Diabetic Retinopathy Detection

5.9.2022 - 13.12.2022

Siliveru Akash Durga
20BD1A055J
III Year, CSE-C

Overview

Diabetic retinopathy is the leading cause of blindness in the working-age population of the developed world. It is estimated to affect over 93 million people.

The US Center for Disease Control and Prevention estimates that 29.1 million people in the US have diabetes and the World Health Organization estimates that 347 million people have the disease worldwide. Diabetic Retinopathy (DR) is an eye disease associated with long-standing diabetes. Around 40% to 45% of Americans with diabetes have some stage of the disease. Progression to vision impairment can be slowed or averted if DR is detected in time, however this can be difficult as the disease often shows few symptoms until it is too late to provide effective treatment.

Dataset Source

The dataset used for the project is “Diabetic Retinopathy Detection” from Kaggle.

Link to the dataset:
https://www.kaggle.com/competitions/diabetic-retinopathy-detection/data

The dataset consists of 35,122 Training images.

A clinician has rated the presence of diabetic retinopathy in each image on a scale of 0 to 4, according to the following scale:

0-No DR

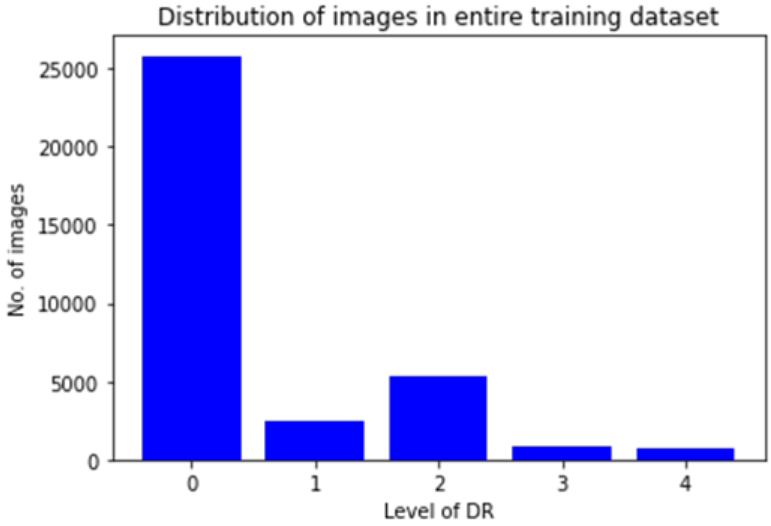
1-Mild DR

2-Moderate DR

3-Severe DR

4-Proliferate DR

The distribution of 5 classes is as follows.



Data Preprocessing

Since, the data was very undistributed across various classes, the dataset size has been limited to 17,500 Training images. Class -3, Class-4 have been Augmented by 4 times, 5 times using ImageDataGenerator(rotation_range=5, shear_range=0.1, zoom_range=0.1, horizontal_flip=True) function from keras.preprocessing.image library,making class 3 a total of 3492 and class 4 a total of 3540 images.

Finally, the dataset consists of 3500 images from each class. The images are then resized to 128*128 .

Model Used - MobilenetV2 Deep Learning

MobileNet uses depthwise separable convolutions. It significantly reduces the number of parameters when compared to the network with regular convolutions with the same depth in the nets. This results in lightweight deep neural networks. A depthwise separable convolution is made from two operations:

-Depthwise convolution.

– Pointwise convolution

Architecture:

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1 Conv / s1	$3 \times 3 \times 512$ dw $1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$ $14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Front End

HTML,CSS,BOOTSTRAP 5

Back End

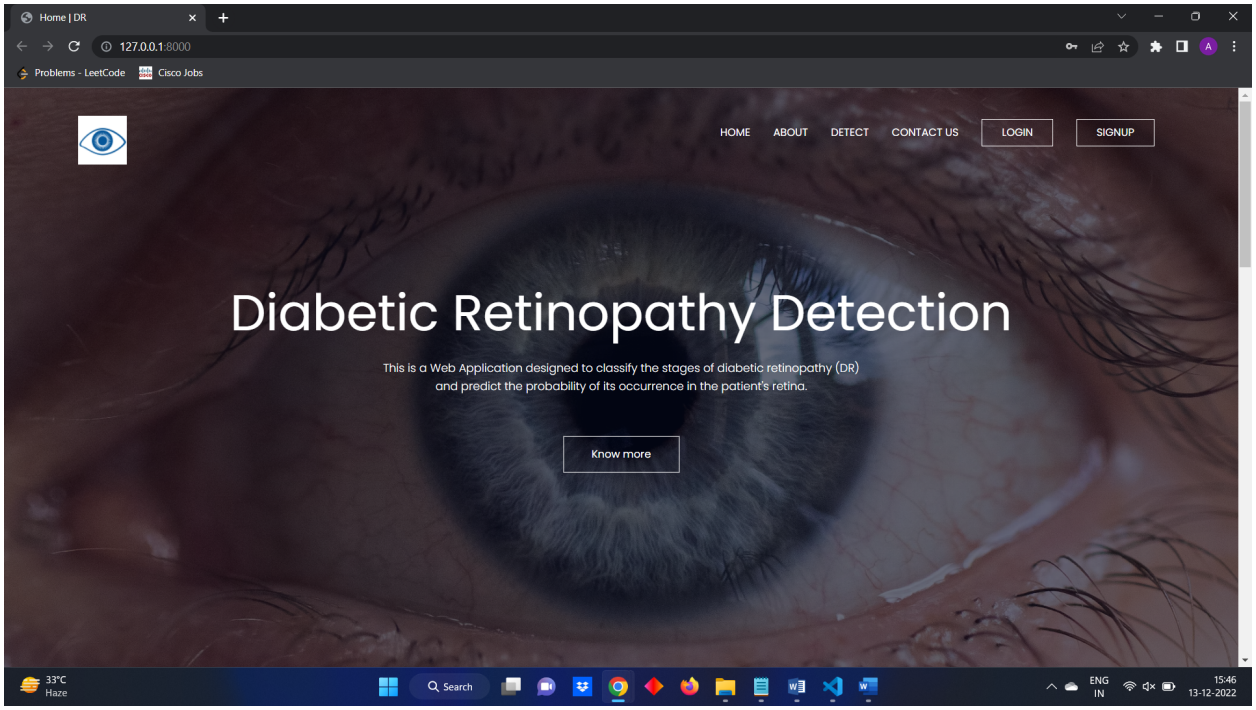
DJANGO

Database

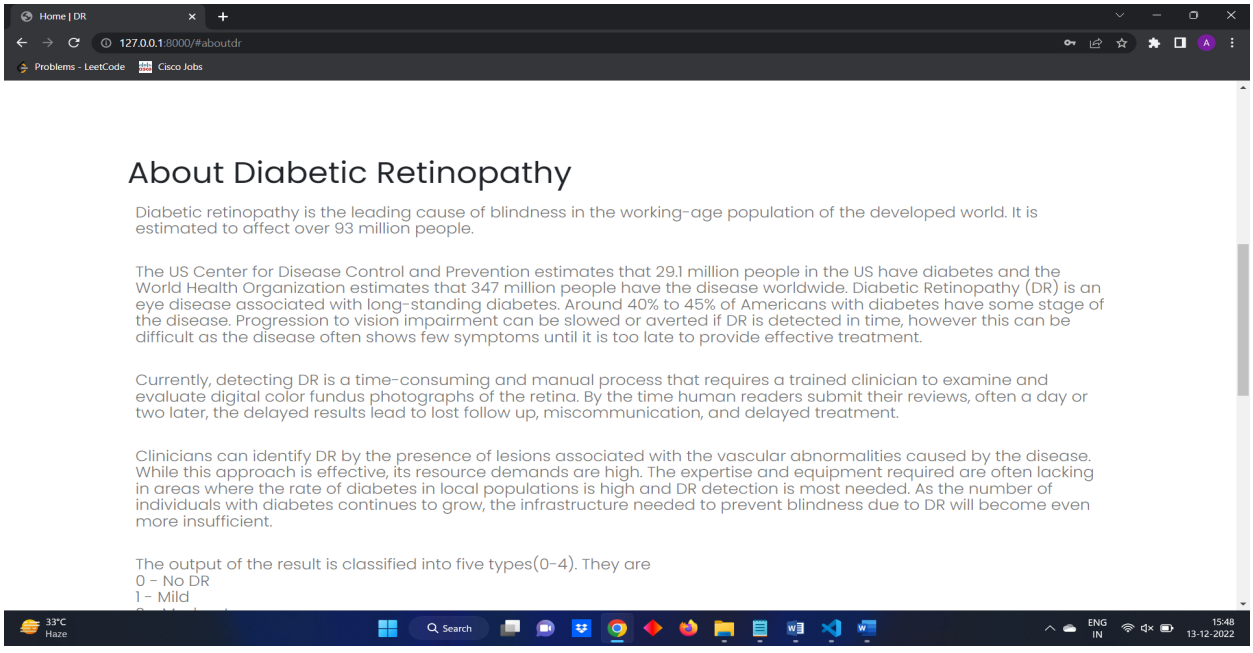
SQLITE

Project Sample

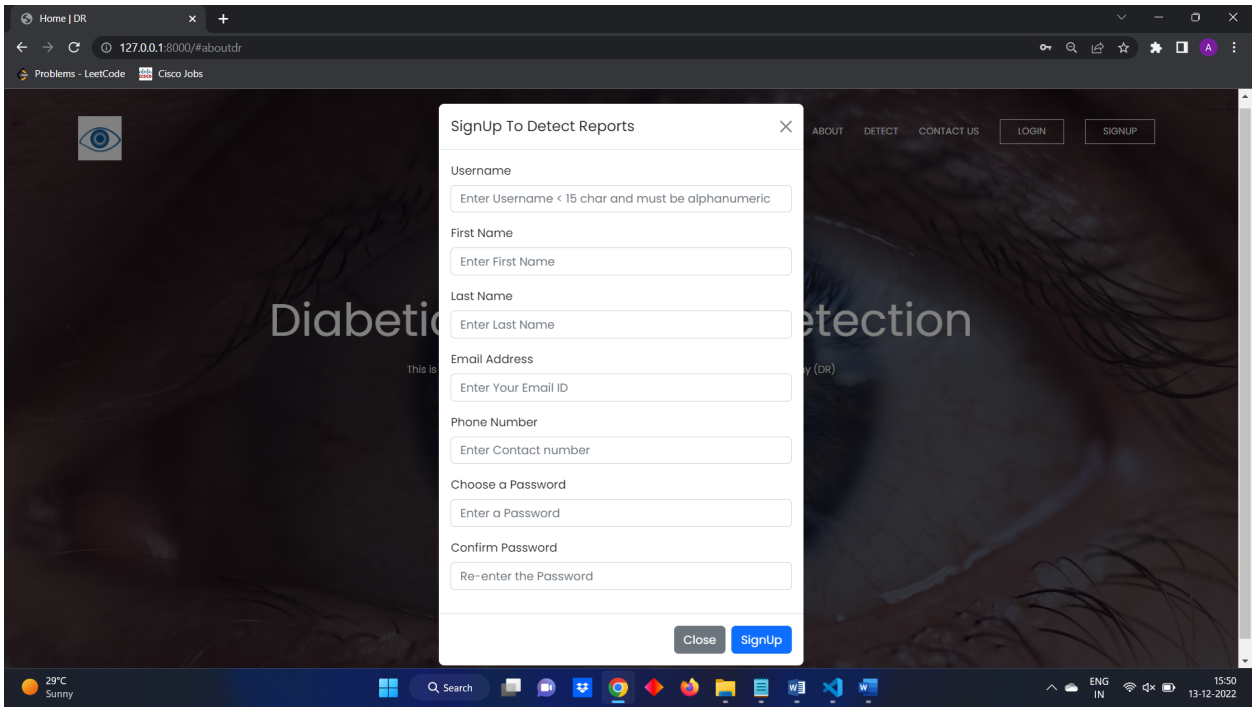
(i) Home Page



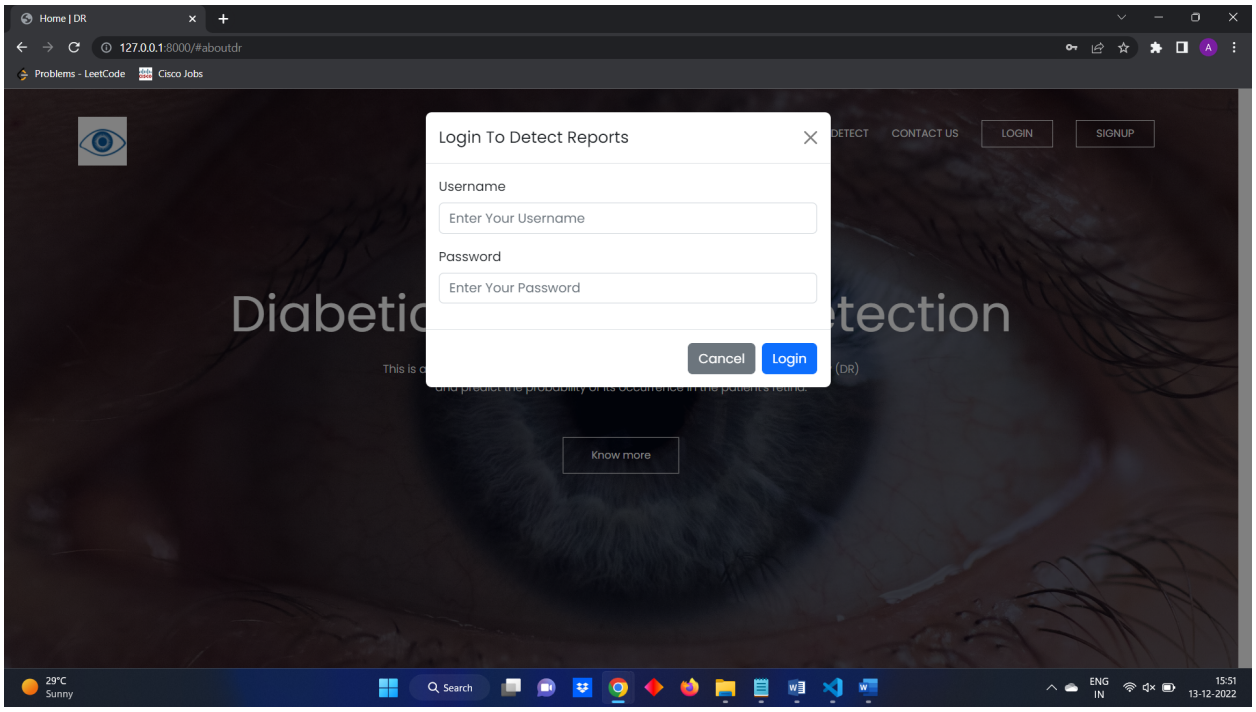
(ii) About



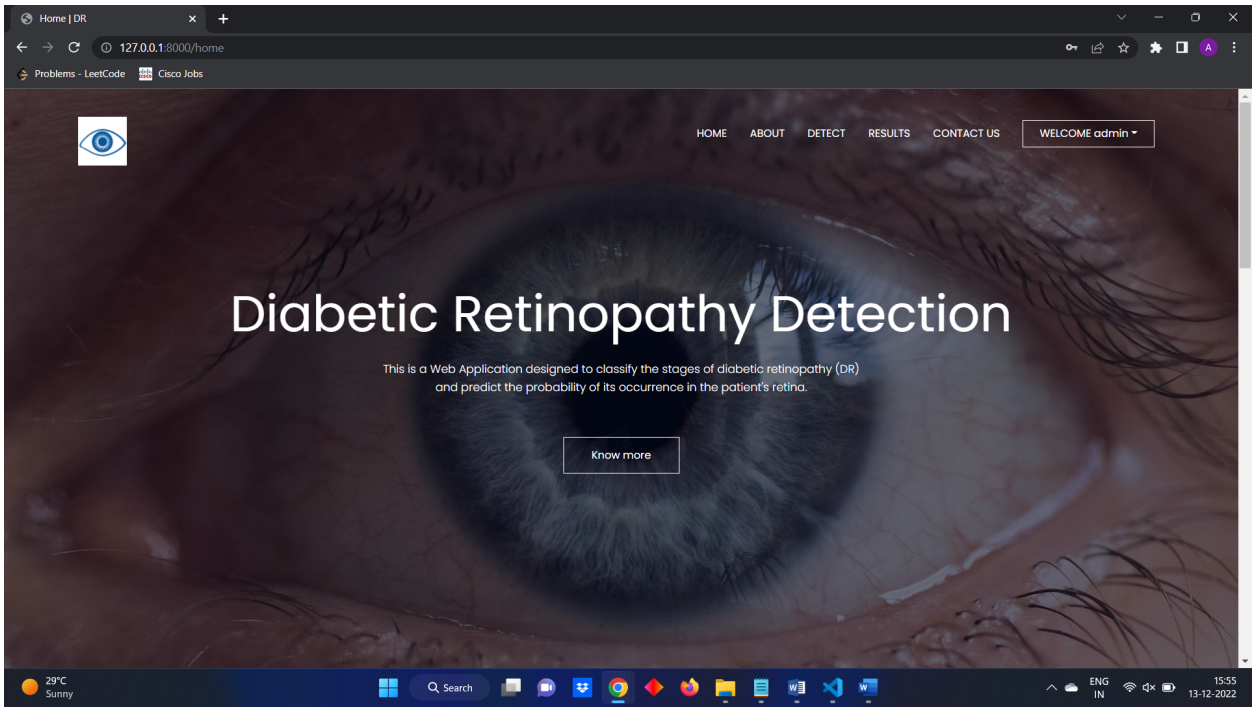
(iii) SignUp



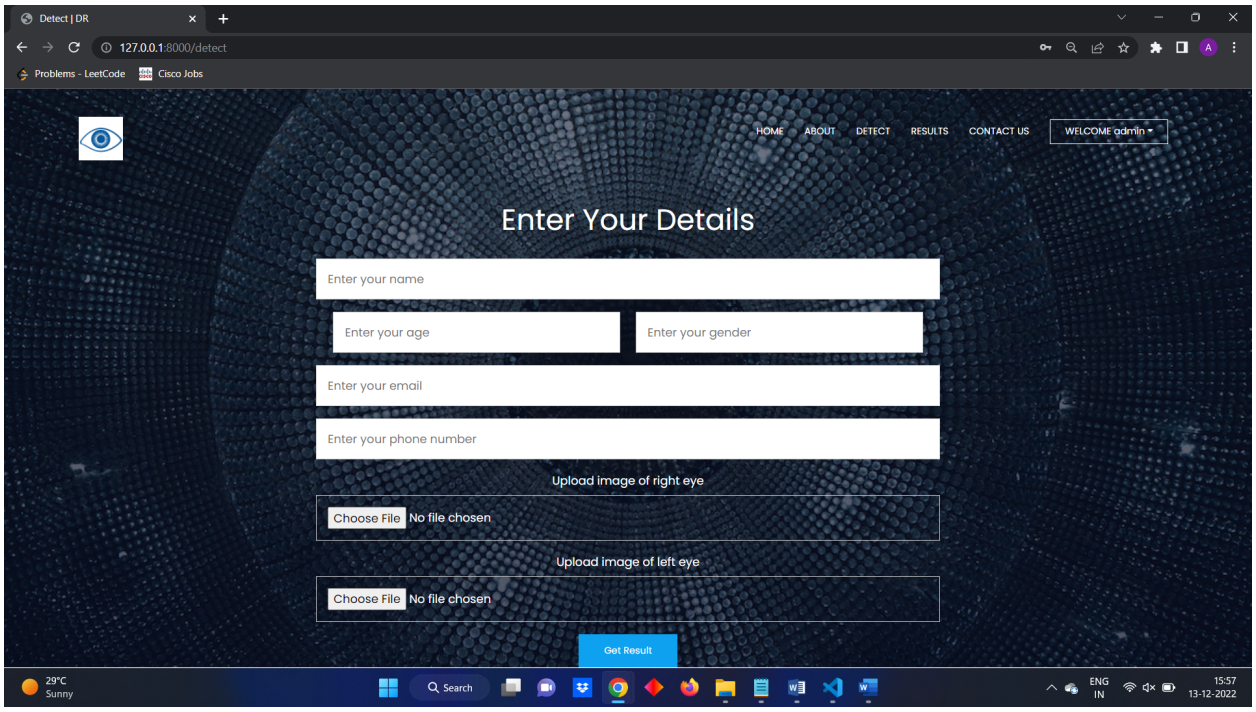
(iv) Login



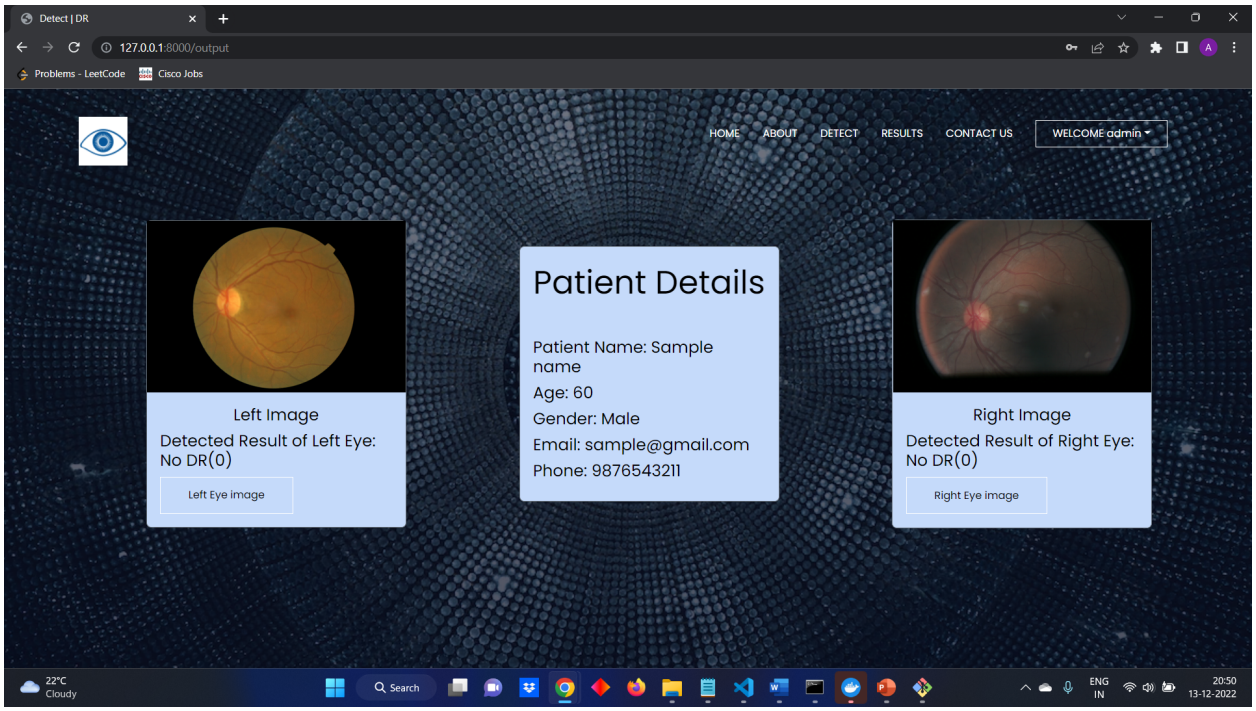
(v) After Login with admin (sample account)



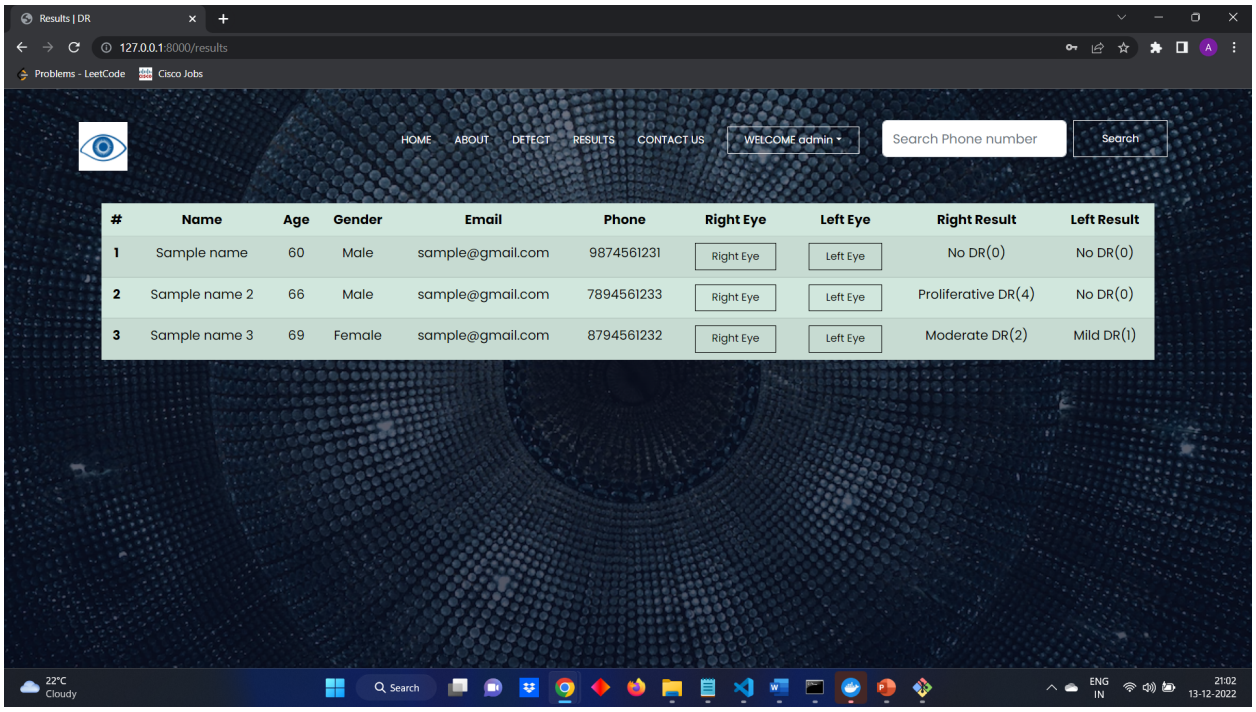
(vi) Detect Page



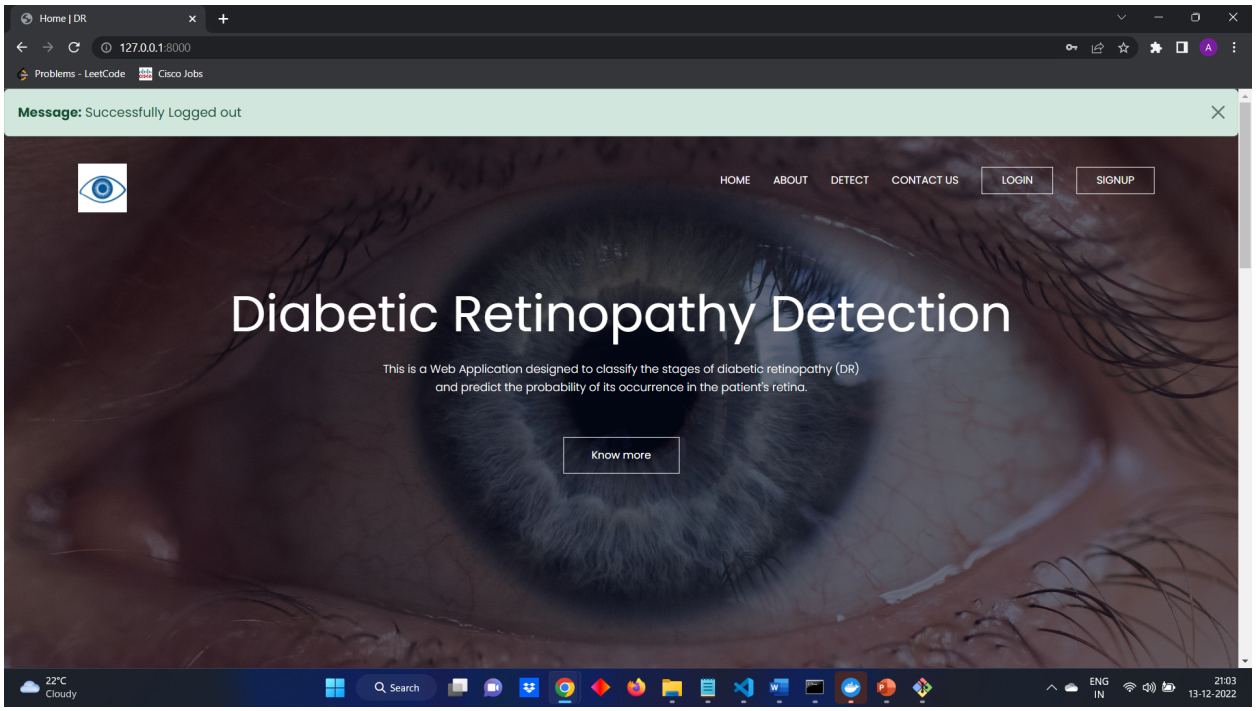
(vii) Output page



(viii) Results page



(ix) After logout



Train Accuracy:

```
C: > Users > SILIVERU AKASH DURGA > Downloads > 85.21%.ipynb > score, acc = model.evaluate(test_datagen.flow(X_test, y_binary_test, batch_size=32, seed=42))
+ Code + Markdown | Run All | Clear Outputs of All Cells | Outline ···
    Epoch 3/100
    383/383 [==============================] - 27s 71ms/step - loss: 0.9115 - accuracy: 0.5974 - val_loss: 1.1508 - val_accuracy: 0.5293
    Epoch 4/100
    383/383 [==============================] - 28s 72ms/step - loss: 0.8795 - accuracy: 0.6194 - val_loss: 0.9305 - val_accuracy: 0.5954
    Epoch 5/100
    383/383 [==============================] - 27s 71ms/step - loss: 0.8344 - accuracy: 0.6416 - val_loss: 0.9508 - val_accuracy: 0.5907
    Epoch 6/100
    383/383 [==============================] - 27s 71ms/step - loss: 0.7921 - accuracy: 0.6646 - val_loss: 0.8270 - val_accuracy: 0.6450
    Epoch 7/100
    383/383 [==============================] - 27s 70ms/step - loss: 0.7509 - accuracy: 0.6883 - val_loss: 0.8522 - val_accuracy: 0.6499
    Epoch 8/100
    383/383 [==============================] - 27s 71ms/step - loss: 0.6956 - accuracy: 0.7088 - val_loss: 0.8285 - val_accuracy: 0.6613
    Epoch 9/100
    383/383 [==============================] - 27s 71ms/step - loss: 0.6671 - accuracy: 0.7265 - val_loss: 0.7302 - val_accuracy: 0.6903
    Epoch 10/100
    383/383 [==============================] - 27s 71ms/step - loss: 0.6112 - accuracy: 0.7513 - val_loss: 1.1901 - val_accuracy: 0.5842
    Epoch 11/100
    383/383 [==============================] - 27s 71ms/step - loss: 0.5571 - accuracy: 0.7780 - val_loss: 0.8134 - val_accuracy: 0.6613
    Epoch 12/100
    383/383 [==============================] - 27s 70ms/step - loss: 0.5278 - accuracy: 0.7863 - val_loss: 0.7867 - val_accuracy: 0.7036
    Epoch 13/100
    ...
    Epoch 99/100
    383/383 [==============================] - 27s 71ms/step - loss: 0.0383 - accuracy: 0.9865 - val_loss: 0.6899 - val_accuracy: 0.8394
    Epoch 100/100
    383/383 [==============================] - 27s 72ms/step - loss: 0.0381 - accuracy: 0.9859 - val_loss: 0.6340 - val_accuracy: 0.8522
```

Test Accuracy:

```
score, acc = model.evaluate(test_datagen.flow(X_test, y_binary_test, batch_size=32, seed=42))
print('Test Loss =', score)
print('Test Accuracy =', acc)
```

```
165/165 [==============================] - 2s 14ms/step - loss: 0.6340 - accuracy: 0.8522
Test Loss = 0.6339656114578247
Test Accuracy = 0.8521904945373535
```