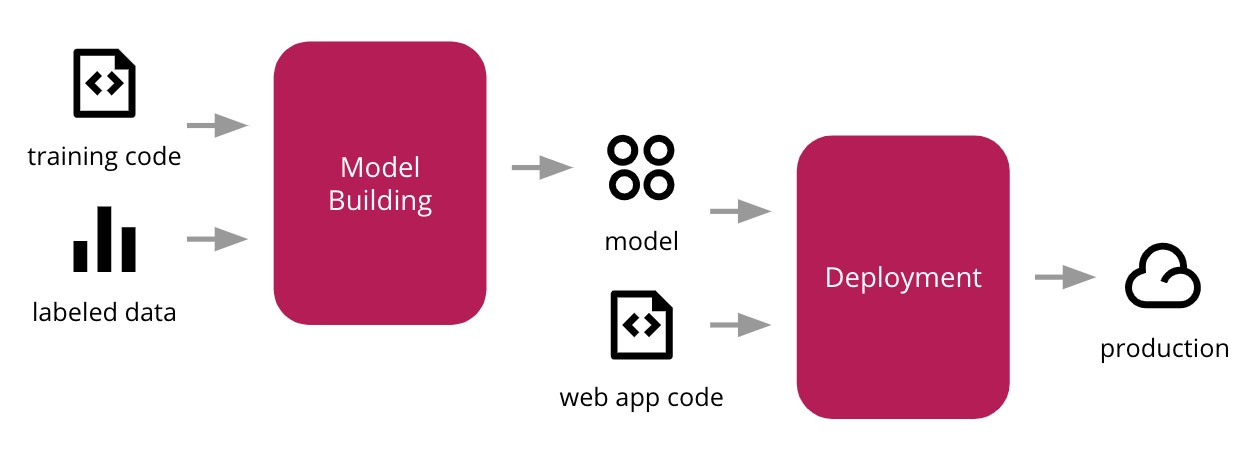
**Nail Disease Prediction**

**Introduction:**

In the healthcare domain, many diseases can be predicted by observing the color and shape of human nails. A white spot here, a rosy stain there, or some winkler projection may be an indication of disease in the body. Problems in the liver, lungs, and heart can show up in your nails. Doctors observe the nails of patients to get assistance in disease identification. Usually, pink nails indicate a healthy human. Healthy nails are smooth and consistent in color. Anything else affecting the growth and appearance of the fingernails or toenails may indicate an abnormality. A person’s nails can say a lot about their health condition. The need of such systems to analyze nails for disease prediction is because the human eye is having subjectivity about colors, having limitation of resolution and small amount of color change in a few pixels on the nail not being highlighted to human eyes which may lead to wrong result, whereas computer recognize small color changes on nails.

To overcome the above problem we are building a model which is used for the prevention and early detection of Nail Disease, Basically nail disease diagnosis depends on the different characteristics like color, shape, texture etc. Here the person can capture the images of the nail and then the image will be sent to the trained model. The model analyzes the image and detects whether the person is having nail disease or not and its type.

**Technical Architecture:**

**Prerequisites:**

**To complete this project, you must require the following software’s, concepts, and packages**

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS.Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupyter notebook and Spyder

To install Anaconda navigator and to know how to use Jupyter Notebook & Spyder using Anaconda watch the video

Link: Click here to watch the video

**1. To build Machine learning models you must require the following packages** ● **Numpy**:

o It is an open-source numerical Python library. It contains a multidimensional array and matrix data structures and can be used to perform mathematical operations ● **Scikit-learn:**

o It is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbors, and it also supports Python numerical and scientific libraries like NumPy and SciPy

**● Streamlit:**

Web framework used for building Web applications

**● Python packages:**

o open anaconda prompt as administrator

o Type “pip install numpy” and click enter.

o Type “pip install pandas” and click enter.

o Type “pip install scikit-learn” and click enter.

o Type “pip install tensorflow==2.3.2” and click enter.

o Type “pip install keras==2.3.1” and click enter.

o Type “pip install streamlit” and click enter.

**● Deep Learning Concepts**

o **VGG16:** VGG16 is a transfer learning method. A pre-trained model trained on 1000 classes of images. VGG basic

o **Streamlit:** Streamlit is an open-source app framework for Machine Learning and Data Science teamsfor building web applications

If you are using Pycharm IDE, you can install the packages through the command prompt and follow the same syntax as above.

**Project Objectives:**

By the end of this project you will:

● Know fundamental concepts and techniques of Convolutional Neural Network. ● Gain a broad understanding of image data.

● Know how to pre-process/clean the data using different data preprocessing techniques.

● know how to build a web application using the Streamlit framework.

**Project Flow:**

● The user interacts with the UI (User Interface) to choose the image. ● The chosen image analyzed by the model which is integrated with streamlit application. ● VGG16 Model analyzes the image, then prediction is showcased on the Streamlit UI.

To accomplish this, we have to complete all the activities and tasks listed below o Data Collection.

o Create Train and Test Folders.

o Model Building

o Importing the Model Building Libraries

o Loading the model

o Adding Flatten Layers

o Adding Output Layer

o Creating a Model object:

o Configure the Learning Process

o Import the ImageDataGenerator library

o Configure ImageDataGenerator class

o Apply ImageDataGenerator functionality to Train set and Test set

o Training

o Train the Model

o Save the Model

o Testing

o Test the model

o Application Building

o Create a Streamlit UI

o Build Python Code

o Run the application

o Final Output

**Project Structure:**

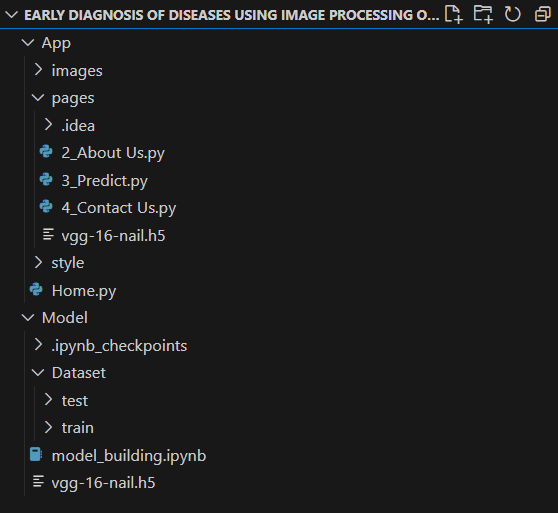
Create a Project folder which contains files as shown below

● The Dataset folder contains the training and testing images for training our model. ● We are building a Streamlit Application that needs multiple pages stored in the **pages** folder and a python script **Home.py** for server side scripting

**●** we need the model which is saved and the saved model in this content is a **vgg-16-nail.h5**

● pages folder contains 2\_About Us.py, 3\_Predict.py, 4\_Contact Us.py, and our model vgg-16-nail.h5. ● An IPYNB file is a notebook document created by Jupyter Notebook.

**model\_building.ipynb**

****

**Milestone 1: Data Collection**

You can download the dataset used in this project using the below link

Dataset:

https://drive.google.com/drive/folders/1AXTYsbiarS1TCAgfj0mancTSrJYYMWMs?usp=sharin g

**Note: For better accuracy, train on more images, which can be downloaded from google and place it in respective folders.**

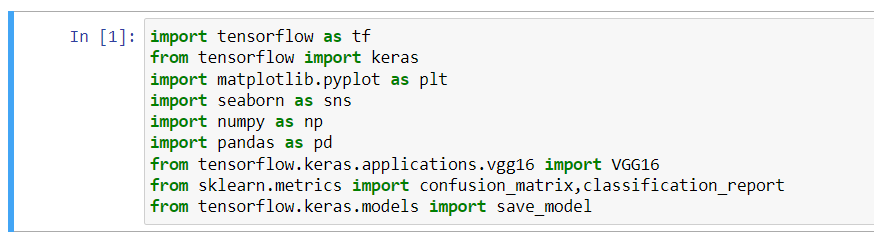
**Milestone 2: Model Building**

Now it's time to Build input and output layers for VGG16 model

Hidden layers freeze because they have trained sequence, so changing the input and output layers.

**Activity 1: Importing the Model Building Libraries**

Importing the necessary libraries

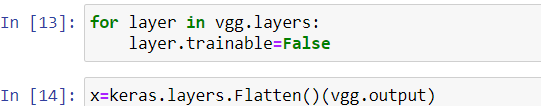


**Activity 2: Loading the model**

****The vgg16 model need to be loaded and we are storing that into a variable called vgg

**Activity 3: Adding Flatten Layers**

For VGG16 model, we need to keep the Hidden layer training as false, because it has trained weights



**Activity 4: Adding Output Layer**

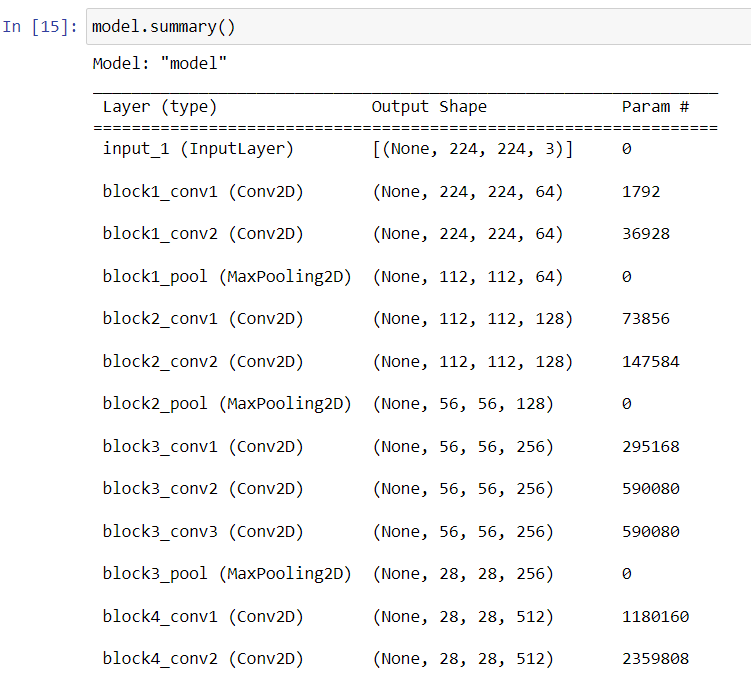
Our dataset has 17 classes, so the output layer need to be changed as per the dataset 

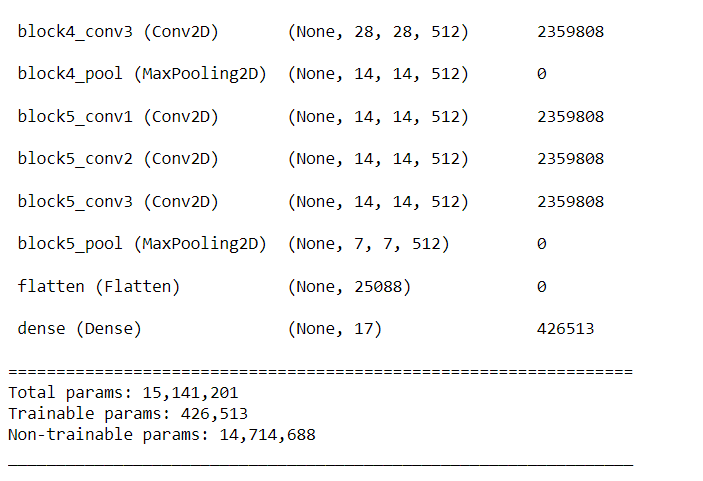
17 indicates no of classes, softmax is the activation function we use for categorical output Adding fully connected layer

**Activity 5: Creating a Model object:**

****

We have created inputs and outputs in the previous steps and we are creating a model and fitting to the vgg16 model, so that it will take inputs as per the given and displays the given no of classes





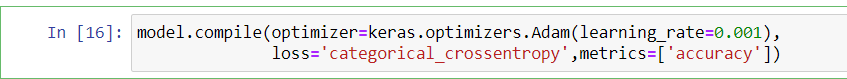
The vgg model has a total of 15,141,201 parameters, out of which 426,513 are trainable and the remaining are freeze.

**Activity 6: Configure the Learning Process**

● The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires a loss function during the model compilation process.

● Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using Adam optimizer

● Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process



**Activity 7: Import the ImageDataGenerator library**

In this we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although perform some geometric transformations of images like rotation, scaling, translation, etc.

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class.

Let us import the ImageDataGenerator class from tensorflow Keras



**Activity 8**: **Configure ImageDataGenerator class**

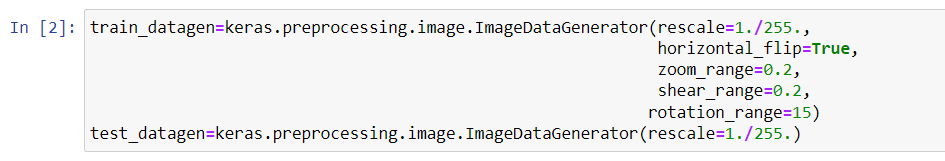
ImageDataGenerator class is instantiated and the configuration for the types of data augmentation

There are five main types of data augmentation techniques for image data; specifically: Image shifts via the width\_shift\_range and height\_shift\_range arguments. The image flips via the horizontal\_flip and vertical\_flip arguments.

Image rotations via the rotation\_range argument

Image brightness via the brightness\_range argument.

Image zoom via the zoom\_range argument.

An instance of the ImageDataGenerator class can be constructed for training and test. **Activity 9: Apply ImageDataGenerator functionality to Trainset and Testset**

Let us apply ImageDataGenerator functionality to Trainset and Testset by using the following code. For Training set using flow\_from\_directory function.

This function will return batches of images from the subdirectories

Arguments:

● Directory: Directory where the data is located. If labels are "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.

● batch\_size: Size of the batches of data which is 10.

● target\_size: Size to resize images after they are read from disk.

● class\_mode:

- ‘int': means that the labels are encoded as integers (e.g. for

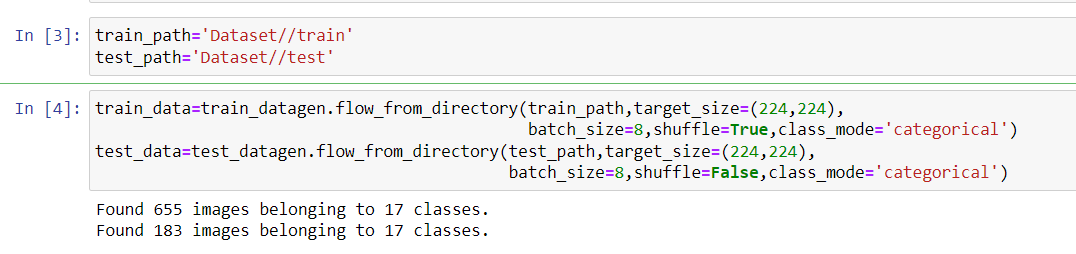
sparse\_categorical\_crossentropy loss).

- 'categorical' means that the labels are encoded as a categorical vector (e.g. for categorical\_crossentropy loss).

- 'binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g. for binary\_crossentropy).

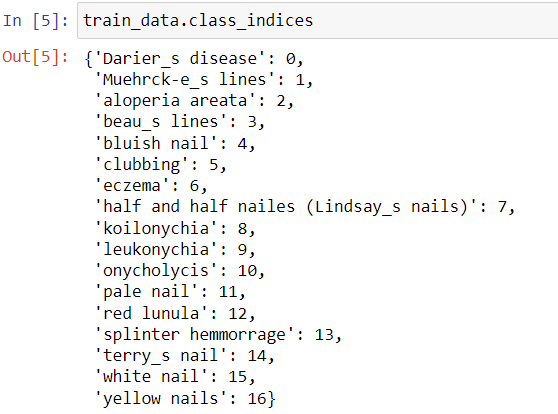
- None (no labels).

Loading our data and performing Data Augmentation



We notice that 655 images belong to 17 classes for training and 183 images belonging to 17 classes for testing purposes.

List of classes we have



**Milestone 3: Training**

**Activity 1: Train the Model**

Now, let us train our model with our image dataset. The model is trained for 30 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch till 30 epochs and probably there is further scope to improve the model.

**Fit\_generator** functions used to train a deep learning neural network

**Arguments:**

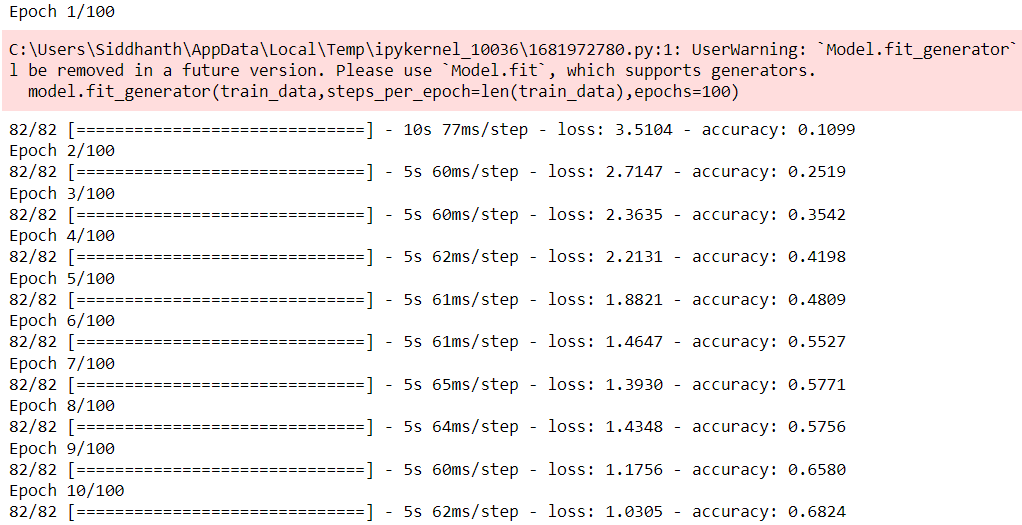
● steps\_per\_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps\_per\_epoch as the total number of samples in your dataset divided by the batch size.

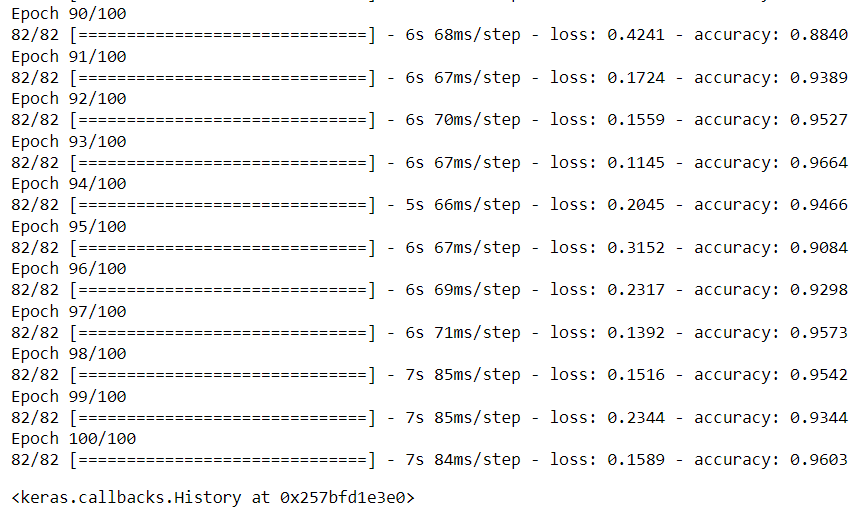
● Epochs: an integer and number of epochs we want to train our model for.



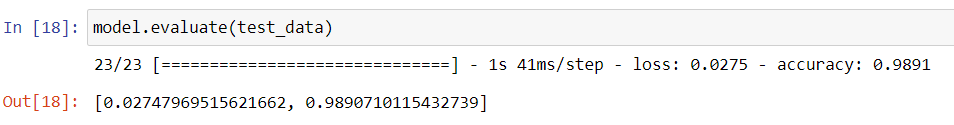
Note: for model, we need to fit the data

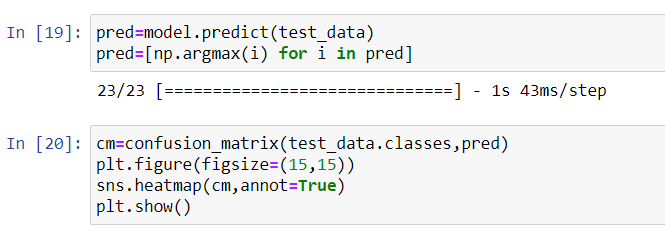
Accuracy of the model after 100 epochs.

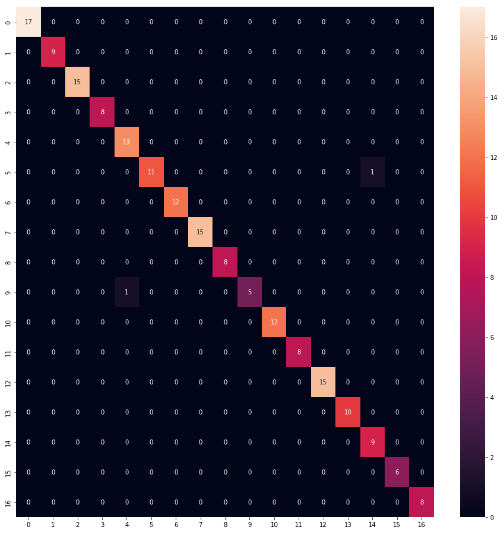


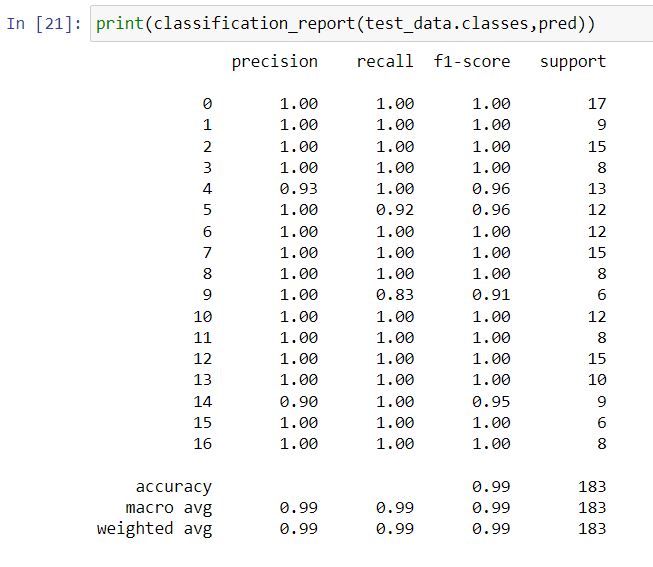


Testing model performance on Testset





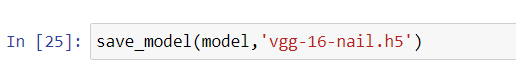




**Activity 2: Save the Model**

The model is saved with .h5 extension as follows

An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

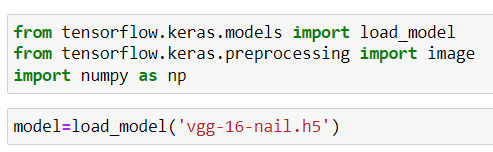


**Milestone 4: Testing**

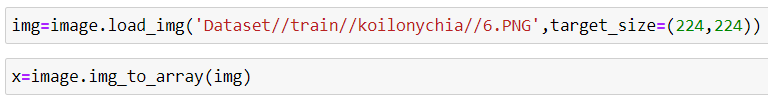
**Activity 1: Test the model**

Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data.

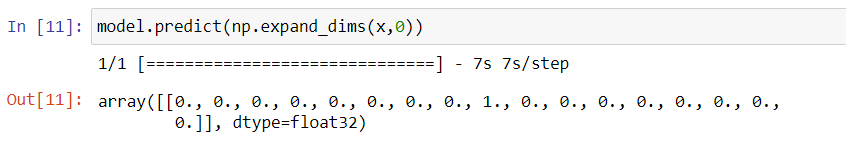
Load the saved model using load\_model



Taking an image as input and checking the results



**Note: We used VGG16, for image is 224,224, so change according to the model** By using the model we are predicting the output for the given input image



The predicted class index name will be printed here.



**Milestone 5: Application Building**

Now that we have trained our model, let us build our Streamlit application which will be running in our local browser with a user interface.

In the Streamlit application, there are multiple pages, the image is taken on the predict page and given to model and predicts the output and give simple recommendations.

**Activity 1: Create Streamlit pages**

**o** We use Streamlit to create the front end part of the web page.

**o** Here, we have created 4 Streamlit pages- Home.py, 2\_About Us.py, 3\_Predict.py and 4\_Contact Us.py

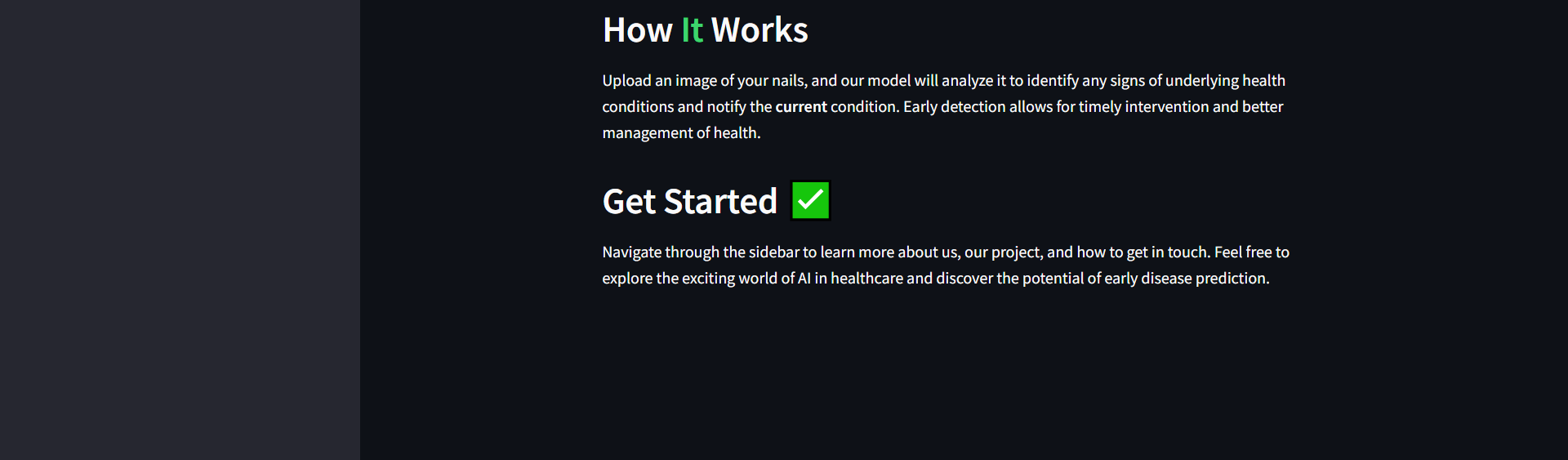
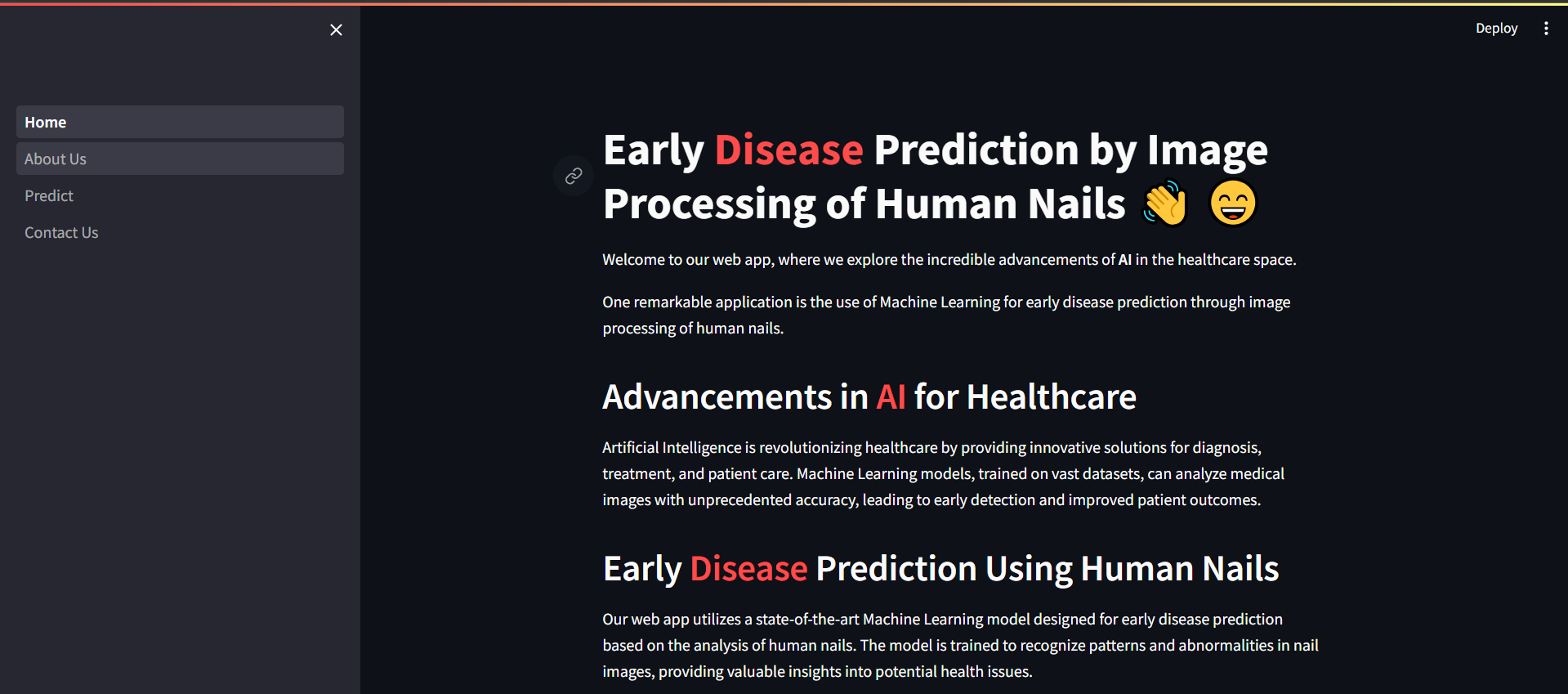
**o** Home.py displays the home page.

**o** 2\_About Us.py display the about us and our socials

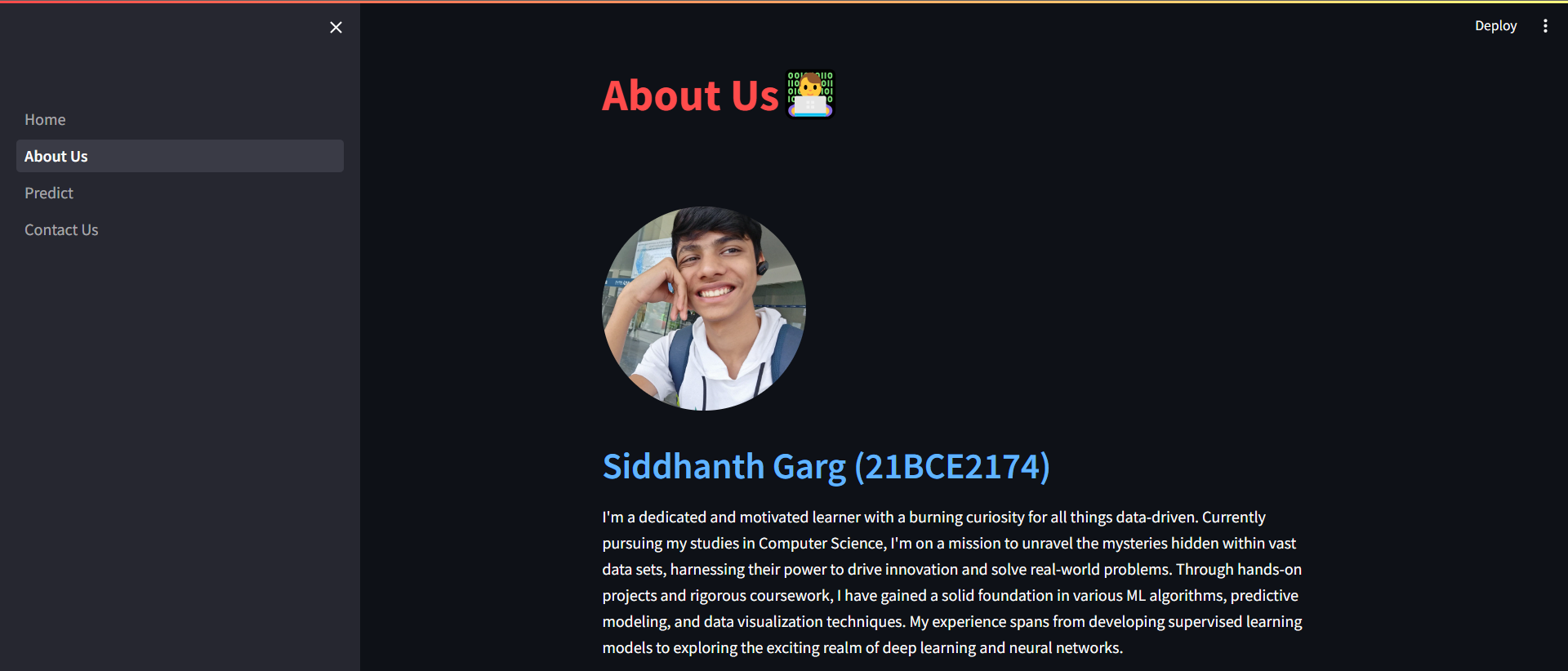
**o** 3\_Predict.py takes the input image and displays the prediction.

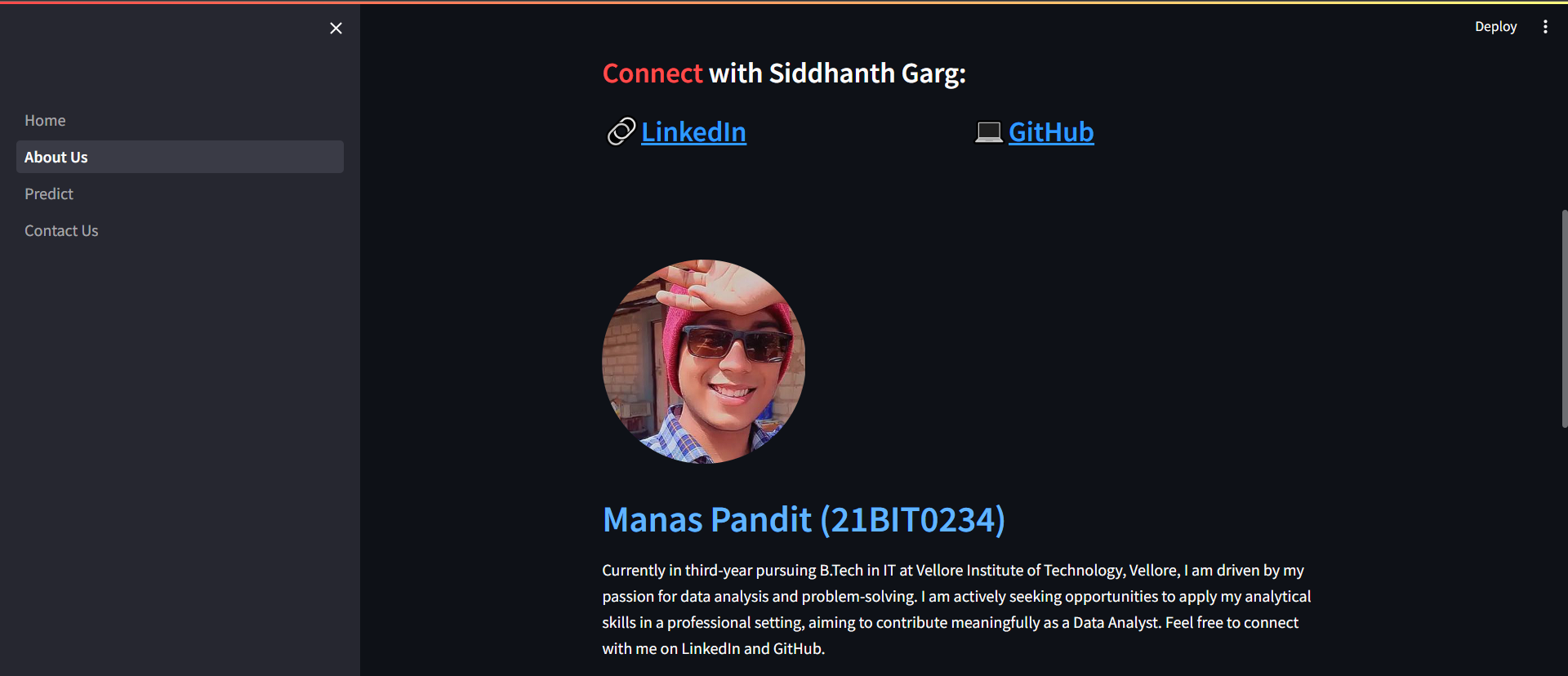
**o** 4\_Contact Us.py is a form to provide feedback.

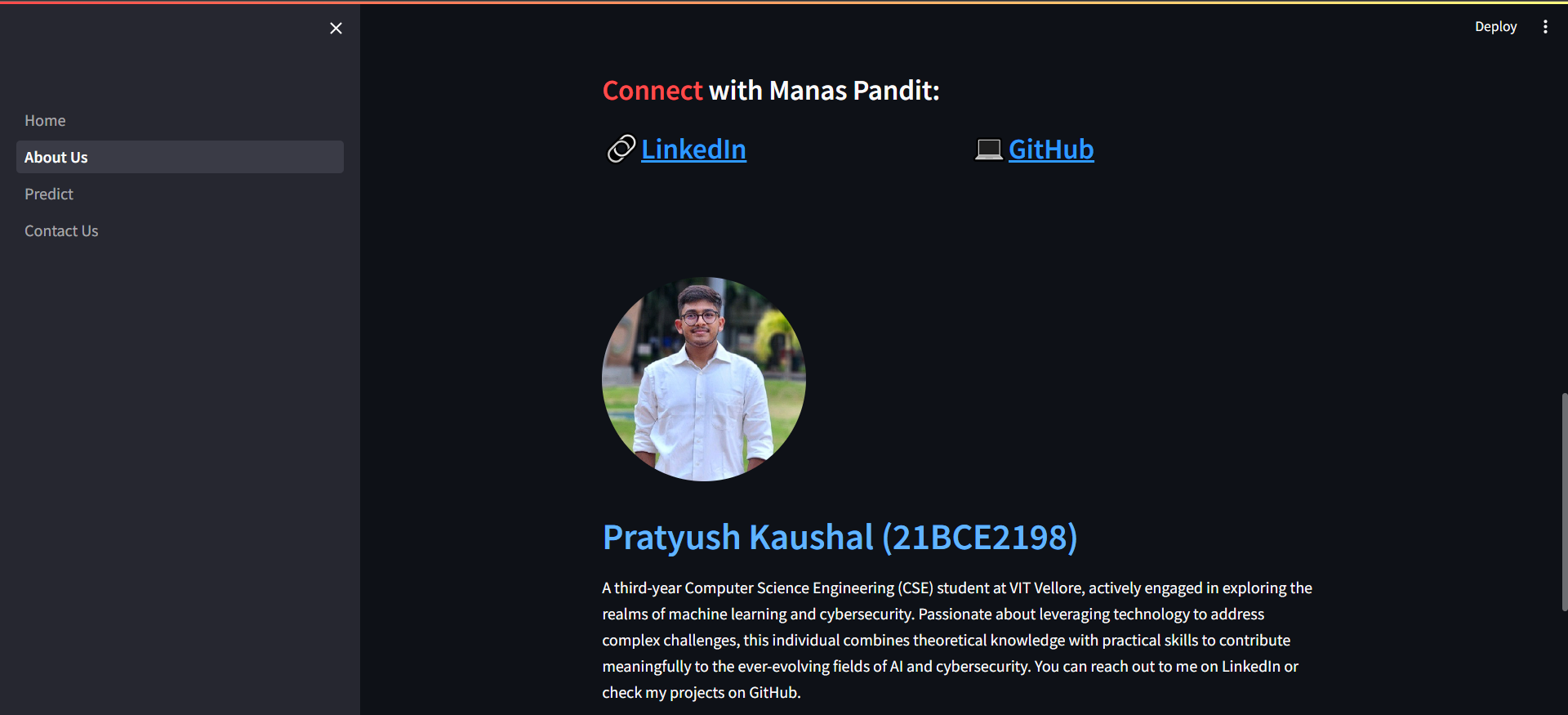
**Home page:-**

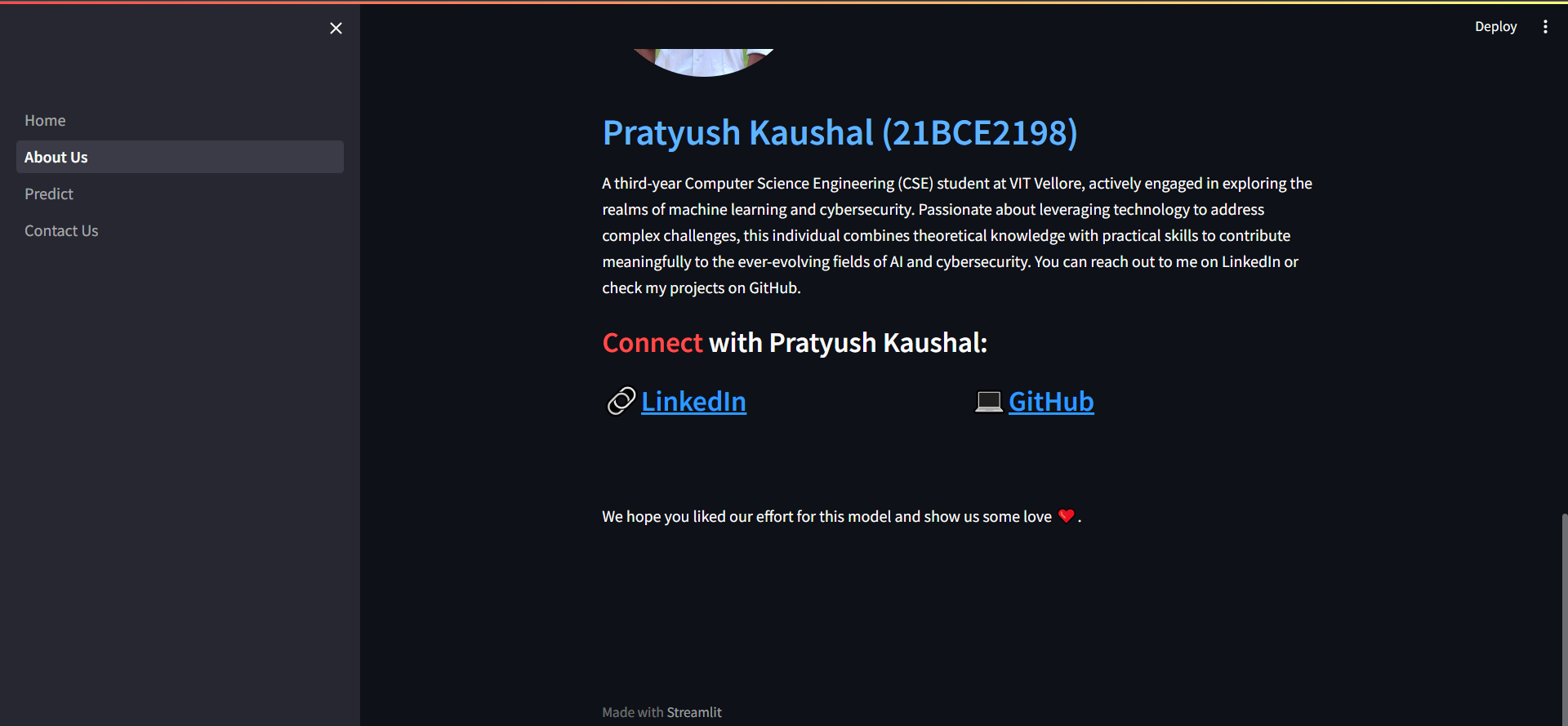
****

**About page:-**

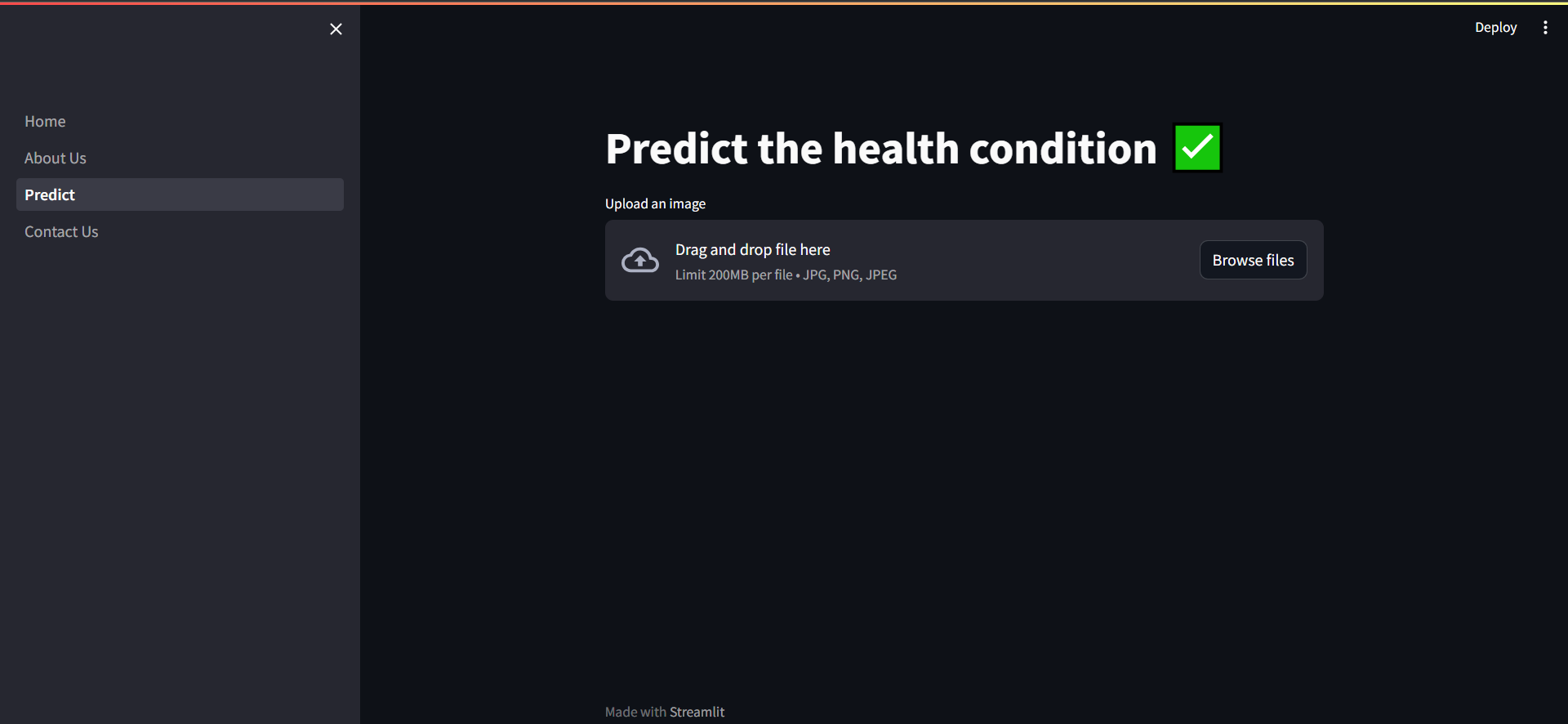
****

****

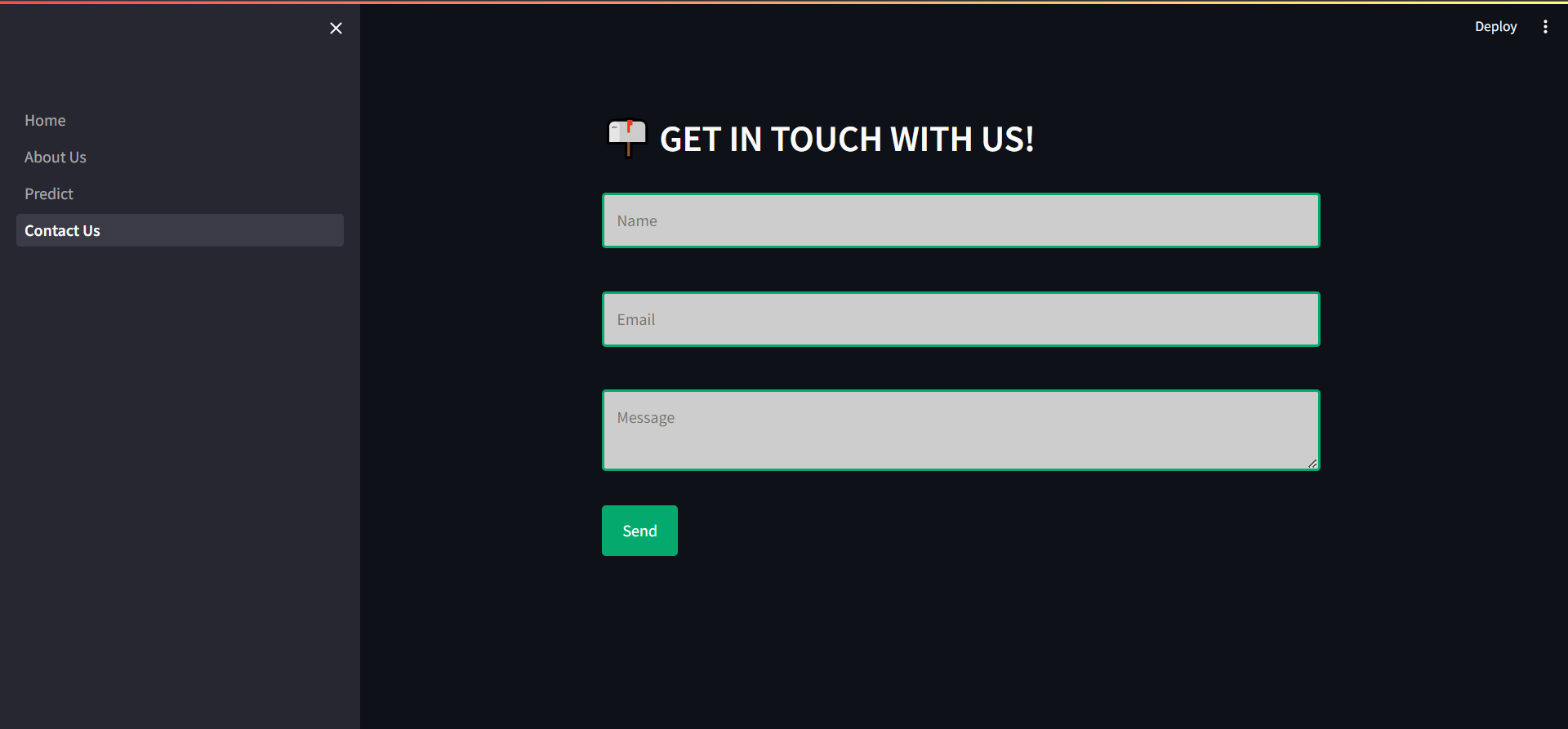
****

****

**Nail Disease Prediction Page:-**

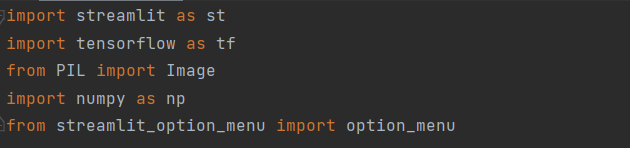
****

**Contact us page:-**

****

**Activity 2: Build python code**

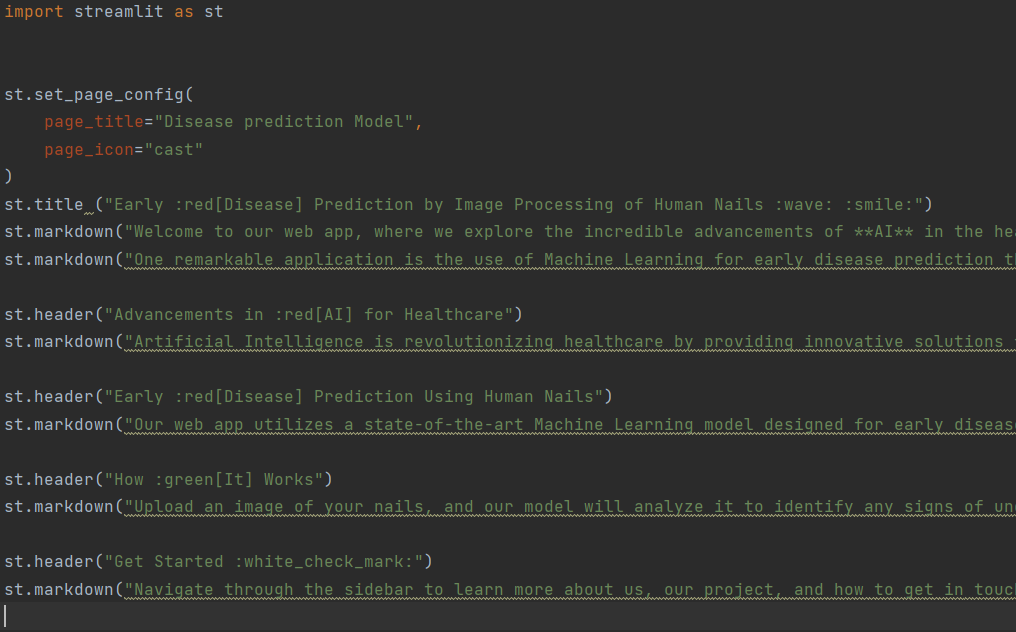
**Task 1: Importing Libraries**

The first step is usually importing the libraries that will be needed in the program. 

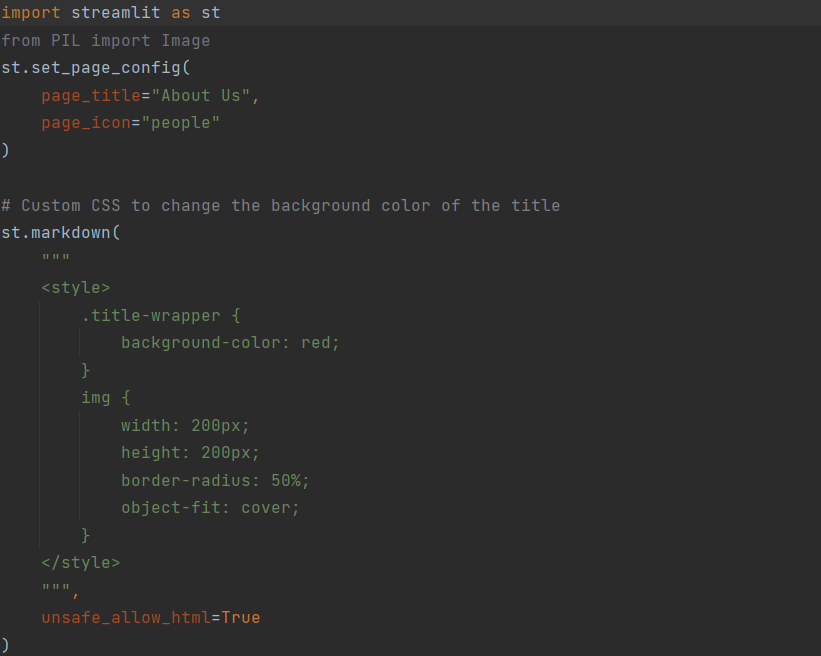
Importing the streamlit module in the project is mandatory. Once after loading the libraries, we need to load the model

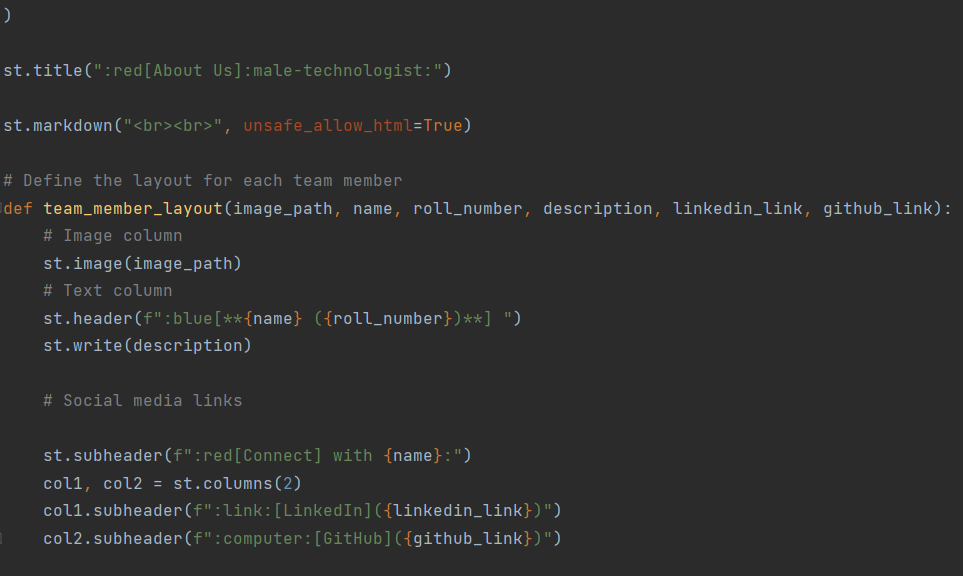
**Task 2: Creating our Streamlit application and loading our model by using load\_model method **

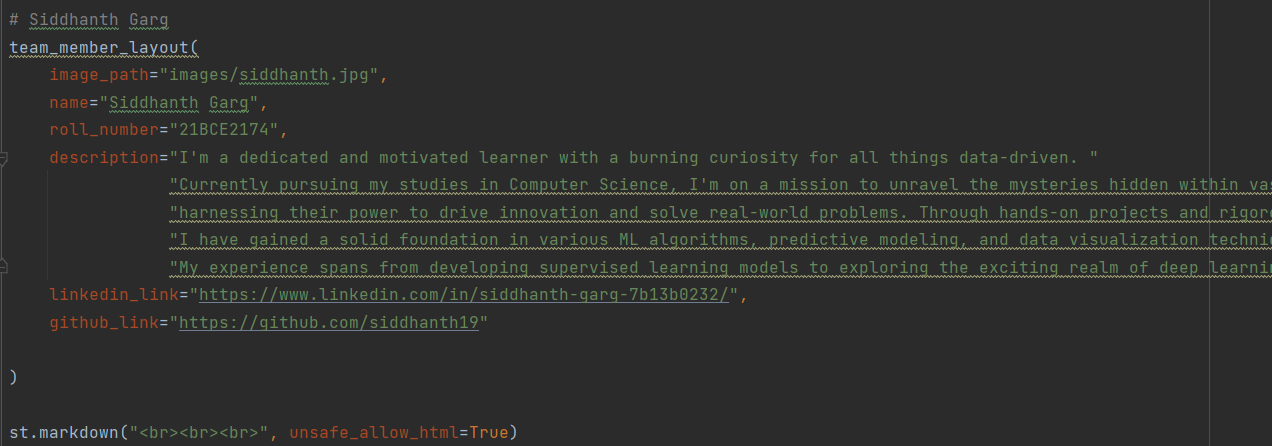
**Default home page**

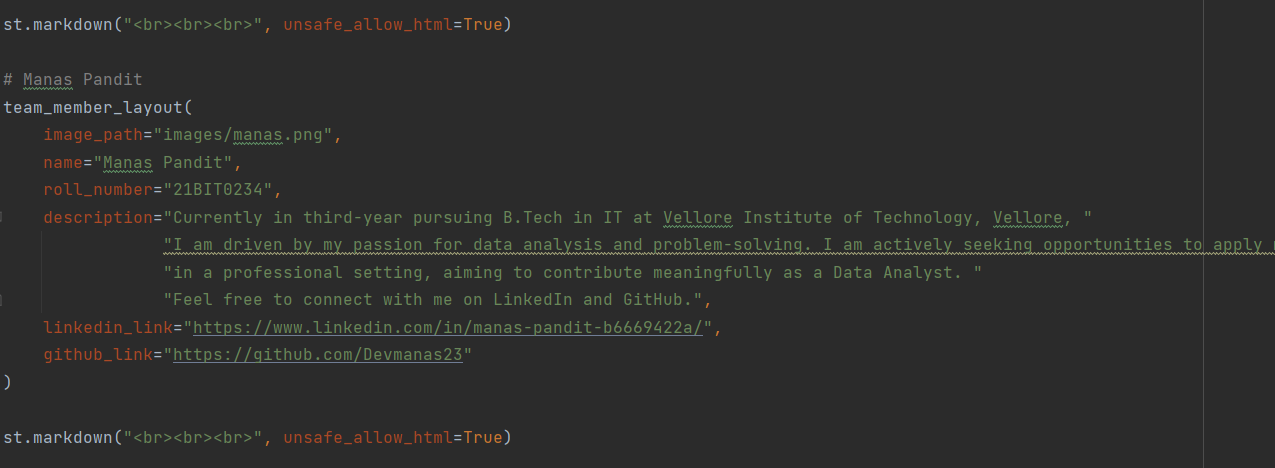
****

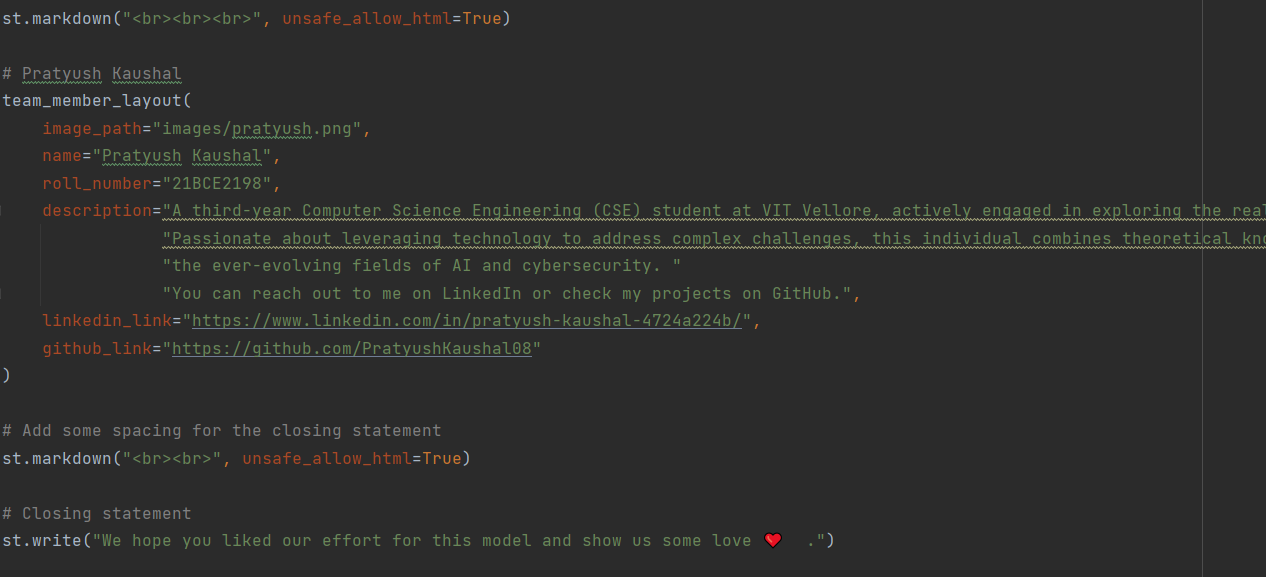
When you click on about us on sidebar, it will redirect you to the about us page



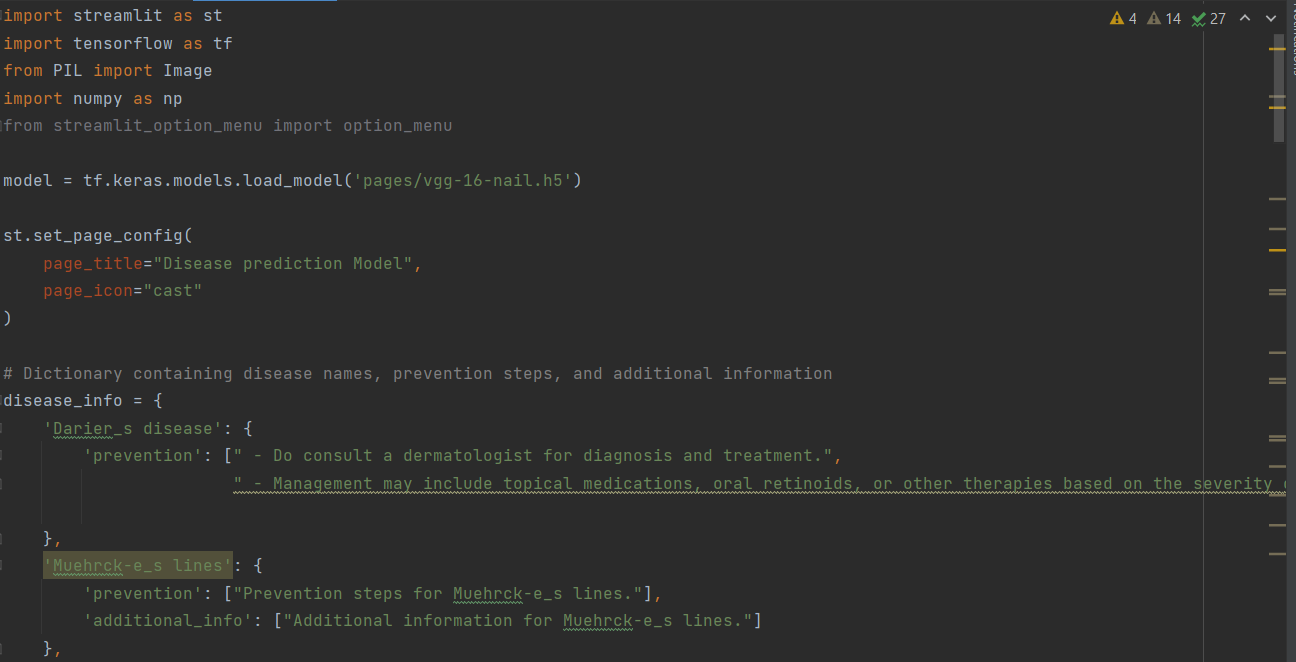


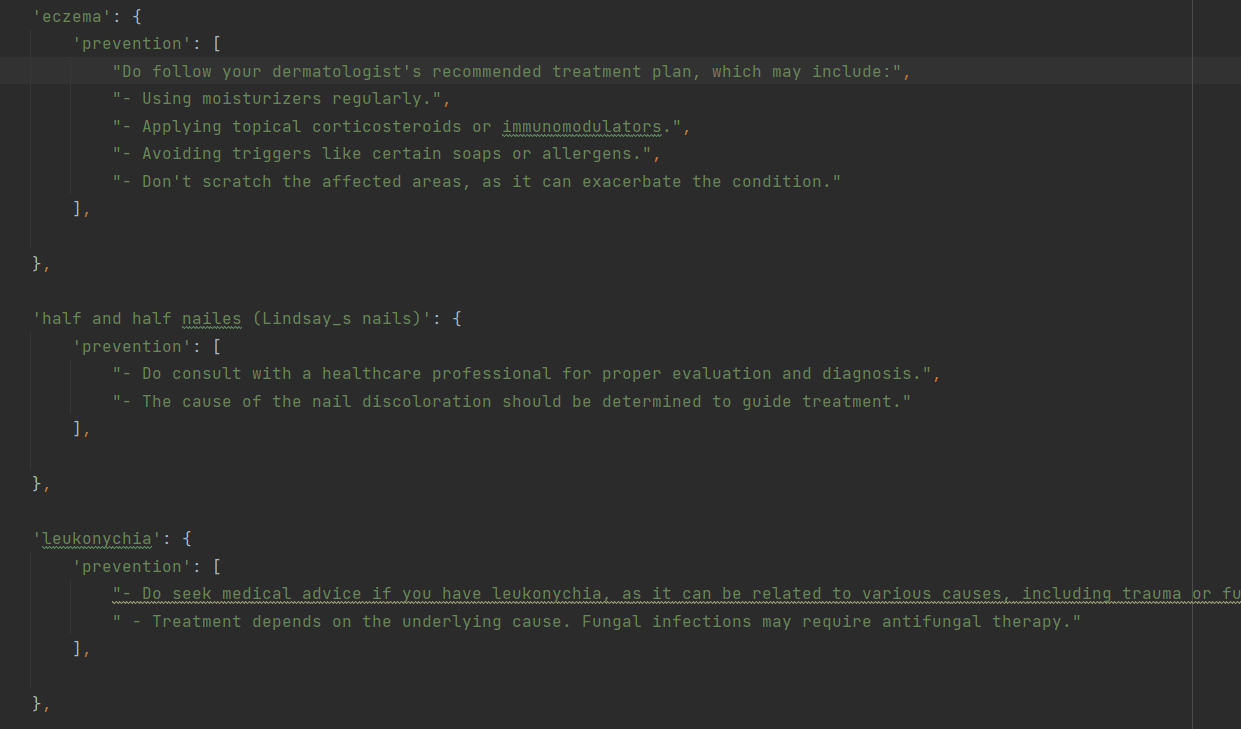


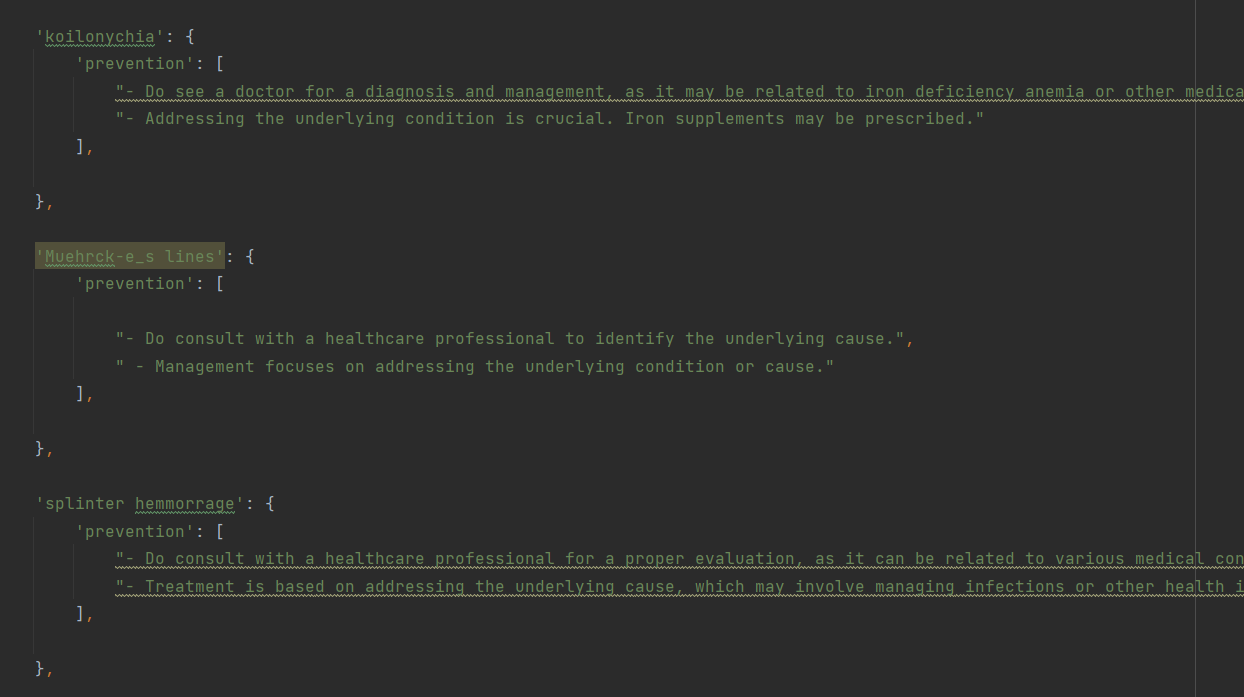


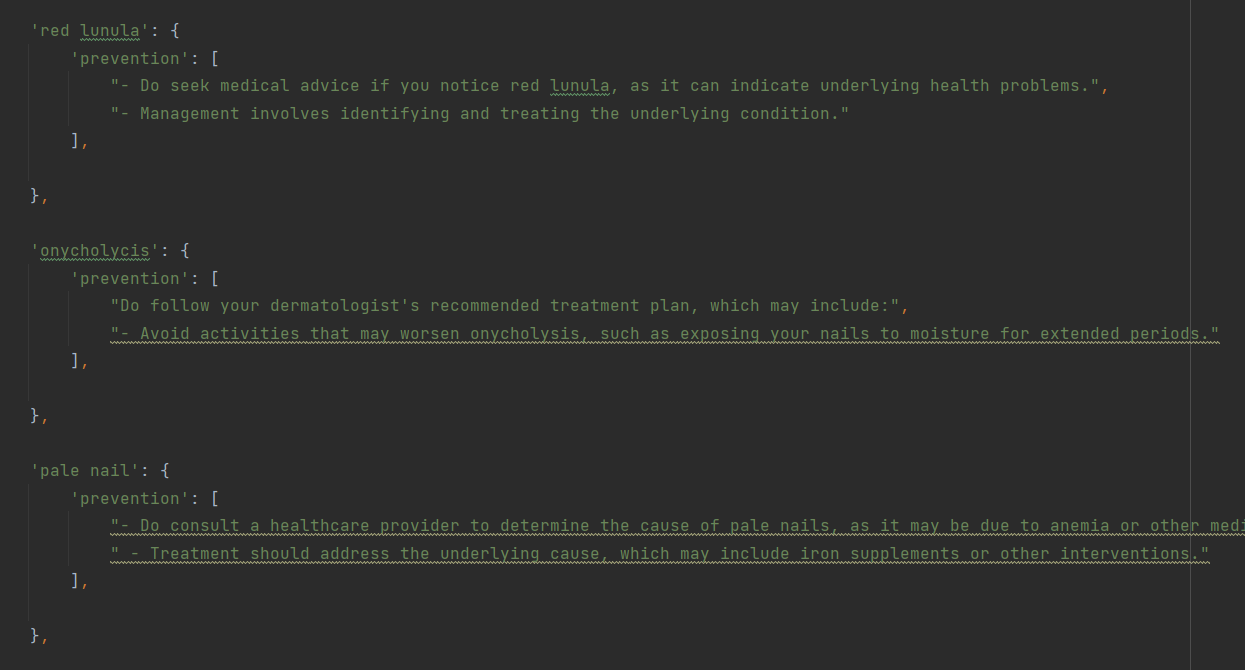


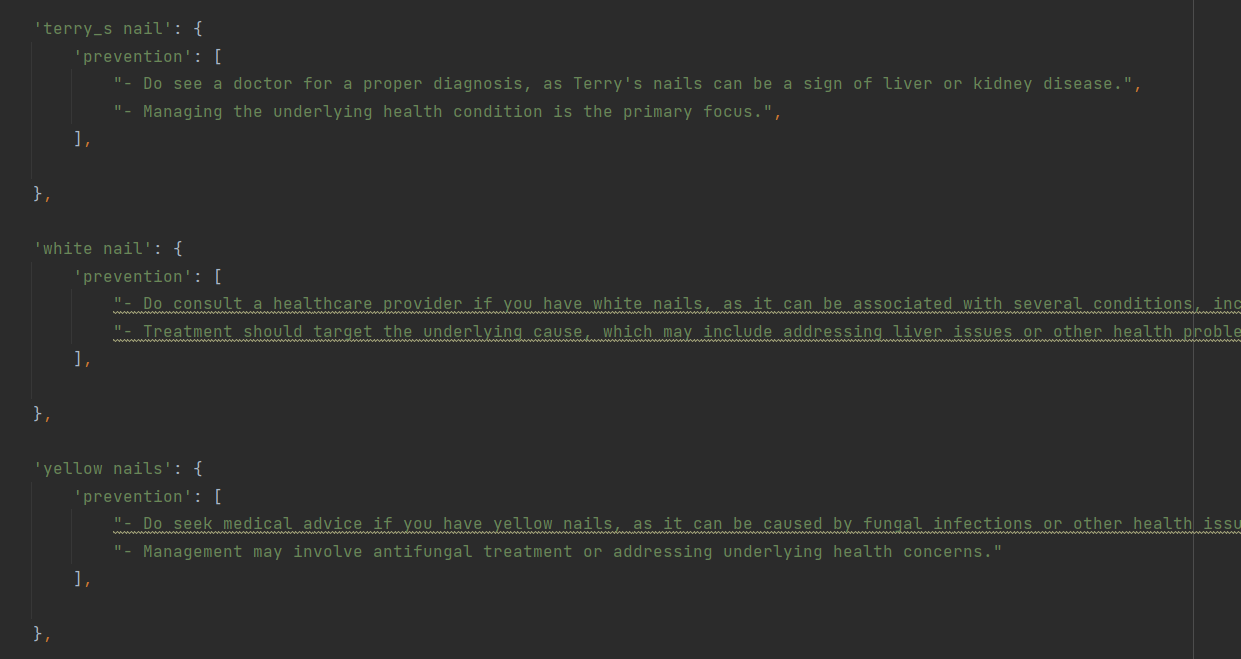
When you click on predict , it will redirect you to the Nail Disease Predict page

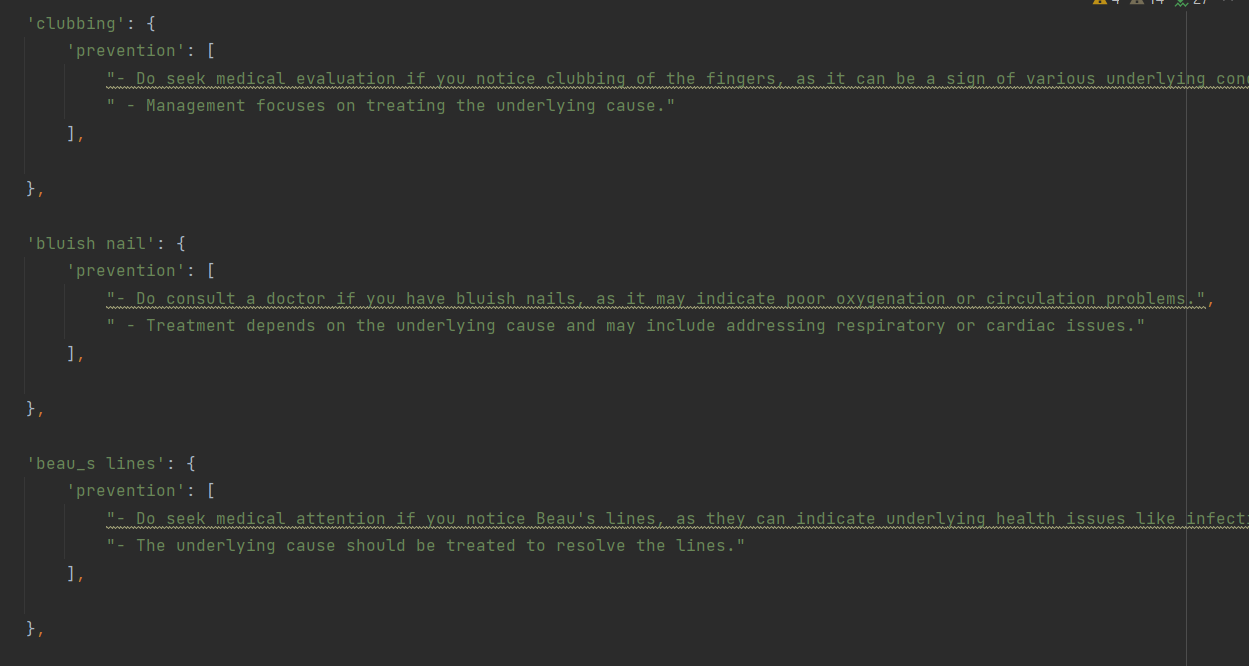




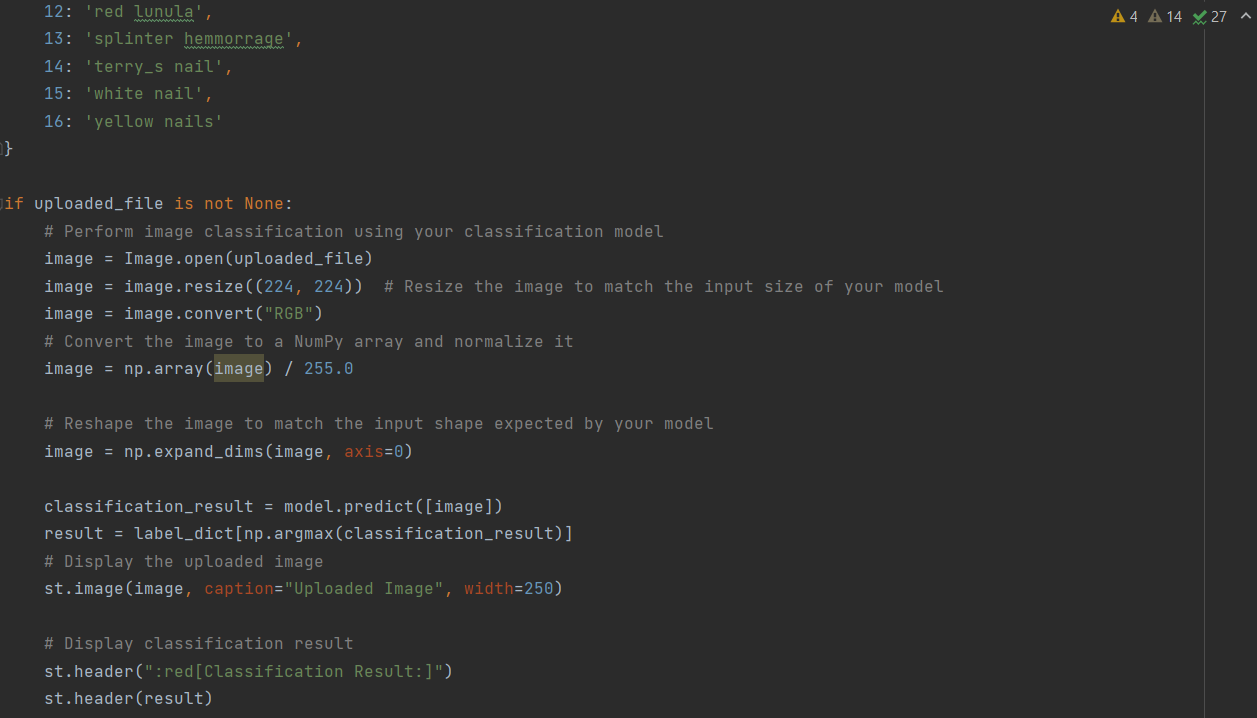


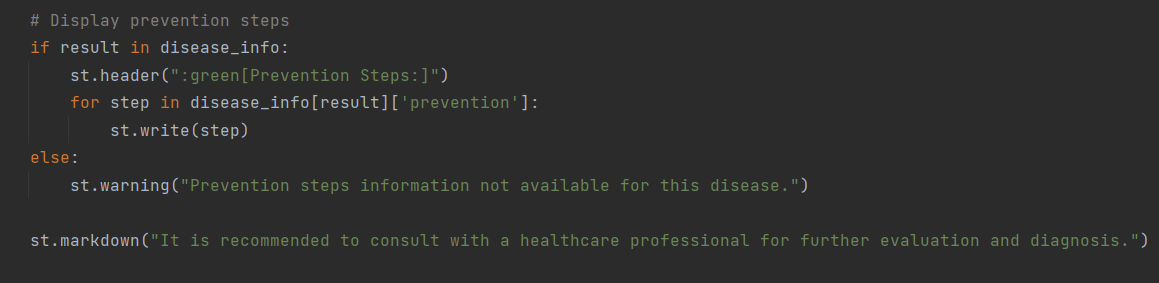




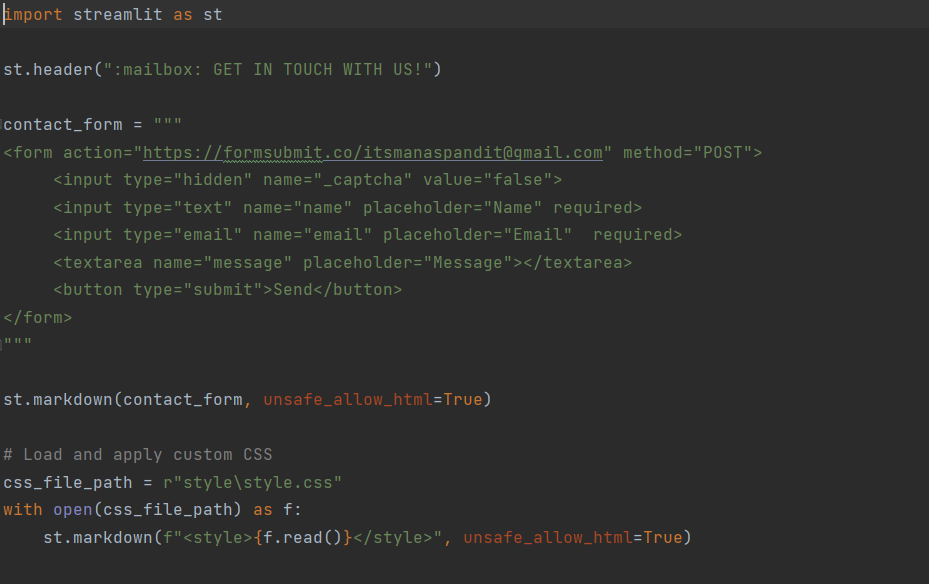




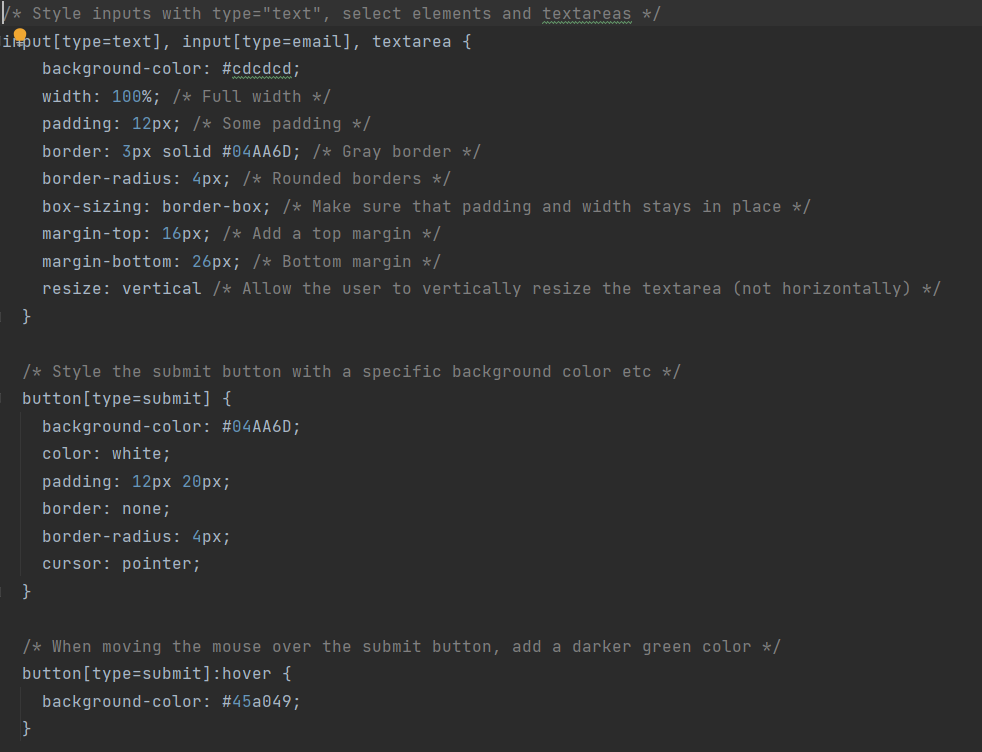




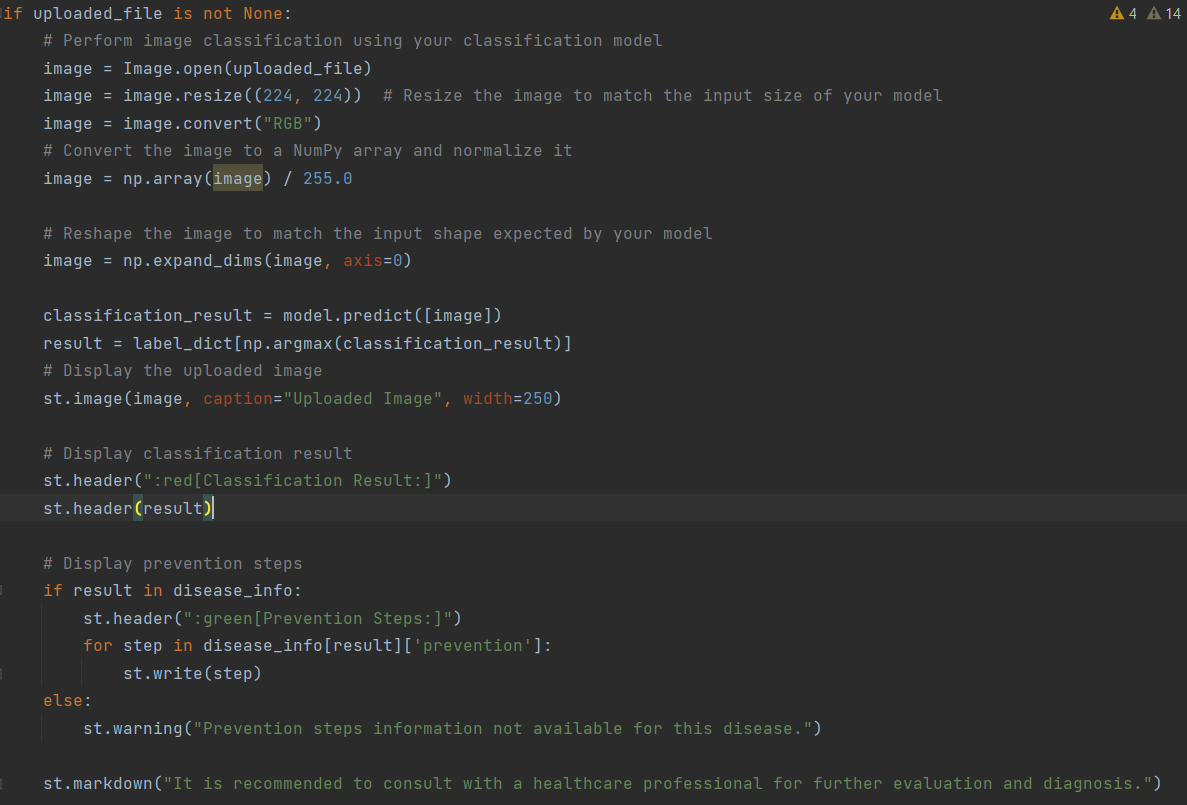
When you click on contact us, it will redirect you to the contact us page



**CSS FILE**



**Once you click on the Predict in Nail Disease Home page, it will redirect you to Nail Disease Prediction page**

**Showcasing prediction on UI: **

**Note: Target size need to change to 224,224 – Because vgg16 takes the input format in that size**

Here we are using the uploaded file on our web app and making predictions. We are applying some image processing techniques and then sending that preprocessed image to the model for predicting the class. This returns the numerical value of a class (like 0, 1, 2, etc.). This numerical value is passed to the index variable declared. This returns the name of the class. This name is rendered to the result variable used.

**Finally, run the application**

This is used to run the application in a localhost.

**Activity 3: Run the application**

● Open the command prompt from the start menu.

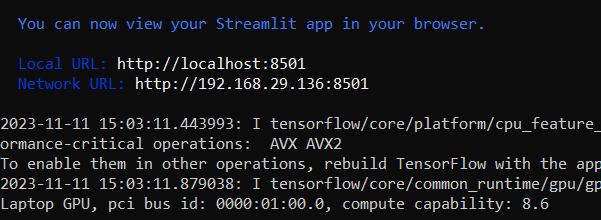
● Navigate to the folder where your Home.py resides.

● Now type “streamlit run Home.py” command.

● It will show the local host where your app is running on **http://192.168.29.136:8501** ● Copy that local host URL and open that URL in the browser. It does navigate me to where you can view your web page.

● Enter the values, click on the predict button and see the result/prediction on the web page.

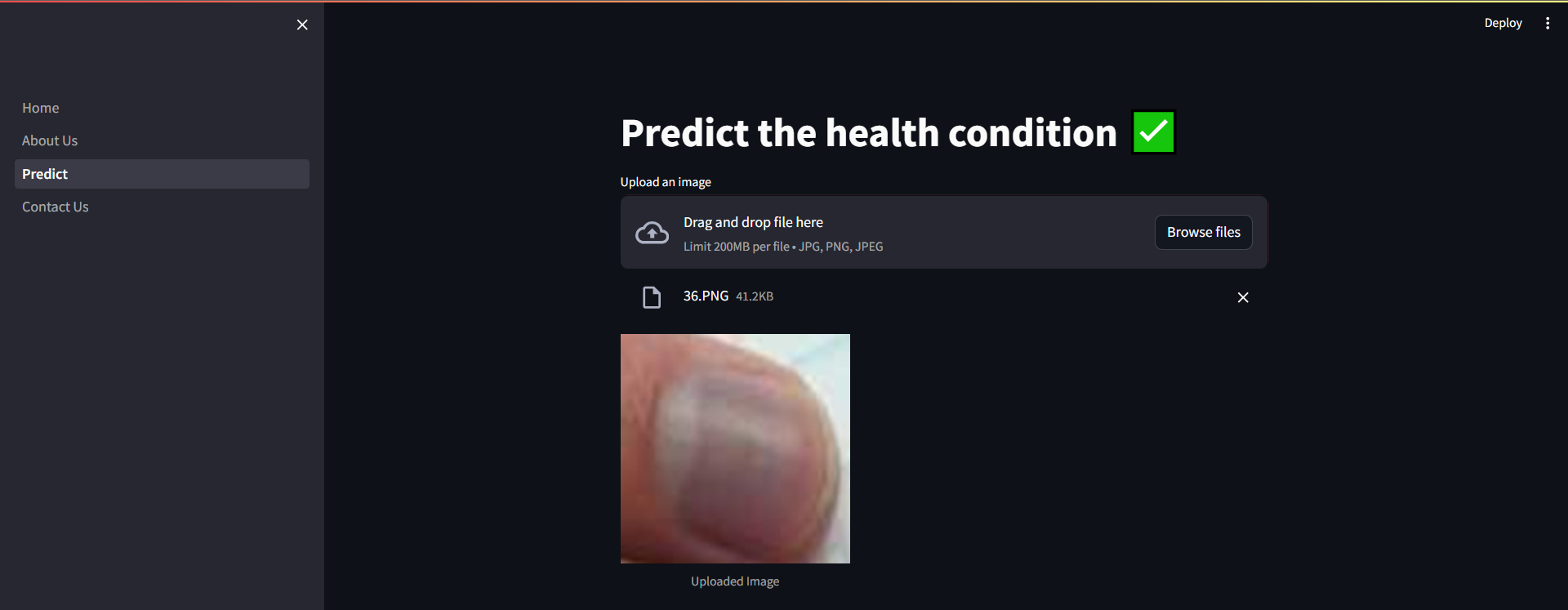
Then it will run on localhost: 8501

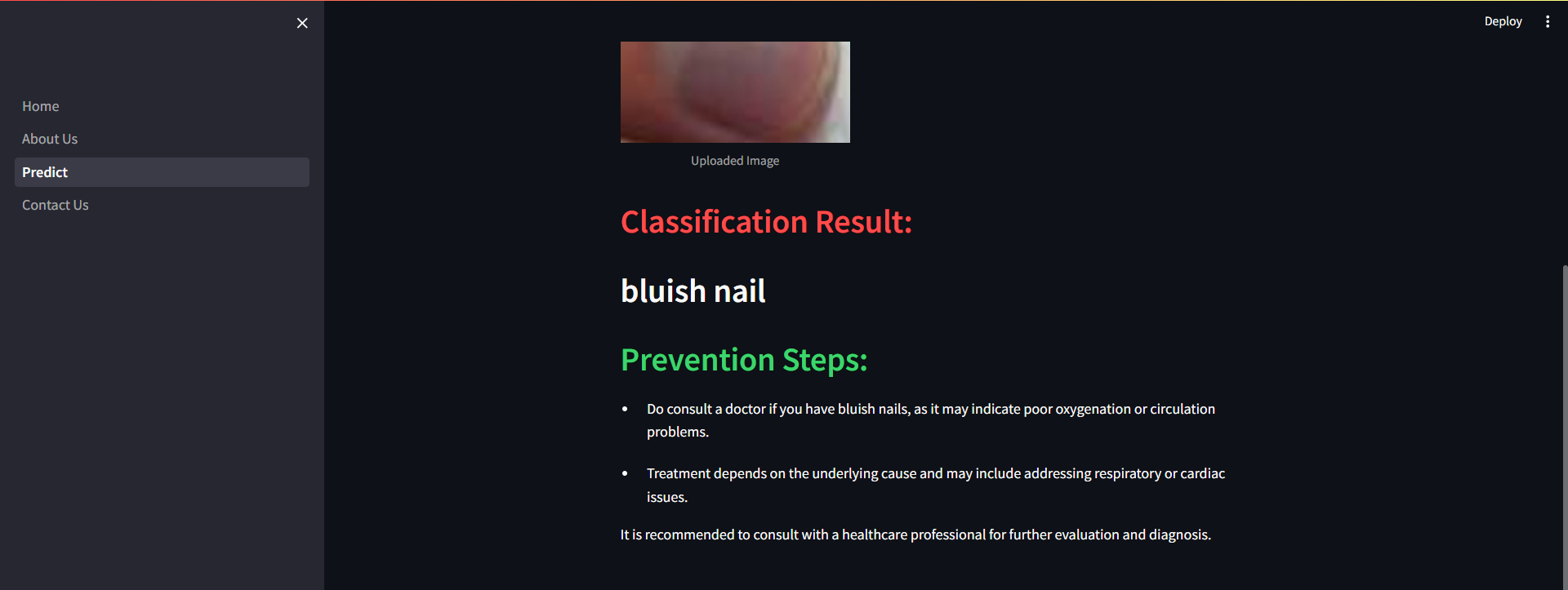


Navigate to the localhost (http://192.168.29.136:8501 ) where you can view your web page.

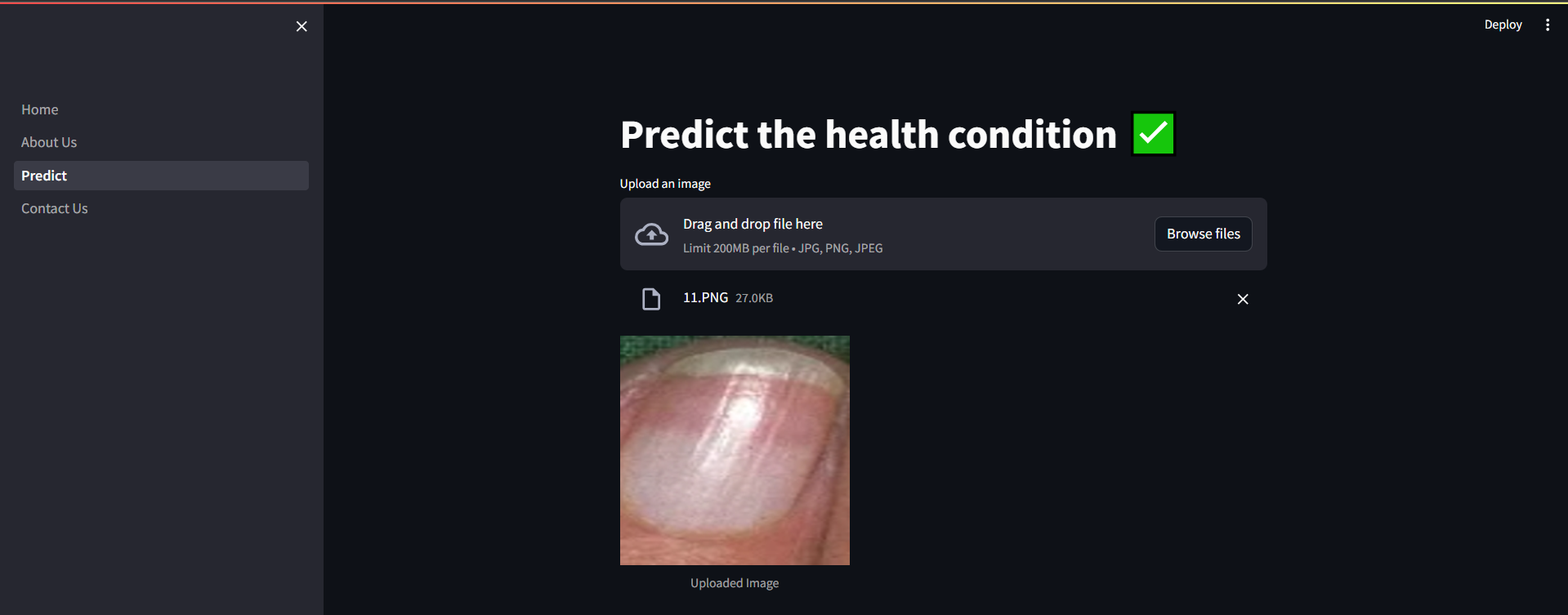
**FINAL OUTPUT:**

**Output 1**

****

****

**Output 2:**

****

****