**AI/ML WITH CLOUD**

**TITLE-Deep Learning Fundus Image Analysis for Early Detection**

**of Diabetic Retinopathy**

**INTRODUCTION:**

**Description:**

Diabetic Retinopathy (DR) is a common complication of diabetes mellitus, which causes lesions on the retina that affect vision. If it is not detected early, it can lead to blindness. Unfortunately, DR is not a reversible process, and treatment only sustains vision. DR early detection and treatment can significantly reduce the risk of vision loss. The manual diagnosis process of DR retina fundus images by ophthalmologists is time, effort and cost-consuming and prone to misdiagnosis unlike computer-aided diagnosis systems.

Transfer learning has become one of the most common techniques that has achieved better performance in many areas, especially in medical image analysis and classification. We used Transfer Learning techniques like Inception V3, Resnet50, Xception V3 that are more widely used as a transfer learning method in medical image analysis and they are highly effective.

**Project Flow:**

1. The user interacts with the UI (User Interface) to choose the image.
2. The chosen image analyzed by the model which is integrated with flask application.
3. The Xception Model analyses the image, then the prediction is showcased on the Flask UI

->To accomplish this, we have to complete all the activities and tasks listed below

1. Data Collection.

a. Create a Train and Test path.

1. Data Pre-processing.

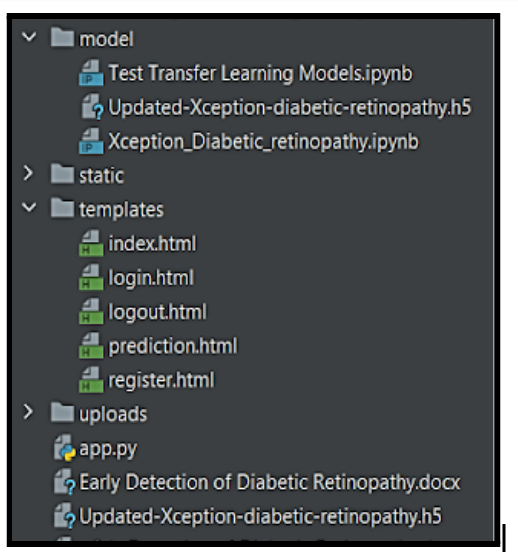
a. Import the required library

b. Configure ImageDataGenerator class

c. ApplyImageDataGenerator functionality to Trainset and Testset

1. Model Build
   1. Pre-trained CNN model as a Feature Extractor
   2. Adding Dense Layer
   3. Configure the Learning Process
   4. Train the model
   5. Save the Model
   6. Test the model
2. Cloudant DB
   1. Register & Login to IBM Cloud
   2. Create Service Instance

**Project Structure:**



->We are building a flask application which needs HTML pages stored in the templates folder, CSS, Images stored in a static folder and a python script app.py for scripting.

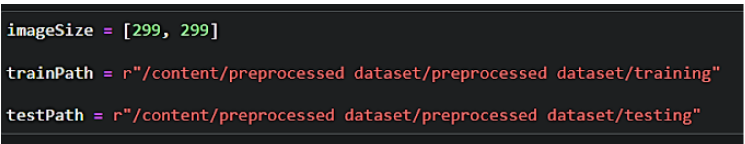
->Updated-Xception-diabetic-retinopathy.h5 is our saved model. Further we will use this model for flask integration. Mode contains model training files.

**Create Training And Testing Path:**

To build a DL model we have to split training and testing data into two separate folders. But in the project dataset folder training and testing folders are presented. So, in this case we just have to assign a variable and pass the folder path to it.

Four different transfer learning models are used in our project and the best model (Xception) is selected.

The image input size of xception model is 299, 299.



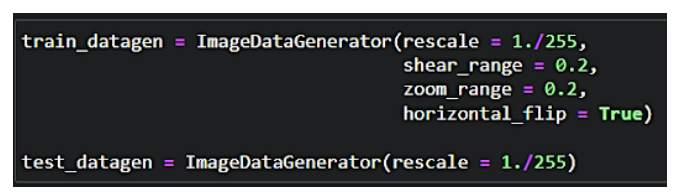
**Configure ImageDataGenerator Class:**

mageDataGenerator 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 train and test.



**Apply ImageDataGenerator Functionality To Train Set And Test Set:**

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

This function will return batches of images from the subdirectories

Arguments:

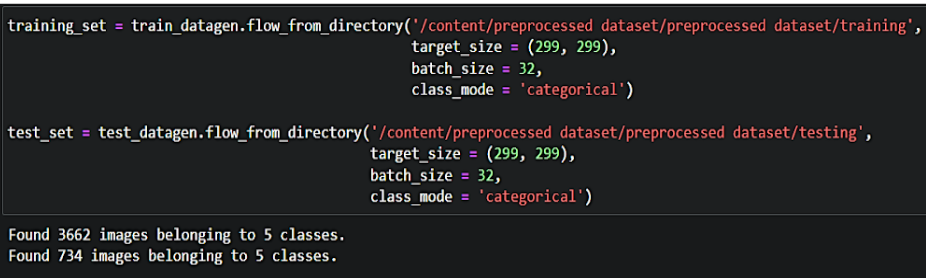
* directory: Directory where the data is located. If labels are "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.
* batch\_size: Size of the batches of data which is  64.
* target\_size: Size to resize images after they are 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).



**MODEL BUILDING:**

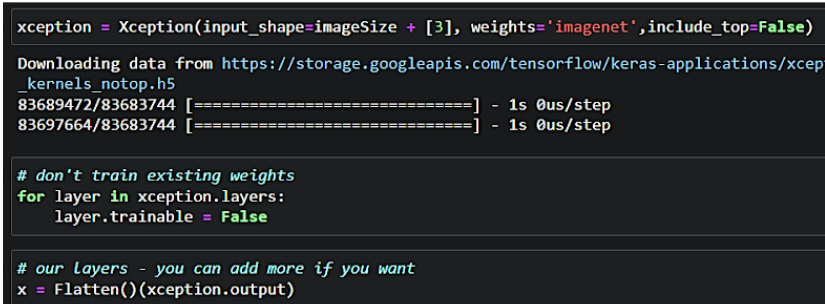
### Pre-Trained CNN Model As A Feature Extractor:

For one of the models, we will use it as a simple feature extractor by freezing all the five convolution blocks to make sure their weights don’t get updated after each epoch as we train our own model.

Here, we have considered images of dimension (229,229,3).

 Also, we have assigned include\_top = False because we are using convolution layer for features extraction and wants to train fully connected layer for our images classification(since it is not the part of Imagenet dataset)

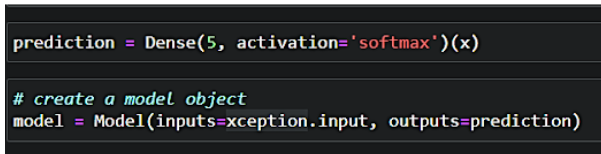
Flatten layer flattens the input. Does not affect the batch size.



### Adding Dense Layers:

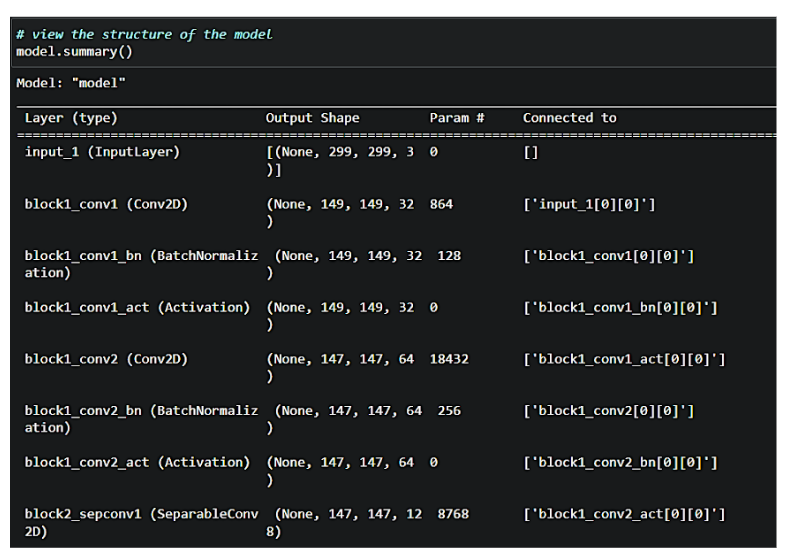
A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer.

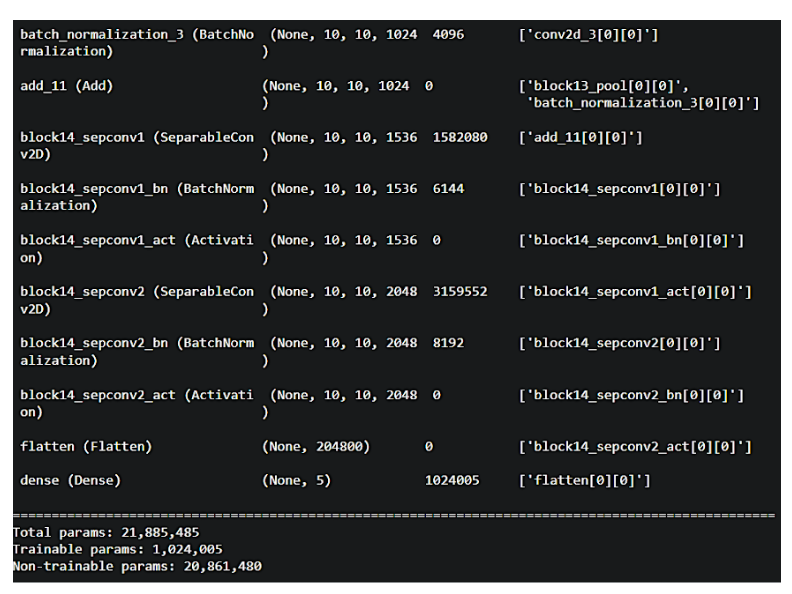
Let us create a model object named model with inputs as xception.input and output as dense layer.



The number of neurons in the Dense layer is the same as the number of classes in the training set. The neurons in the last Dense layer, use softmax activation to convert their outputs into respective probabilities.

Understanding the model is a very important phase to properly use it for training and prediction purposes. Keras provides a simple method, summary to get the full information about the model and its layers.



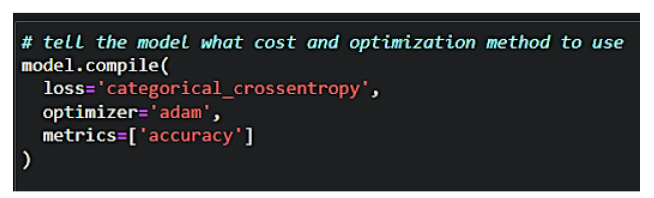


### Configure The Learning Process:

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



**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 10 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.
* validation\_data can be either:

                      - an inputs and targets list

                      - a generator

                      - an inputs, targets, and sample\_weights list which can be used to evaluate

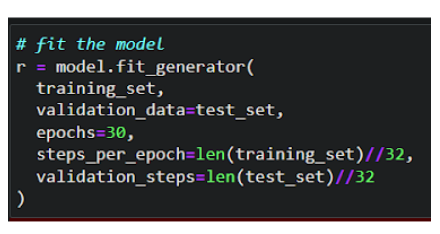
                        the loss and metrics for any model after any epoch has ended.

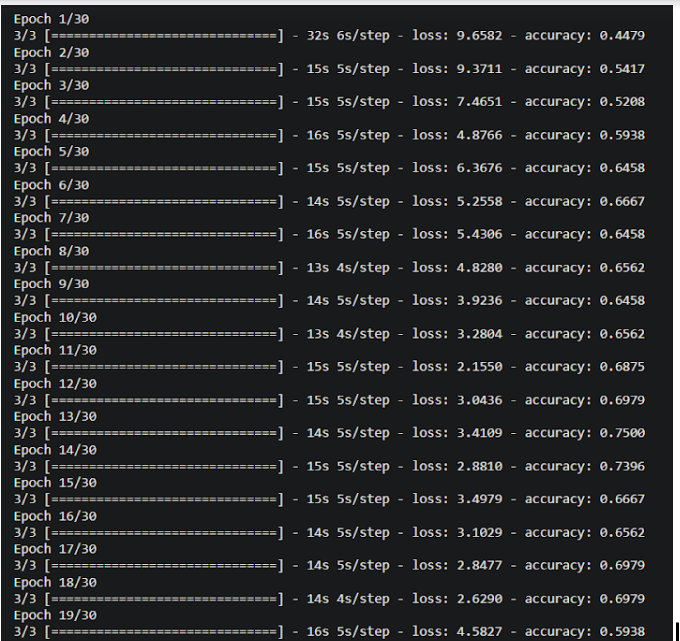
* validation\_steps: only if the validation\_data is a generator then only this argument

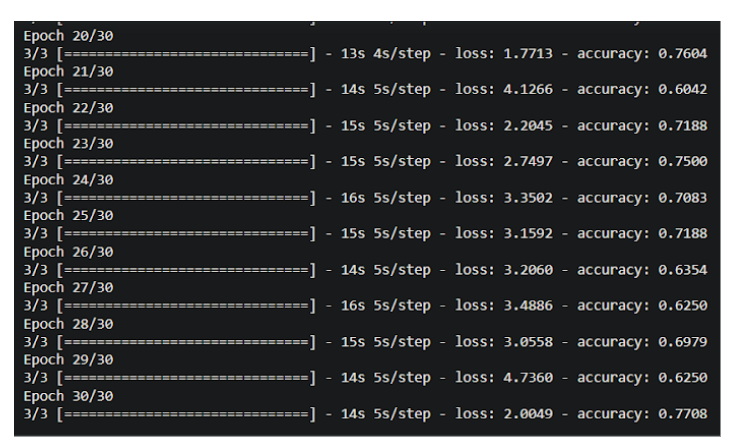
can be used. It specifies the total number of steps taken from the generator before it is

stopped at every epoch and its value is calculated as the total number of validation data points

in your dataset divided by the validation batch size.







### 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 multi-dimensional arrays of scientific data.

****

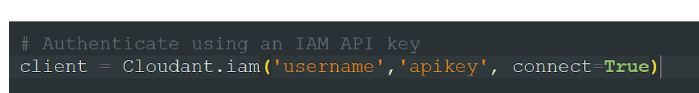
### Cloudant DB:

**Create Database:**

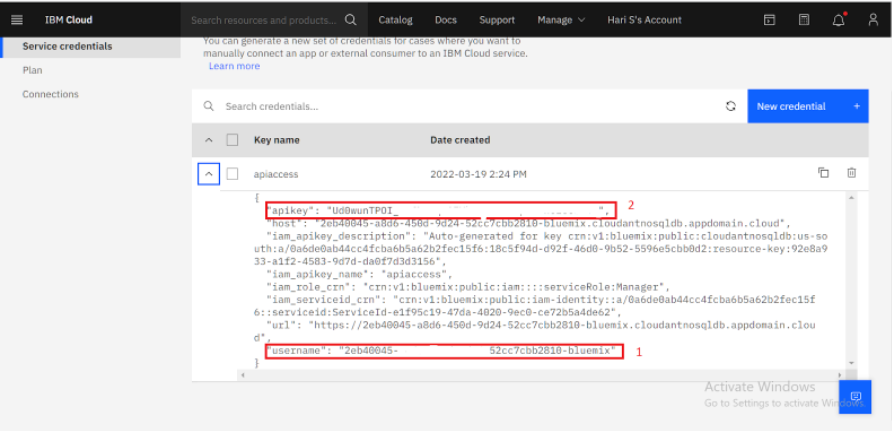
* In order to manage a connection from a local system you must first initialize the connection by constructing a Cloudant client.We need to import the cloudant library.



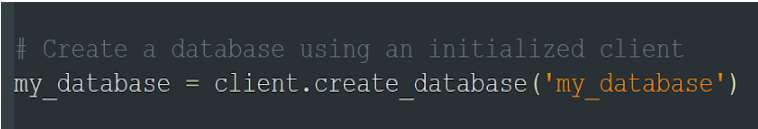
* IBM Cloud Identity & Access Management enables you to securely authenticate users and control access to all cloud resources consistently in the IBM Bluemix Cloud Platform.



In the above cloudant.iam() method we have to give  username & apikey to build the connection with cloudant DB.



* Once a connection is established you can then create a database, open an existing database.
* Create a database as my\_database.



**Application Building:**

In this section, we will be building a web application that is integrated to the model we built. A UI is provided to the user where he has uploaded the image. Based on the saved model, the uploaded image will be analyzed and prediction is showcased on the UI.

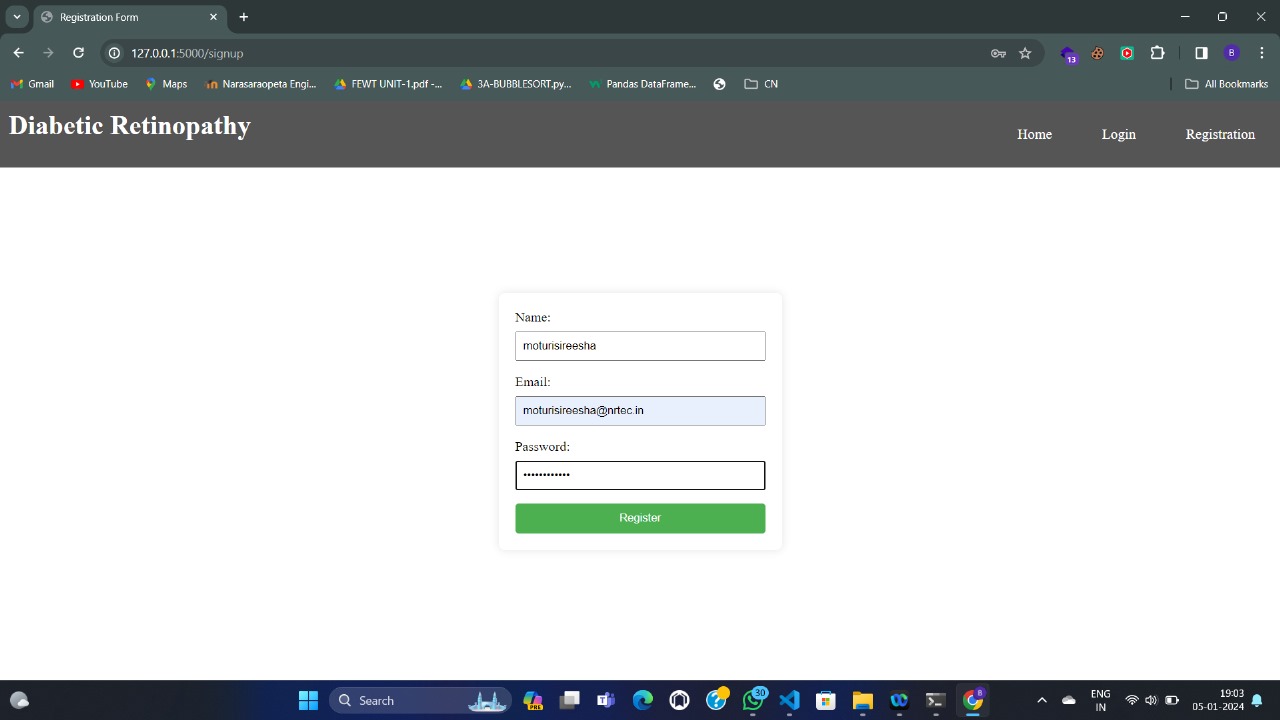
This section has the following tasks

* Building HTML Pages
* Building server-side script

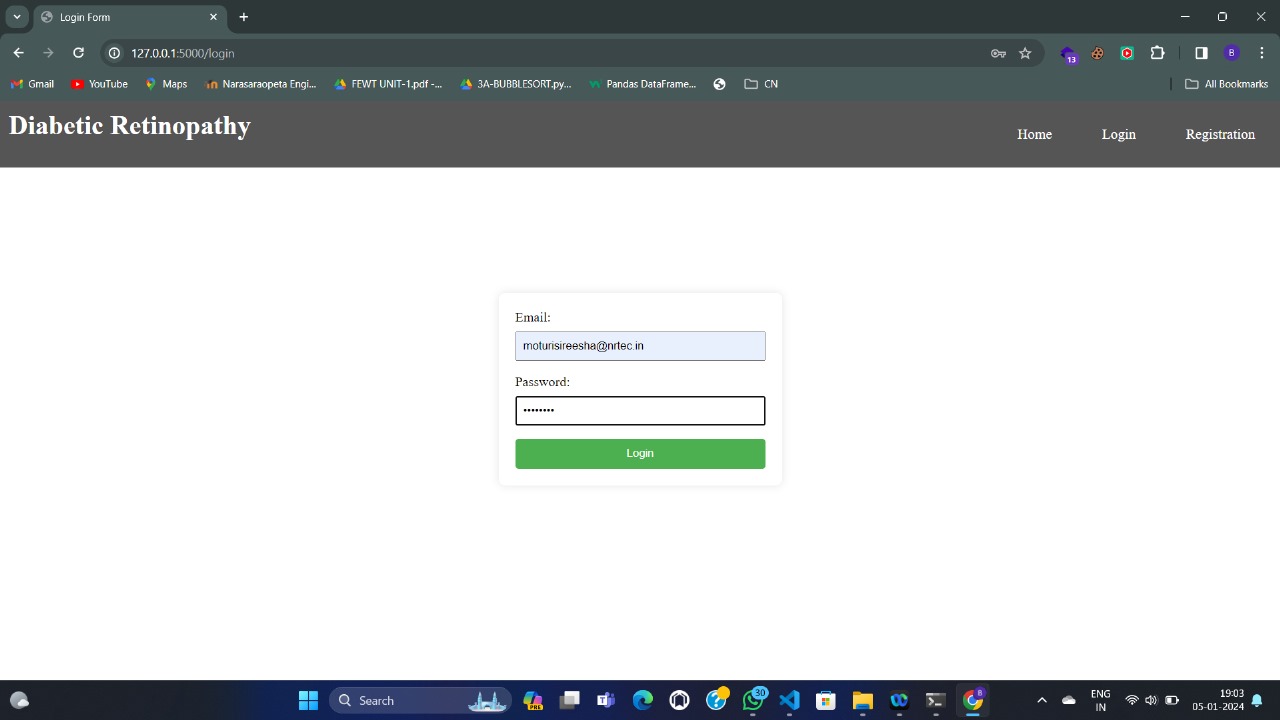
**Building Html Pages:**

For this project create three HTML files namely

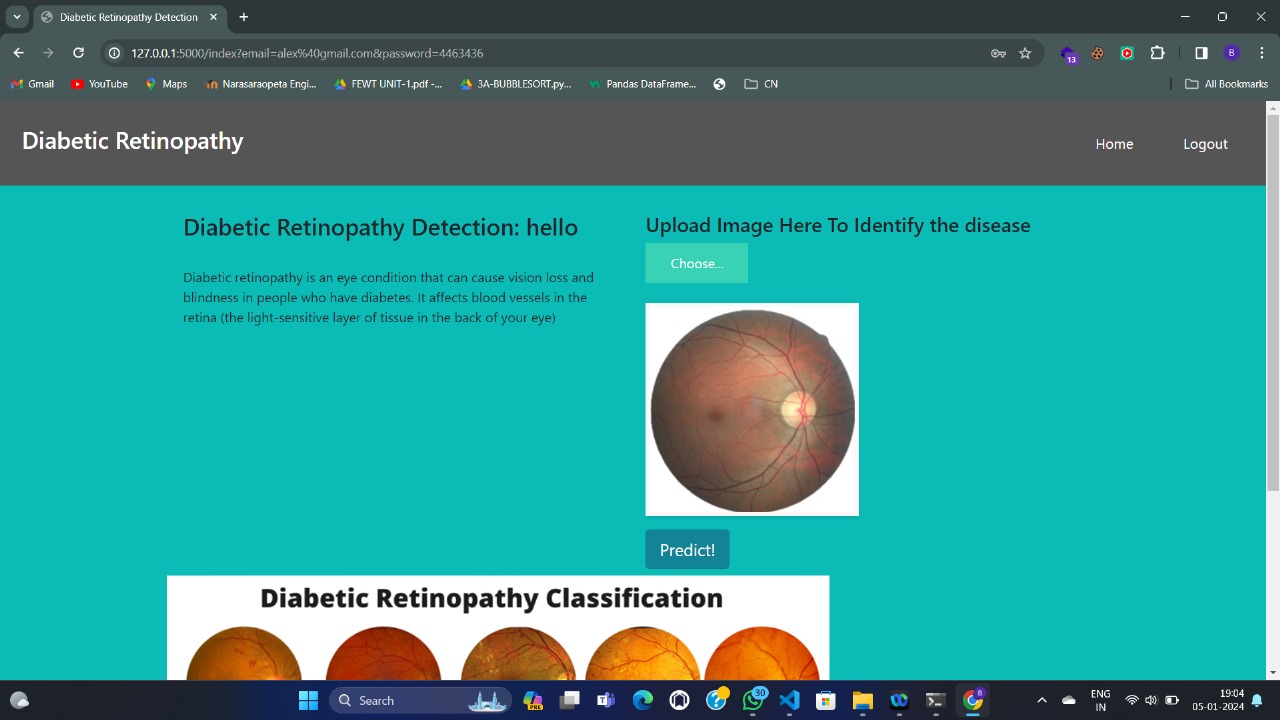
* index.html
* register.html
* login.html
* prediction.html
* logout.html
* This is Register form

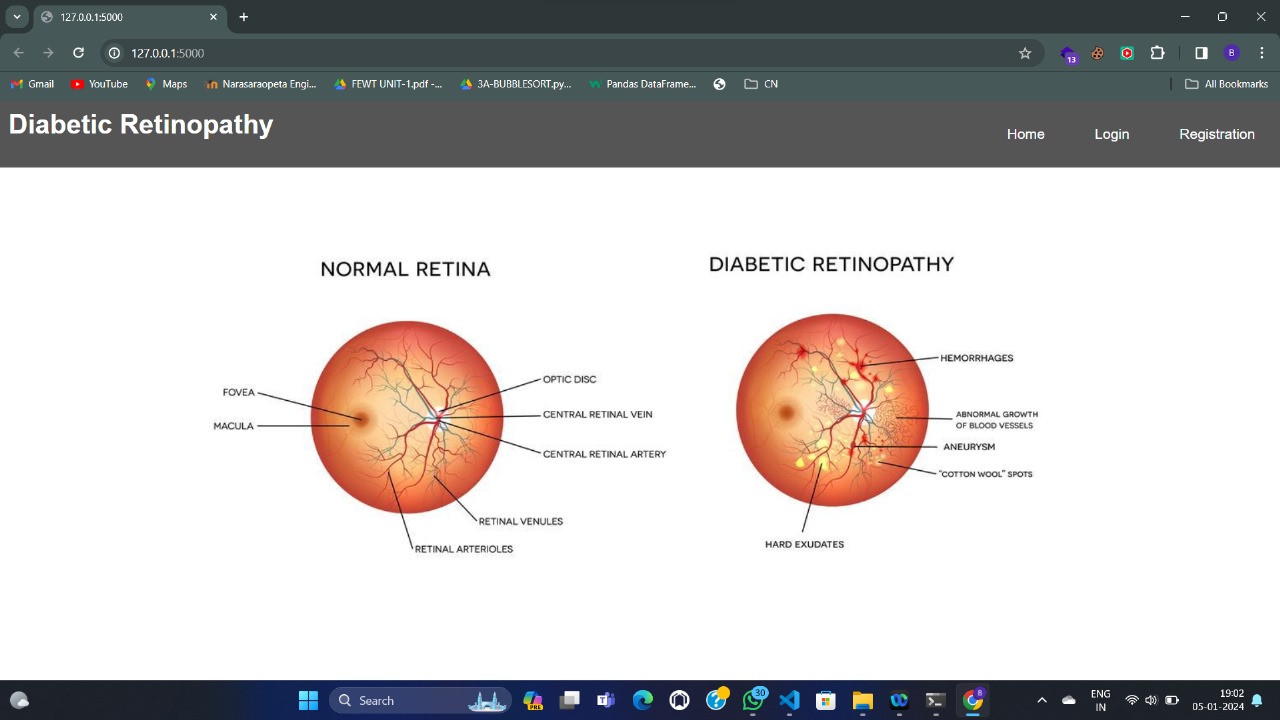


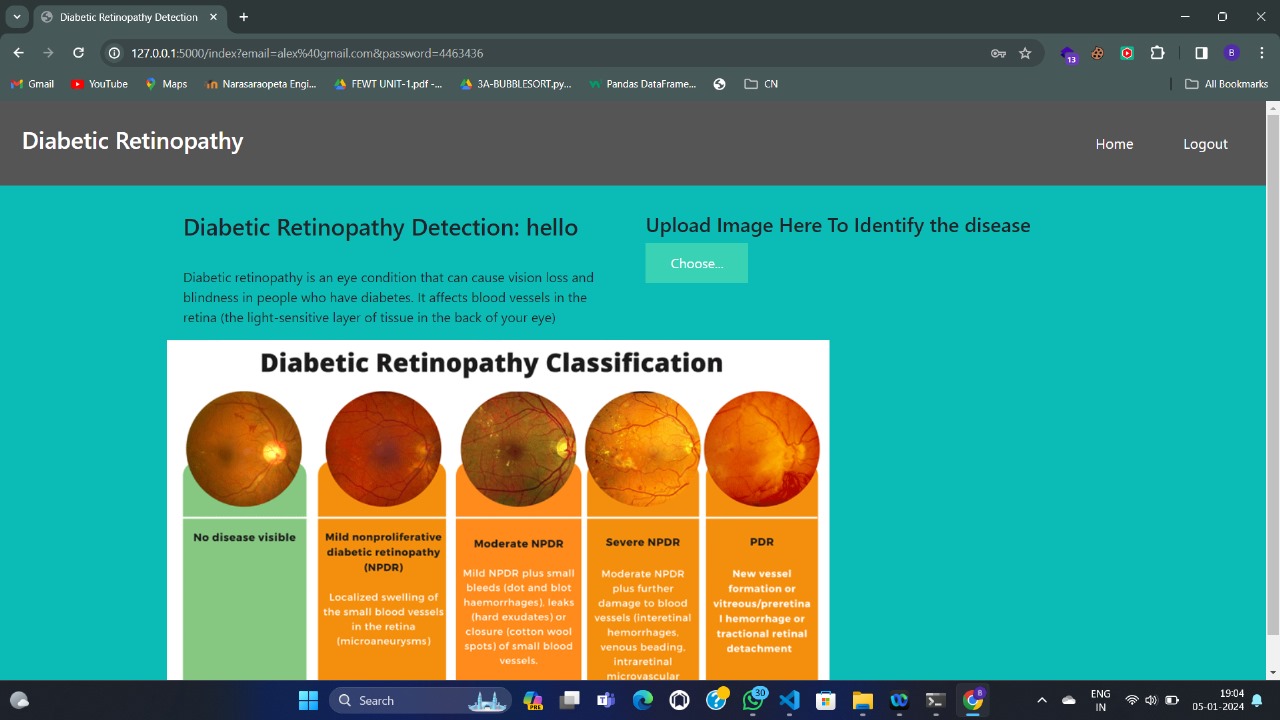
* This is Login form



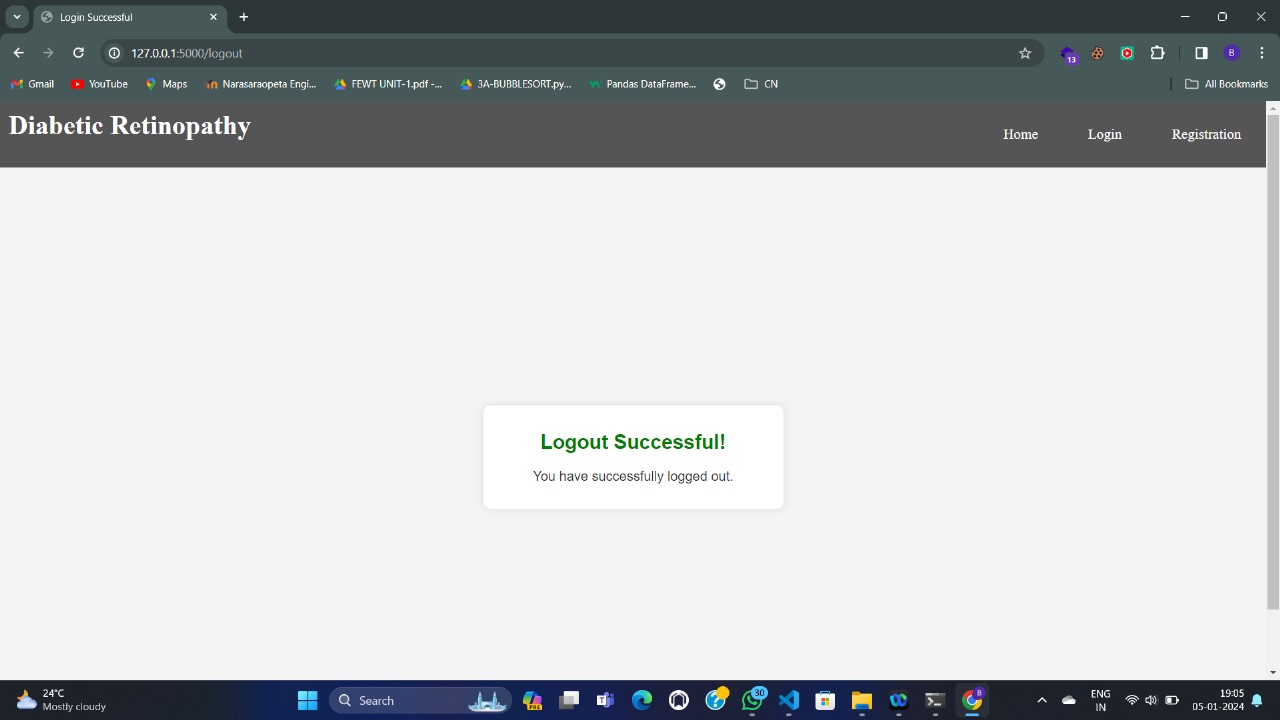
* This is Home page



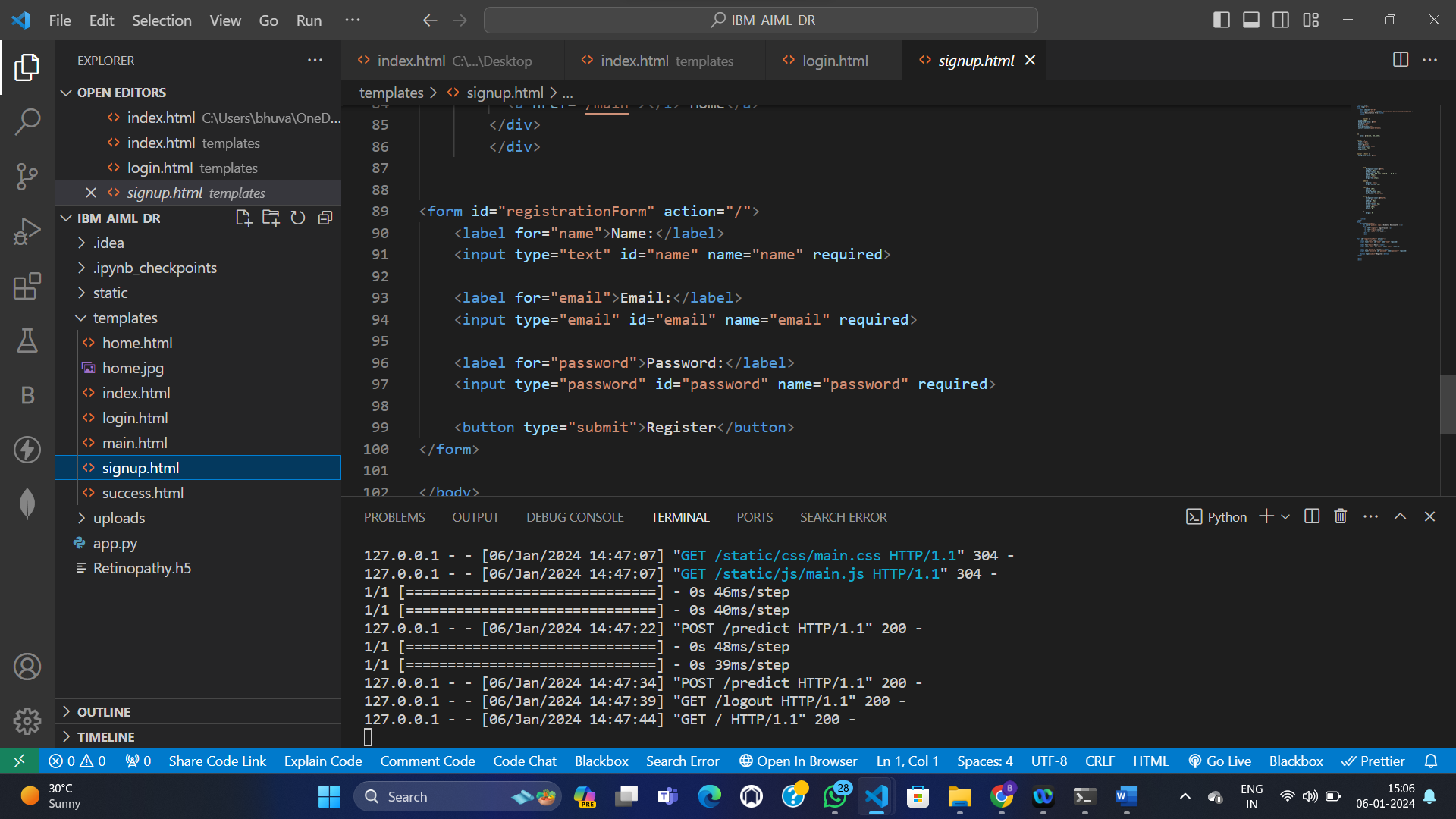


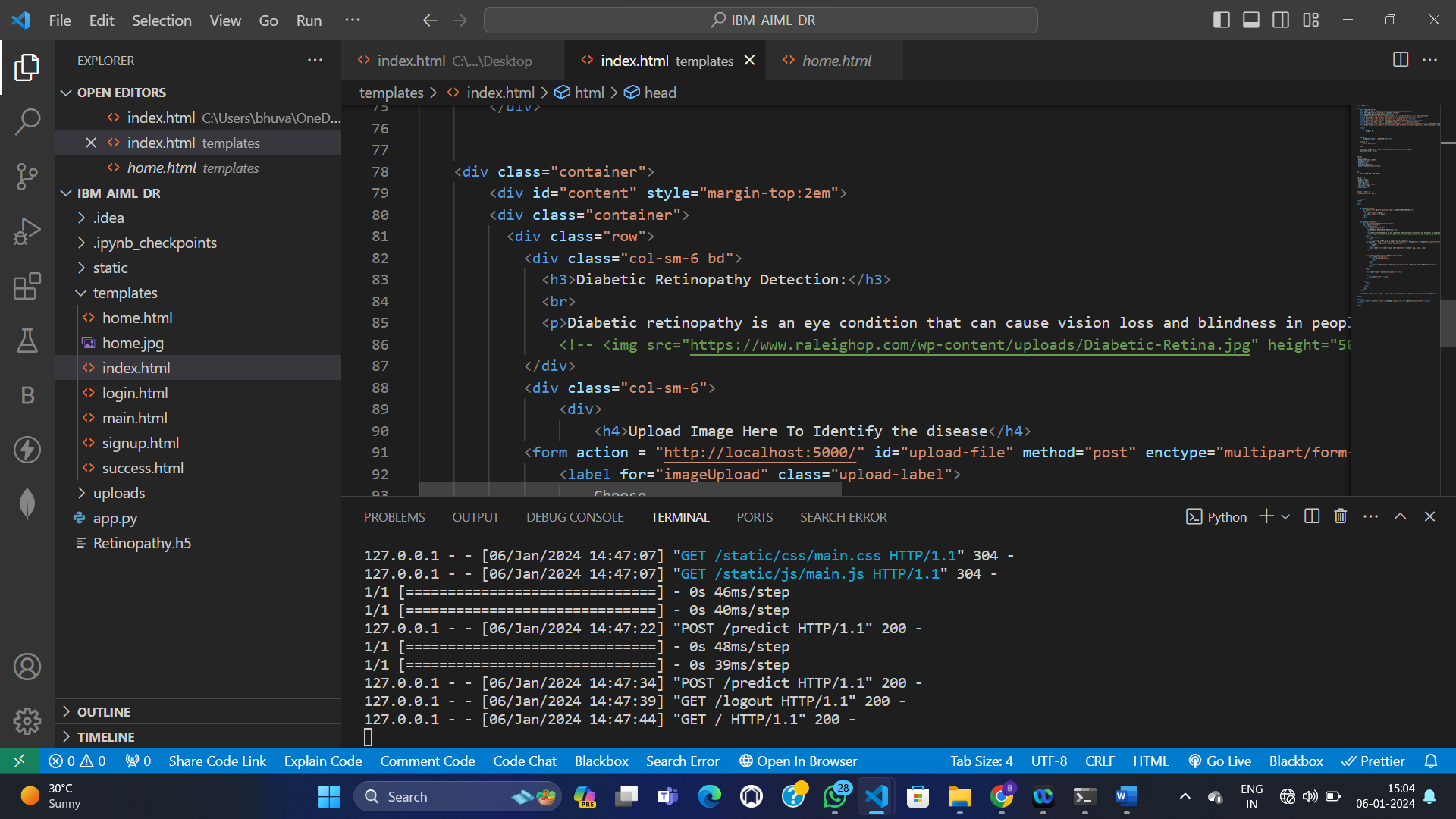


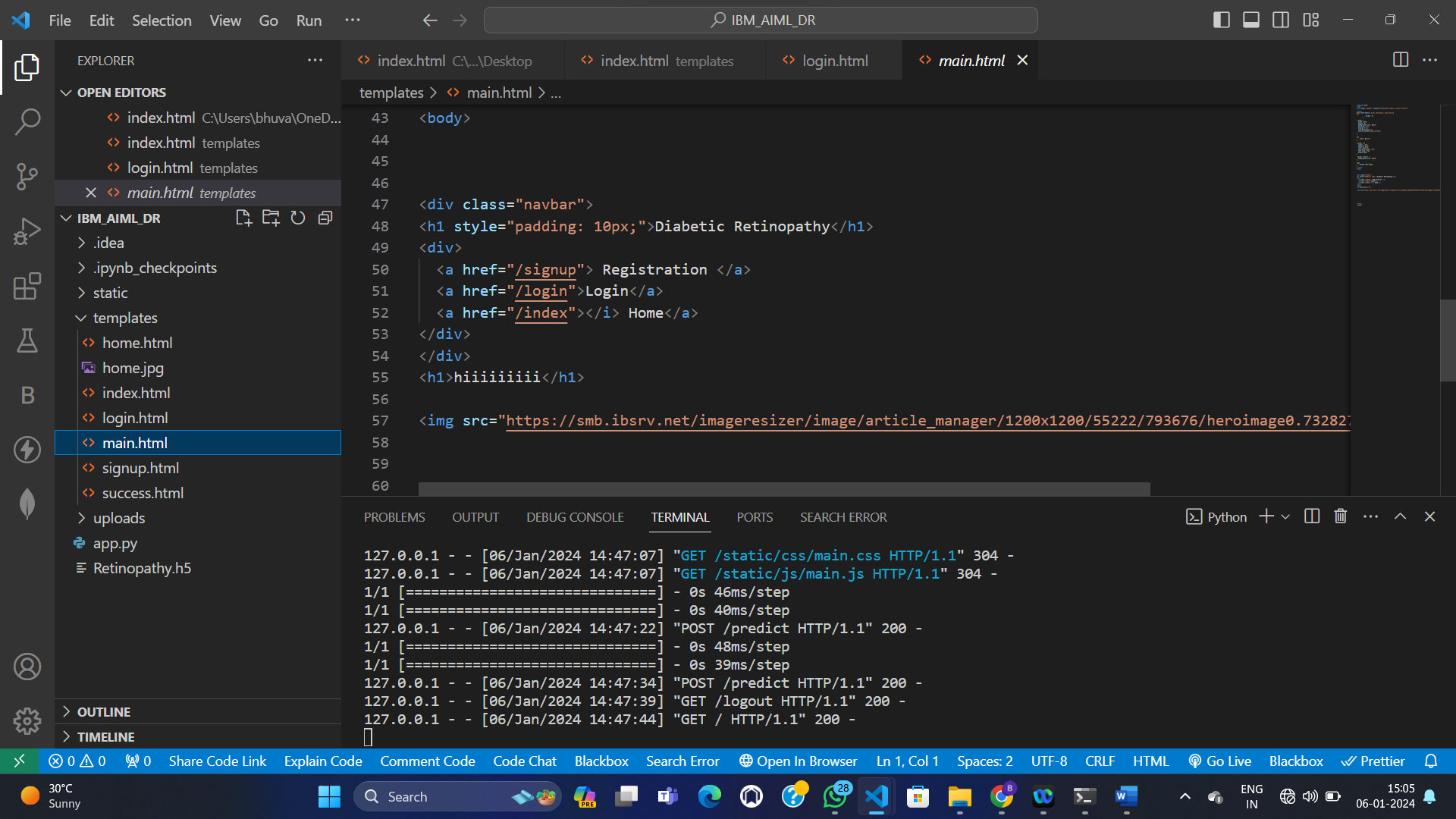
* This is logout page



**Build Python Code:**

****

****

****

**Run The Application:**

Open anaconda prompt from the start menu

Navigate to the folder where your python script is.

Now type “python app.py” command

Navigate to the localhost where you can view your web page.

Click on the predict button from the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

