows us to create a robust API that receives image files from the frontend, processes them through our model, and sends back the classification results.

Here's how the integration works:

- Endpoint Setup: We've created an API endpoint using FastAPI that listens for incoming image uploads or requests from the frontend. This endpoint is responsible for handling image data and returning classification results.
- Image Processing: When an image is uploaded via drag-and-drop or by clicking to open, FastAPI processes the image data, ensuring it's in a suitable format for our deep learning model.
- Model Inference: FastAPI then passes the preprocessed image to our pre-trained VGG16 model for classification. The model makes predictions based on the image content, determining the class or label associated with it.
- Result Transmission: The classification results, including the predicted class and associated confidence scores, are sent back to the frontend in a structured JSON format.

3.1 Frontend User Experience:

Our frontend is designed with simplicity and user-friendliness in mind. Users can interact with it using the following methods:

- Drag-and-Drop Feature: Users can simply drag an image file and drop it onto the designated area on the webpage. The frontend then handles the file upload and communicates with the FastAPI backend for classification.
- Click to Open: Alternatively, users can click on an area to open their file explorer and select an image file for classification. The selected image is then processed in the same manner as with the drag-and-drop feature.

3.2 Node.js for Backend Interaction:

Node.js plays a crucial role in our frontend-backend communication. It helps facilitate real-time interactions between the user interface and the FastAPI backend. Node.js is responsible for making requests to the FastAPI endpoints and handling responses, ensuring a seamless and responsive user experience.

In summary, our integrated system combines the power of FastAPI for handling image classification requests with the user-friendly features of our frontend, allowing users to effortlessly upload and classify images using drag-and-drop or click-to-open functionalities. This cohesive solution provides a streamlined experience for users while leveraging advanced image classification capabilities.

4 USAGE SCENARIO:

The Plant Disease Detection System, integrated into our web application, harnesses the power of advanced deep learning technology, specifically a Convolutional Neural Network (CNN), to revolutionize crop disease management for potato and pepper plants. Our user-friendly web application enables farmers to effortlessly identify diseases by uploading leaf images. These images are swiftly processed and analyzed by our pretrained deep learning model, categorizing them as "Healthy" or "Diseased." In cases of disease detection, the system further classifies the disease type, such as "Early Blight," "Late Blight," or "Bacterial Spot." The application provides rapid reports and actionable recommendations, allowing farmers to make informed decisions and minimize crop losses.
With speed and accuracy, the system generates rapid reports indicating the health status of the uploaded leaves. In cases of disease detection, the report specifies the probable disease type, enabling farmers to promptly take informed actions. Recommendations for potential treatments, interventions, or preventive measures are provided, assisting farmers in minimizing crop losses and managing disease spread effectively.
An invaluable aspect of the system lies in its continuous learning and improvement process. By accumulating user interaction data, including uploaded images and subsequent farmer actions, the system periodically fine-tunes the deep learning model. This iterative enhancement process ensures that the model becomes increasingly accurate over time, broadening its ability to detect an expanding range of diseases and adapt to emerging disease patterns.
In practice, the Potato and Pepper Leaf Disease Detection System embodies a sophisticated synergy of deep learning training, application deployment, secure data management, and real-world problem-solving in image processing and disease recognition. This holistic approach promises to transform crop disease management, enhancing the resilience and productivity of potato and pepper cultivation.

5 SCENARIO BASED MODELING OF PLANT DISEASE CLASSIFICATION:

This chapter describes the scenario based modeling of "Plant Disease Classification System".

Scenario based modeling comprises of three diagrams:

- Use-case Diagram
- Class Diagram
- Flowchart

5.1 Use-case Diagram

A use case diagram is a graphical representation that helps software developers have a visualization and clear idea about how the user is interacting with the functionalities of a software. And based on the interaction, how the software is reacting. The components of Use- Case Diagram are,

- Actors, Individuals or roles that interact with the software. There are two types of actors. They are primary and secondary actors.
- The use cases. These use cases are the functionalities of a software that reacts based on actors performances.
- Relation, which is the connection between actors with use cases and use cases with other use cases. The relation between use cases with another use cases can be include and exclude

In the project, "Plant Disease Classification System", We have

- Primary Actors: Farmers
- Secondary Actors: System
- Use Cases: Upload Image, Clear Picture, Predict Disease, and Show Result

Use Case

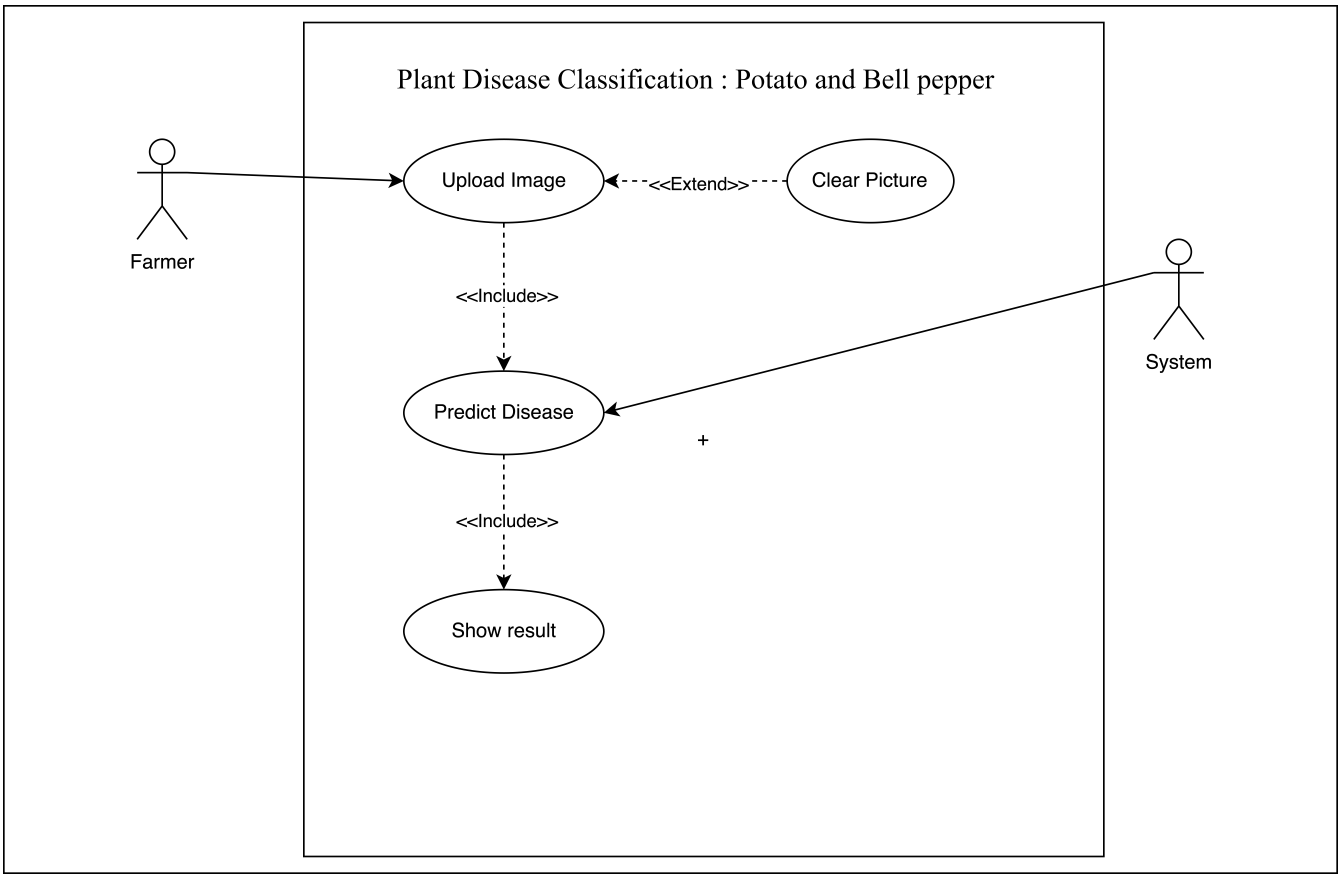

Figure 5.1: Use Case Diagram

5.2 Class Diagram

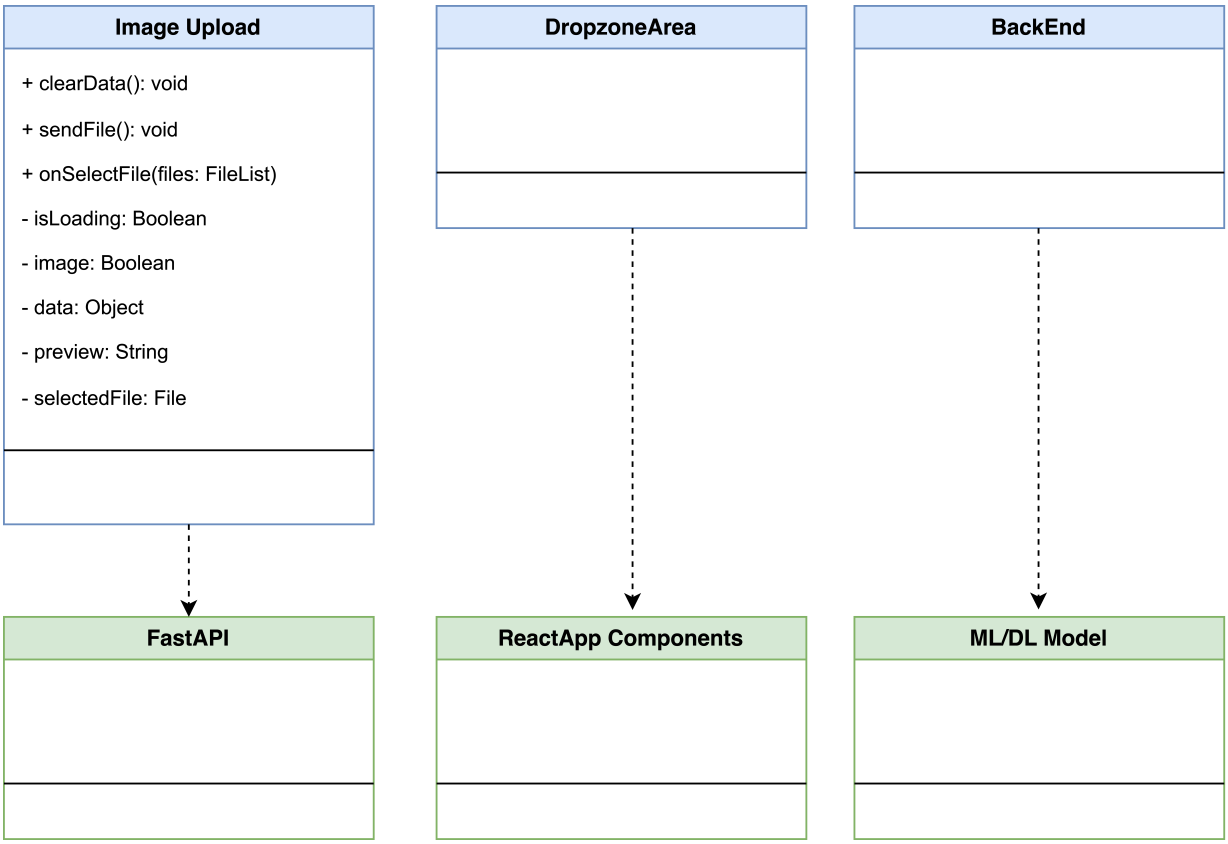

Figure 5.2: Class Diagram

5.3 Flow chart

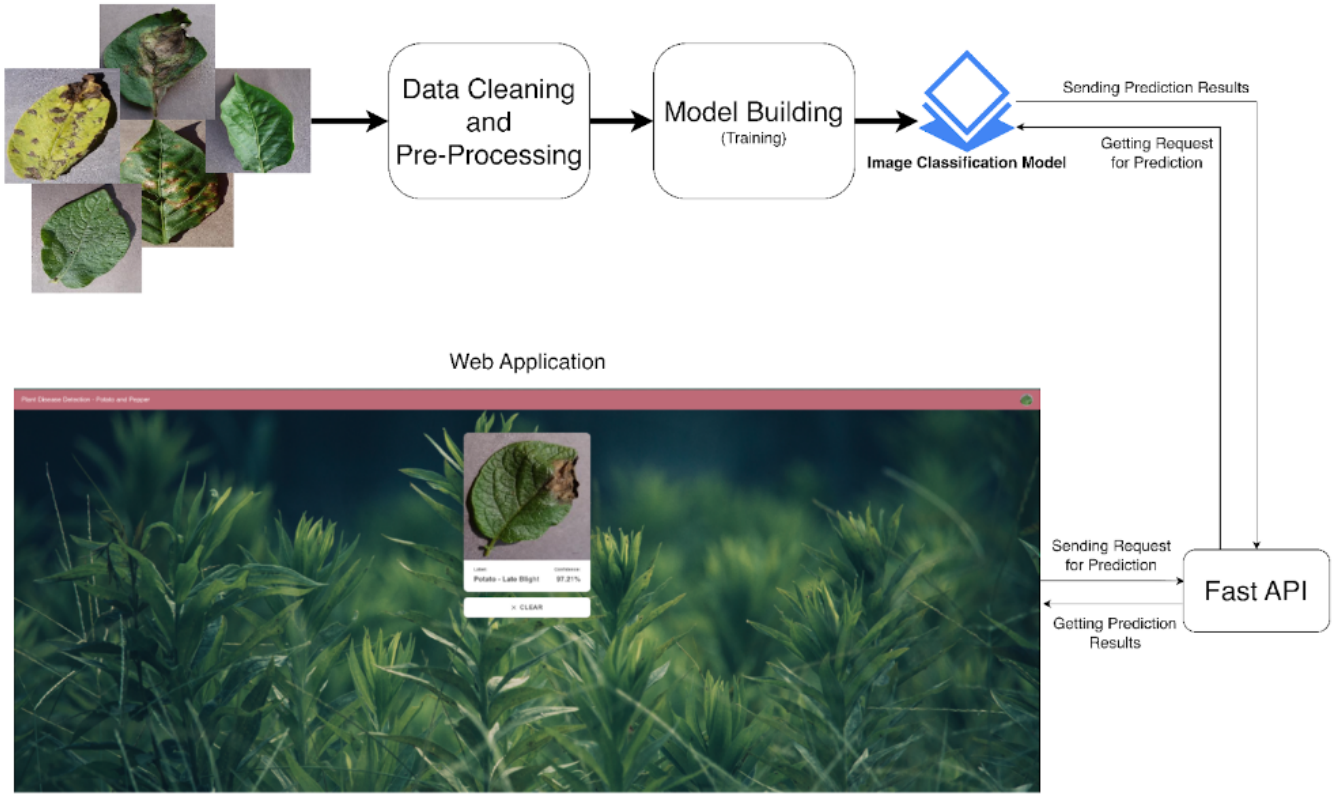

Figure 5.3: Flow chart

6 User Guide:

6.1 Getting Started

At the very start of this web application's home screen, you will have an upload area where one can upload an image of a potato or pepper bell plant in order to predict the disease.

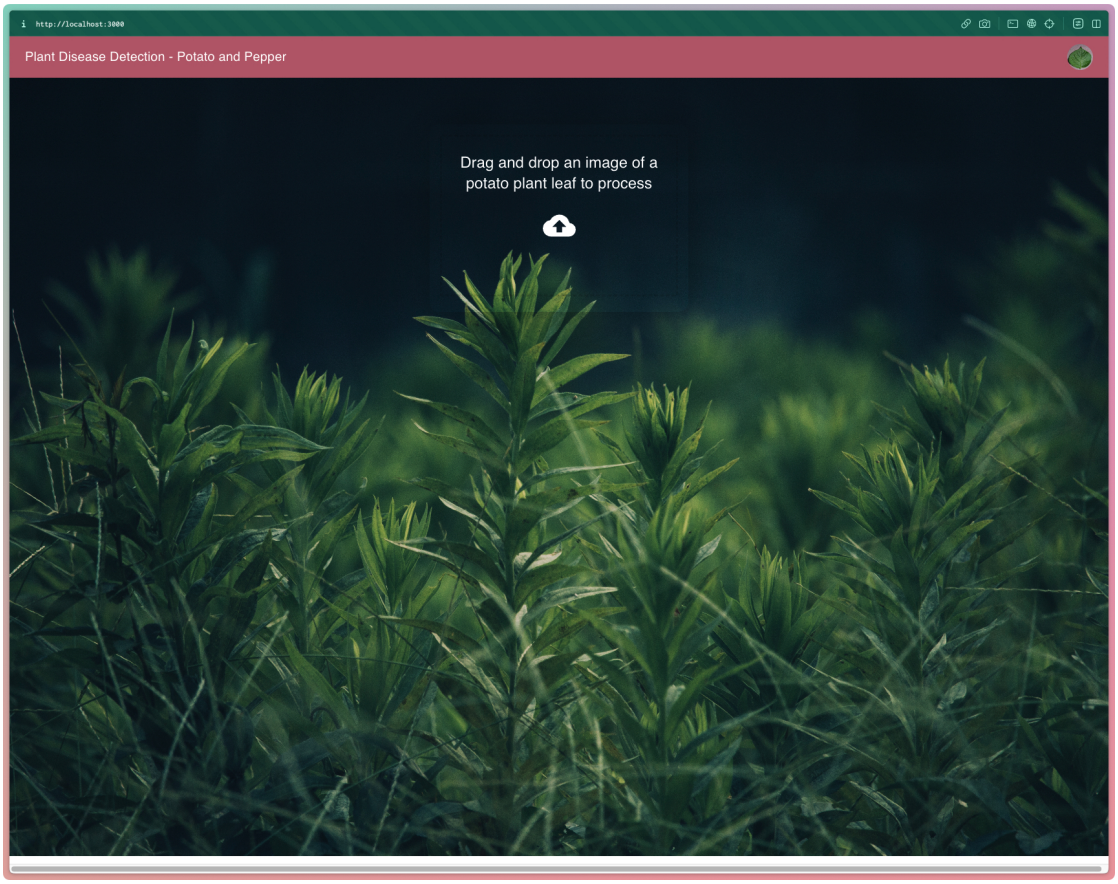

Figure 6.1:

6.2 Uploading Images

To upload an image, you have two options: Drag and drop an image file from your computer's file manager directly onto the designated upload area in the web application.

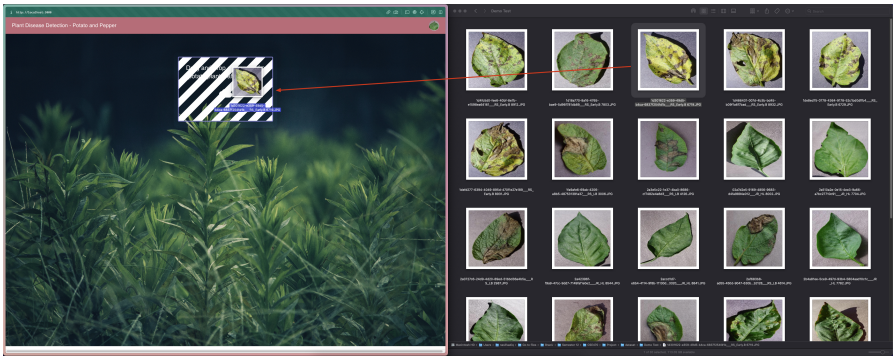

Figure 6.2:

Click on the upload area to open your computer's file manager. From there, select the image you want to predict and click "Open" or the equivalent action.

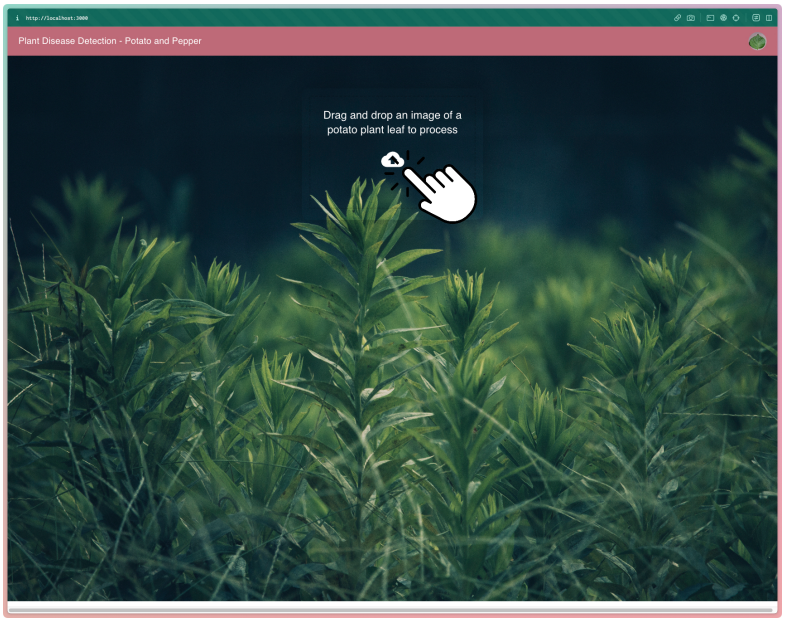

Figure 6.3:

From there, select the image you want to predict and click "Open" or the equivalent action.

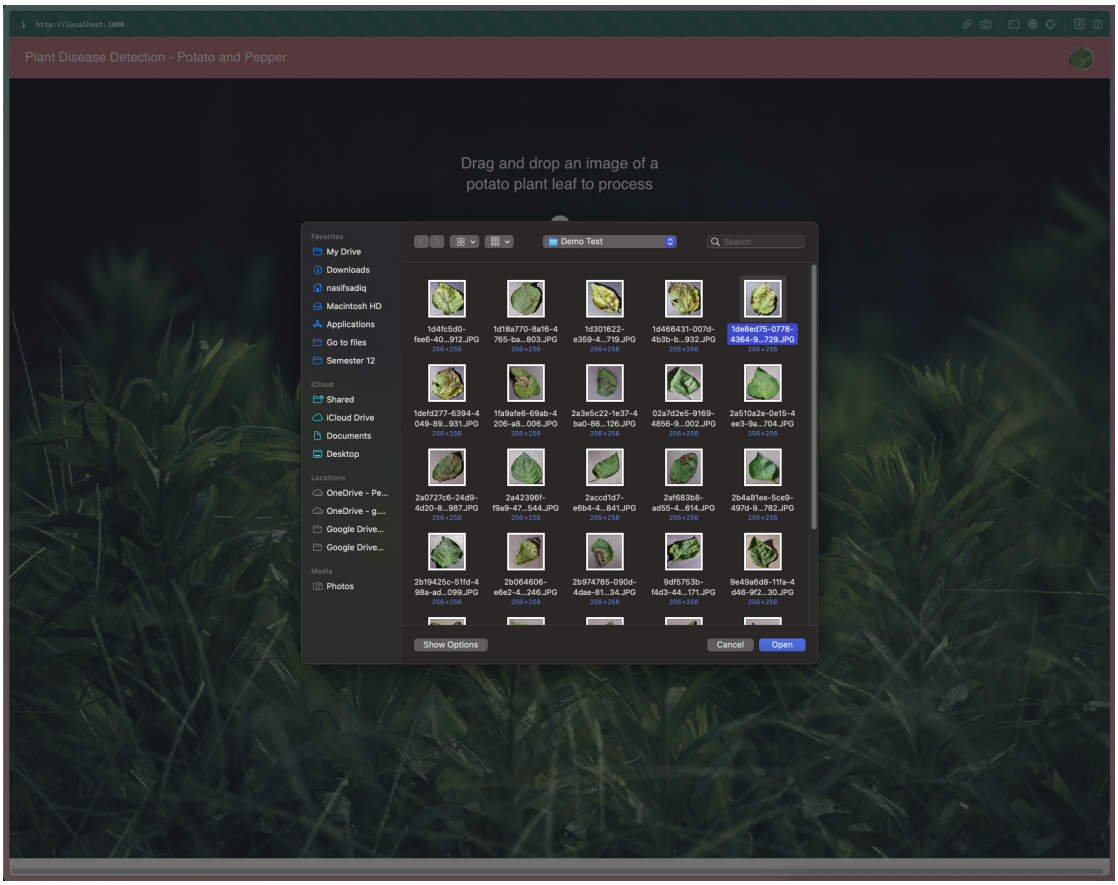

Figure 6.4:

6.3 Predicting Plant Diseases

After uploading an image, you will see a preview of the uploaded image on the application's interface. Next, The application will analyze the uploaded image and provide you with the predicted plant disease along with the confidence level of the following prediction.

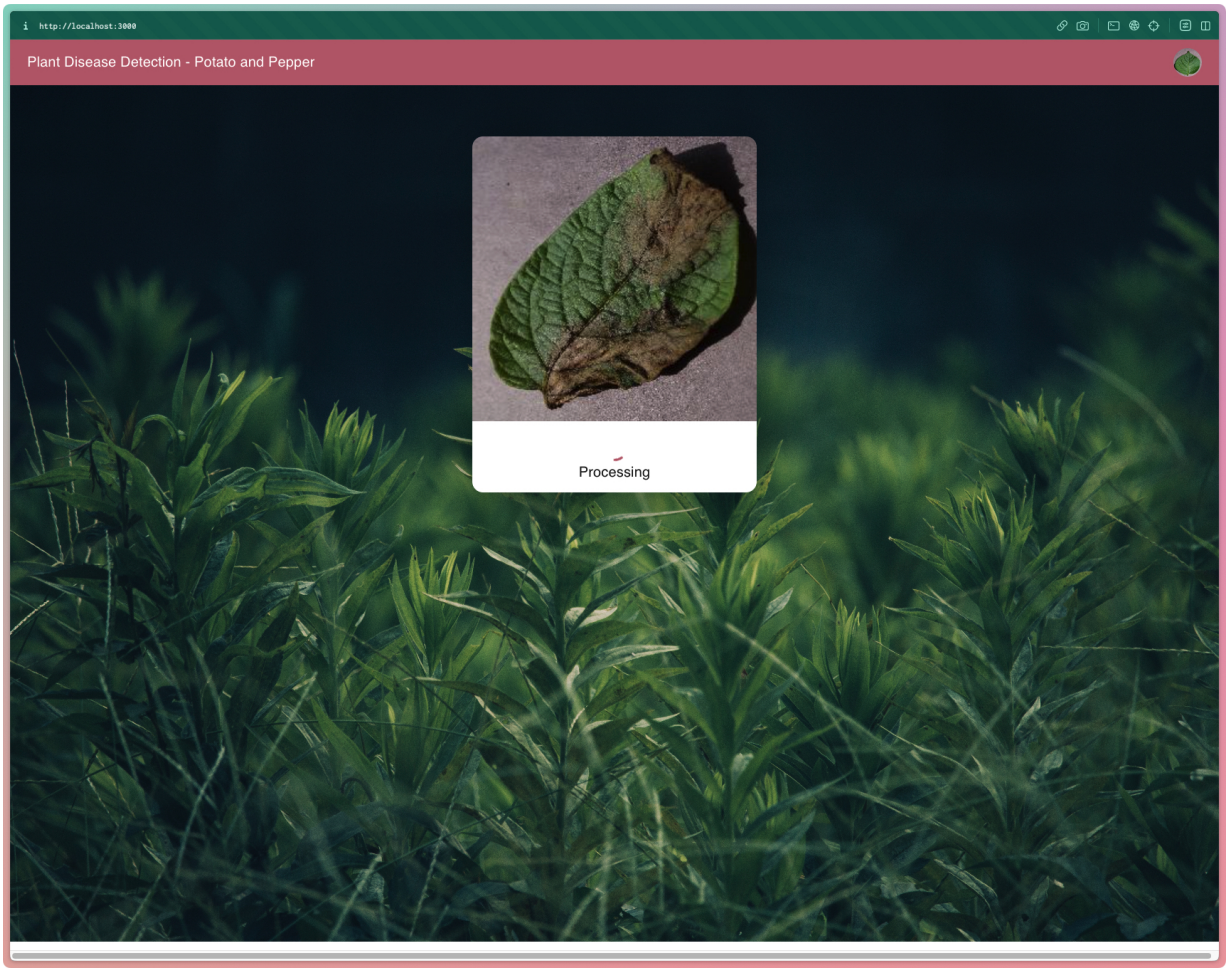

Figure 6.5:

6.4 Uploading Another Image

If you want to predict diseases for another image, you don't need to refresh the page. All you need is to clear the currently uploaded image for a fresh start. To do that, click the "Clear" button.

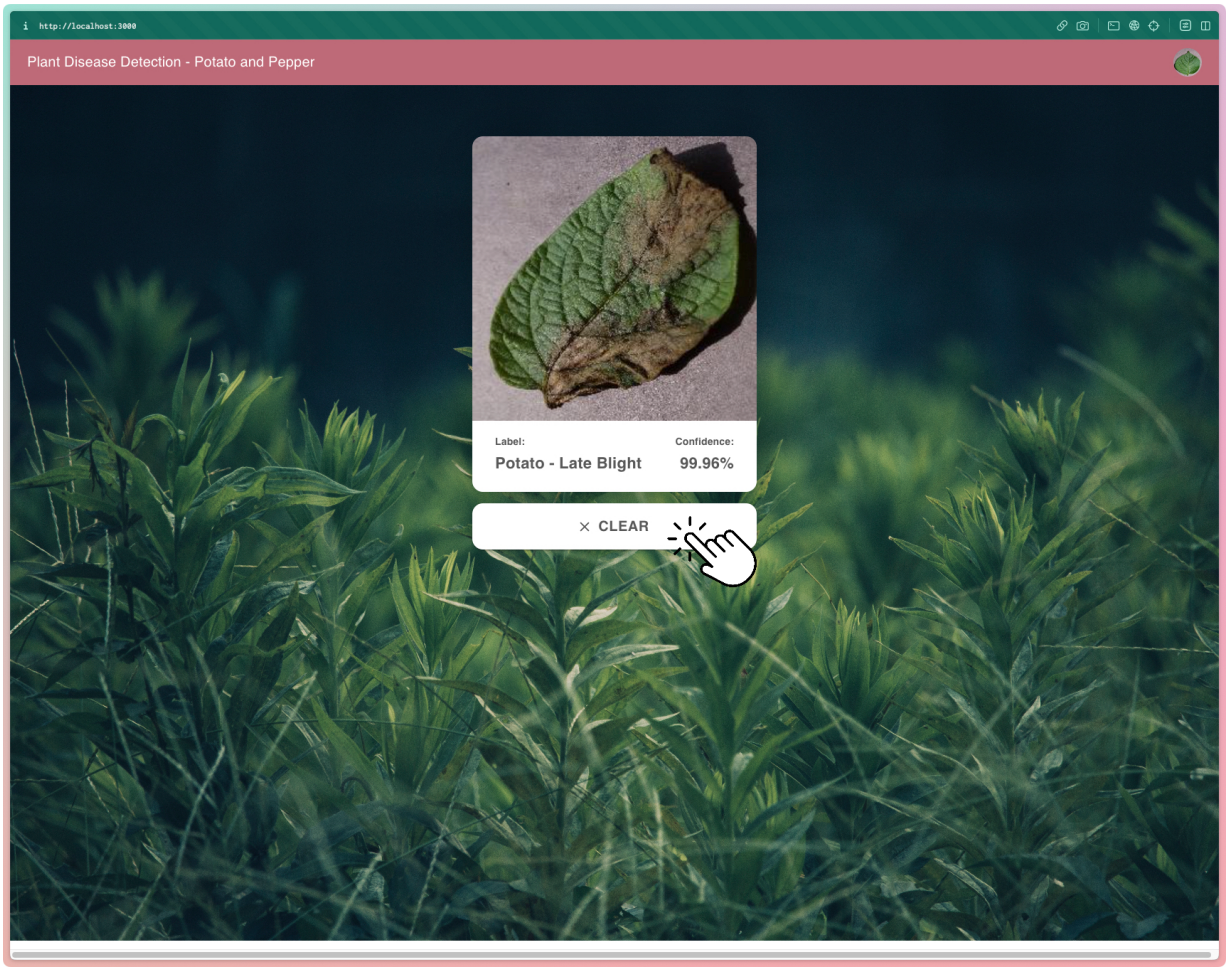

Figure 6.6:

7 Requirement Analysis:

7.1 Functional Requirements:

- Upload Images:
  - Images of Plant Leaves can be uploaded to the system
- Disease Identification:
  - Identify common diseases like blight and bacterial spots.
  - Distinguish between healthy and infected plant parts.
- Accuracy and Reliability:
  - Achieve disease identification accuracy above 90
  - Provide consistent and reliable results across different conditions.

7.2 Non Functional Requirements:

- Speed and Performance:
  - Process images and detect diseases within a few seconds.
- Robustness and Reliability:
  - Process noisy or low-quality images accurately.
- Adaptability:
  - Design modularly to accommodate new diseases and plant types.
- Usability:
  - Create an intuitive, user-friendly interface requiring minimal training.

8 Testing:

Testing the software is a critical phase in the development process to ensure that it functions correctly, meets user expectations, and is free from critical bugs or issues. Here's a detailed overview of how we tested the software:

8.1 Unit Testing:

- Backend (FastAPI): We initiated testing at the unit level, starting with the backend written in FastAPI. Unit tests focus on individual components or functions of the code. For our backend, this included testing API endpoints, data preprocessing functions, and the interaction with the deep learning model.
- Model Testing: We tested the deep learning model separately to confirm its accuracy and reliability in image classification. This involved using a diverse set of test images with known labels and checking if the model's predictions matched the expected results.

```
In [27]: 1 scores = new_model.evaluate(test_ds)

         15/15 [==============================] - 6s 283ms/step - loss: 0.0739 - accuracy: 0.9729
```

Figure 8.1: Test Accuracy

8.2 2. Integration Testing:

- a. Frontend-Backend Integration: The frontend and backend components need to work seamlessly together. We conducted integration testing to ensure that image data is transmitted correctly from the frontend to the backend through API requests and that the classification results are received and displayed accurately on the frontend.
- b. File Upload Testing: We tested the drag-and-drop and click-to-open file upload features extensively to ensure that they functioned as intended. This involved trying various file formats, sizes, and scenarios to identify any issues with file handling.

- 3. Cross-Browser and Cross-Device Testing: We tested the frontend on various web browsers (e.g., Chrome, Firefox, Safari, Edge) and different devices (e.g., desktop, mobile, tablet) to ensure compatibility and consistent performance.
- 4. Stress Testing: We used virtual machine for this as we can easily reduce the resources of the virtual device and then run our web application inside the virtual machine. We used "Parallels Desktop" application to create low configured operating system.

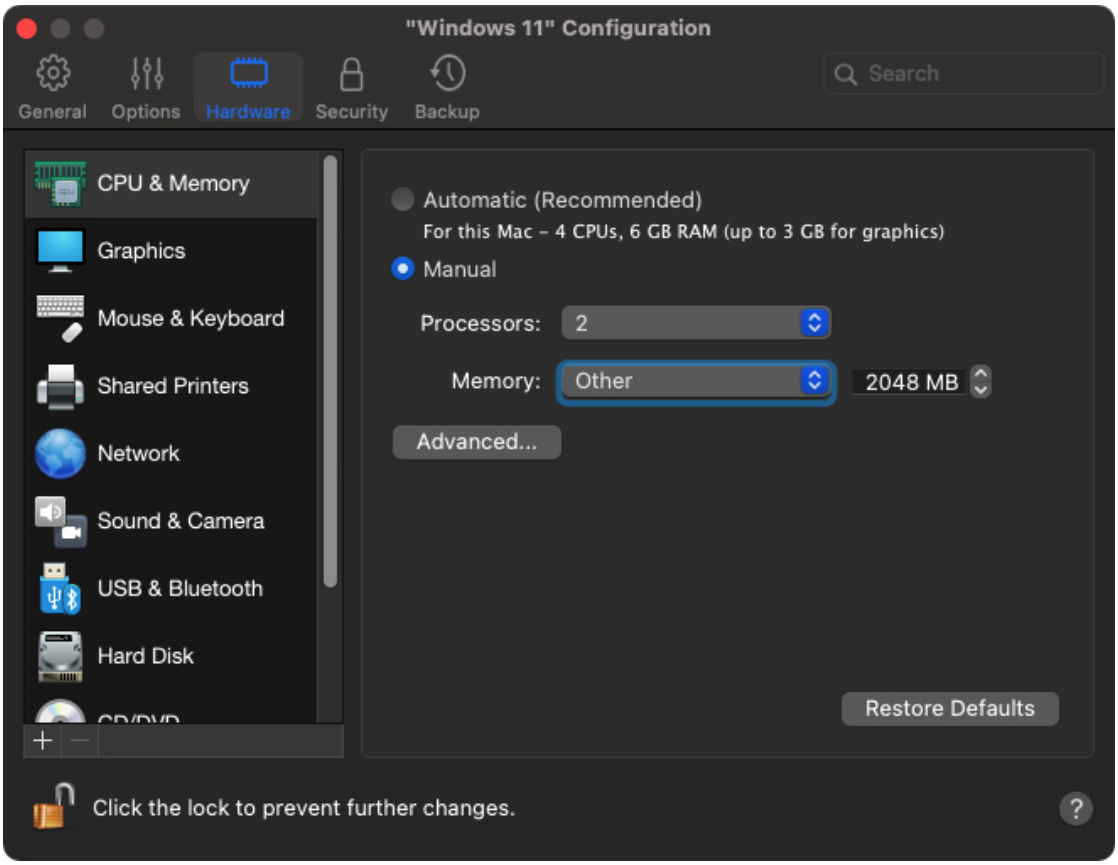

Figure 8.2:

Here is the screenshot of the lowest configuration where we were able to run our web application locally without any problem.

9 Conclusion:

In the realm of image classification, our project is a testament to the effectiveness of deep learning architectures and data augmentation techniques. Through collaboration and iterative experimentation, we've built a reliable tool capable of assisting in plant disease identification, which holds great potential for real-world agricultural applications. While our project has yielded promising outcomes, ongoing improvement and fine-tuning remain essential. As future work, we plan to explore other architectures, optimize hyperparameters, and expand the dataset for even better generalization across various scenarios. Our commitment to innovation and learning ensures that our image classification solution will continue to evolve and contribute to the field of precision agriculture. In conclusion, this project underscores the transformative potential of AI and deep learning in solving real-world challenges, exemplifying our dedication to technological advancement and impactful contributions to society.