**SOURCE CODE**

User Side views.py

**from** django.shortcuts **import** render, HttpResponse  
**from** .forms **import** UserRegistrationForm  
**from** .models **import** UserRegistrationModel,UserImagePredictinModel  
**from** django.contrib **import** messages  
**from** django.core.files.storage **import** FileSystemStorage  
**from** .utility.GetImageStressDetection **import** ImageExpressionDetect  
**from** .utility.MyClassifier **import** KNNclassifier  
**from** subprocess **import** Popen, PIPE  
**import** subprocess  
*# Create your views here.  
  
# Create your views here.***def** UserRegisterActions(request):  
 **if** request.method == **'POST'**:  
 form = UserRegistrationForm(request.POST)  
 **if** form.is\_valid():  
 print(**'Data is Valid'**)  
 form.save()  
 messages.success(request, **'You have been successfully registered'**)  
 form = UserRegistrationForm()  
 **return** render(request, **'UserRegistrations.html'**, {**'form'**: form})  
 **else**:  
 messages.success(request, **'Email or Mobile Already Existed'**)  
 print(**"Invalid form"**)  
 **else**:  
 form = UserRegistrationForm()  
 **return** render(request, **'UserRegistrations.html'**, {**'form'**: form})  
  
  
**def** UserLoginCheck(request):  
 **if** request.method == **"POST"**:  
 loginid = request.POST.get(**'loginname'**)  
 pswd = request.POST.get(**'pswd'**)  
 print(**"Login ID = "**, loginid, **' Password = '**, pswd)  
 **try**:  
 check = UserRegistrationModel.objects.get(loginid=loginid, password=pswd)  
 status = check.status  
 print(**'Status is = '**, status)  
 **if** status == **"activated"**:  
 request.session[**'id'**] = check.id  
 request.session[**'loggeduser'**] = check.name  
 request.session[**'loginid'**] = loginid  
 request.session[**'email'**] = check.email  
 print(**"User id At"**, check.id, status)  
 **return** render(request, **'users/UserHome.html'**, {})  
 **else**:  
 messages.success(request, **'Your Account Not at activated'**)  
 **return** render(request, **'UserLogin.html'**)  
 **except** Exception **as** e:  
 print(**'Exception is '**, str(e))  
 **pass** messages.success(request, **'Invalid Login id and password'**)  
 **return** render(request, **'UserLogin.html'**, {})  
  
  
**def** UserHome(request):  
 **return** render(request, **'users/UserHome.html'**, {})  
  
**def** UploadImageForm(request):  
 loginid = request.session[**'loginid'**]  
 data = UserImagePredictinModel.objects.filter(loginid=loginid)  
 **return** render(request, **'users/UserImageUploadForm.html'**, {**'data'**: data})  
  
**def** UploadImageAction(request):  
 image\_file = request.FILES[**'file'**]  
  
 *# let's check if it is a csv file* **if not** image\_file.name.endswith(**'.jpg'**):  
 messages.error(request, **'THIS IS NOT A JPG FILE'**)  
  
 fs = FileSystemStorage()  
 filename = fs.save(image\_file.name, image\_file)  
 *# detect\_filename = fs.save(image\_file.name, image\_file)* uploaded\_file\_url = fs.url(filename)  
 obj = ImageExpressionDetect()  
 emotion = obj.getExpression(filename)  
 username = request.session[**'loggeduser'**]  
 loginid = request.session[**'loginid'**]  
 email = request.session[**'email'**]  
 UserImagePredictinModel.objects.create(username=username,email=email,loginid=loginid,filename=filename,emotions=emotion,file=uploaded\_file\_url)  
 data = UserImagePredictinModel.objects.filter(loginid=loginid)  
 **return** render(request, **'users/UserImageUploadForm.html'**, {**'data'**:data})  
  
**def** UserEmotionsDetect(request):  
 **if** request.method==**'GET'**:  
 imgname = request.GET.get(**'imgname'**)  
 obj = ImageExpressionDetect()  
 emotion = obj.getExpression(imgname)  
 loginid = request.session[**'loginid'**]  
 data = UserImagePredictinModel.objects.filter(loginid=loginid)  
 **return** render(request, **'users/UserImageUploadForm.html'**, {**'data'**: data})  
  
**def** UserLiveCameDetect(request):  
 obj = ImageExpressionDetect()  
 obj.getLiveDetect()  
 **return** render(request, **'users/UserLiveHome.html'**, {})  
  
**def** UserKerasModel(request):  
 *# p = Popen(["python", "kerasmodel.py --mode display"], cwd='StressDetection', stdout=PIPE, stderr=PIPE)  
 # out, err = p.communicate()* subprocess.call(**"python kerasmodel.py --mode display"**)  
 **return** render(request, **'users/UserLiveHome.html'**, {})  
  
**def** UserKnnResults(request):  
 obj = KNNclassifier()  
 df,accuracy,classificationerror,sensitivity,Specificity,fsp,precision = obj.getKnnResults()  
 df.rename(columns={**'Target'**: **'Target'**, **'ECG(mV)'**: **'Time pressure'**, **'EMG(mV)'**: **'Interruption'**, **'Foot GSR(mV)'**: **'Stress'**, **'Hand GSR(mV)'**: **'Physical Demand'**, **'HR(bpm)'**: **'Performance'**, **'RESP(mV)'**: **'Frustration'**, }, inplace=**True**)  
 data = df.to\_html()  
 **return** render(request,**'users/UserKnnResults.html'**,{**'data'**:data,**'accuracy'**:accuracy,**'classificationerror'**:classificationerror,  
 **'sensitivity'**:sensitivity,**"Specificity"**:Specificity,**'fsp'**:fsp,**'precision'**:precision})

**user side forms.py**

**from** django **import** forms  
**from** .models **import** UserRegistrationModel  
  
  
**class** UserRegistrationForm(forms.ModelForm):  
 name = forms.CharField(widget=forms.TextInput(attrs={**'pattern'**: **'[a-zA-Z]+'**}), required=**True**, max\_length=100)  
 loginid = forms.CharField(widget=forms.TextInput(attrs={**'pattern'**: **'[a-zA-Z]+'**}), required=**True**, max\_length=100)  
 password = forms.CharField(widget=forms.PasswordInput(attrs={**'pattern'**: **'(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z]).{8,}'**,  
 **'title'**: **'Must contain at least one number and one uppercase and lowercase letter, and at least 8 or more characters'**}),  
 required=**True**, max\_length=100)  
 mobile = forms.CharField(widget=forms.TextInput(attrs={**'pattern'**: **'[56789][0-9]{9}'**}), required=**True**,  
 max\_length=100)  
 email = forms.CharField(widget=forms.TextInput(attrs={**'pattern'**: **'[a-z0-9.\_%+-]+@[a-z0-9.-]+\.[a-z]{2,}$'**}),  
 required=**True**, max\_length=100)  
 locality = forms.CharField(widget=forms.TextInput(), required=**True**, max\_length=100)  
 address = forms.CharField(widget=forms.Textarea(attrs={**'rows'**: 4, **'cols'**: 22}), required=**True**, max\_length=250)  
 city = forms.CharField(widget=forms.TextInput(  
 attrs={**'autocomplete'**: **'off'**, **'pattern'**: **'[A-Za-z ]+'**, **'title'**: **'Enter Characters Only '**}), required=**True**,  
 max\_length=100)  
 state = forms.CharField(widget=forms.TextInput(  
 attrs={**'autocomplete'**: **'off'**, **'pattern'**: **'[A-Za-z ]+'**, **'title'**: **'Enter Characters Only '**}), required=**True**,  
 max\_length=100)  
 status = forms.CharField(widget=forms.HiddenInput(), initial=**'waiting'**, max\_length=100)  
  
 **class** Meta():  
 model = UserRegistrationModel  
 fields = **'\_\_all\_\_'**

user side **Models.py**

**from** django.db **import** models  
  
*# Create your models here.***class** UserRegistrationModel(models.Model):  
 name = models.CharField(max\_length=100)  
 loginid = models.CharField(unique=**True**, max\_length=100)  
 password = models.CharField(max\_length=100)  
 mobile = models.CharField(unique=**True**, max\_length=100)  
 email = models.CharField(unique=**True**, max\_length=100)  
 locality = models.CharField(max\_length=100)  
 address = models.CharField(max\_length=1000)  
 city = models.CharField(max\_length=100)  
 state = models.CharField(max\_length=100)  
 status = models.CharField(max\_length=100)  
  
 **def** \_\_str\_\_(self):  
 **return** self.loginid  
  
 **class** Meta:  
 db\_table = **'UserRegistrations'  
class** UserImagePredictinModel(models.Model):  
 username = models.CharField(max\_length=100)  
 email = models.CharField(max\_length=100)  
 loginid = models.CharField(max\_length=100)  
 filename = models.CharField(max\_length=100)  
 emotions = models.CharField(max\_length=100000)  
 file = models.FileField(upload\_to=**'files/'**)  
 cdate = models.DateTimeField(auto\_now\_add=**True**)  
  
 **def** \_\_str\_\_(self):  
 **return** self.loginid  
  
 **class** Meta:  
 db\_table = **"UserImageEmotions"**

Image Classification:

**from** django.conf **import** settings  
**from** PyEmotion **import** \*  
**import** cv2 **as** cv  
**class** ImageExpressionDetect:  
 **def** getExpression(self,imagepath):  
 filepath = settings.MEDIA\_ROOT + **"\\"** + imagepath  
 PyEmotion()  
 er = DetectFace(device=**'cpu'**, gpu\_id=0)  
 *# Open you default camera  
 # img = cv.imread('test.jpg')  
 # cap = cv.VideoCapture(0)  
 # ret, frame = cap.read()* frame, emotion = er.predict\_emotion(cv.imread(filepath))  
 cv.imshow(**'Alex Corporation'**, frame)  
 cv.waitKey(0)  
 print(**"Hola Hi"**,filepath,**"Emotion is "**,emotion)  
 **return** emotion  
  
 **def** getLiveDetect(self):  
 print(**"Streaming Started"**)  
 PyEmotion()  
 er = DetectFace(device=**'cpu'**, gpu\_id=0)  
 *# Open you default camera* cap = cv.VideoCapture(0)  
 **while** (**True**):  
 ret, frame = cap.read()  
 frame, emotion = er.predict\_emotion(frame)  
 cv.imshow(**'Press Q to Exit'**, frame)  
 **if** cv.waitKey(1) & 0xFF == ord(**'q'**):  
 **break** cap.release()  
 cv.destroyAllWindows()

Deeplearning Model:

**import** numpy **as** np  
**import** argparse  
**import** cv2  
**from** keras.models **import** Sequential  
**from** keras.layers.core **import** Dense, Dropout, Flatten  
**from** keras.layers.convolutional **import** Conv2D  
**from** keras.optimizers **import** Adam  
**from** keras.layers.pooling **import** MaxPooling2D  
**from** keras.preprocessing.image **import** ImageDataGenerator  
**import** os  
os.environ[**'TF\_CPP\_MIN\_LOG\_LEVEL'**] = **'2'  
import** matplotlib **as** mpl  
mpl.use(**'TkAgg'**)  
**import** matplotlib.pyplot **as** plt  
  
*# command line argument*ap = argparse.ArgumentParser()  
ap.add\_argument(**"--mode"**,help=**"train/display"**)  
a = ap.parse\_args()  
mode = a.mode   
  
**def** plot\_model\_history(model\_history):  
 *"""  
 Plot Accuracy and Loss curves given the model\_history  
 """* fig, axs = plt.subplots(1,2,figsize=(15,5))  
 *# summarize history for accuracy* axs[0].plot(range(1,len(model\_history.history[**'acc'**])+1),model\_history.history[**'acc'**])  
 axs[0].plot(range(1,len(model\_history.history[**'val\_acc'**])+1),model\_history.history[**'val\_acc'**])  
 axs[0].set\_title(**'Model Accuracy'**)  
 axs[0].set\_ylabel(**'Accuracy'**)  
 axs[0].set\_xlabel(**'Epoch'**)  
 axs[0].set\_xticks(np.arange(1,len(model\_history.history[**'acc'**])+1),len(model\_history.history[**'acc'**])/10)  
 axs[0].legend([**'train'**, **'val'**], loc=**'best'**)  
 *# summarize history for loss* axs[1].plot(range(1,len(model\_history.history[**'loss'**])+1),model\_history.history[**'loss'**])  
 axs[1].plot(range(1,len(model\_history.history[**'val\_loss'**])+1),model\_history.history[**'val\_loss'**])  
 axs[1].set\_title(**'Model Loss'**)  
 axs[1].set\_ylabel(**'Loss'**)  
 axs[1].set\_xlabel(**'Epoch'**)  
 axs[1].set\_xticks(np.arange(1,len(model\_history.history[**'loss'**])+1),len(model\_history.history[**'loss'**])/10)  
 axs[1].legend([**'train'**, **'val'**], loc=**'best'**)  
 fig.savefig(**'plot.png'**)  
 plt.show()  
  
*# Define data generators*train\_dir = **'data/train'**val\_dir = **'data/test'**num\_train = 28709  
num\_val = 7178  
batch\_size = 64  
num\_epoch = 50  
  
train\_datagen = ImageDataGenerator(rescale=1./255)  
val\_datagen = ImageDataGenerator(rescale=1./255)  
  
train\_generator = train\_datagen.flow\_from\_directory(  
 train\_dir,  
 target\_size=(48,48),  
 batch\_size=batch\_size,  
 color\_mode=**"grayscale"**,  
 class\_mode=**'categorical'**)  
  
validation\_generator = val\_datagen.flow\_from\_directory(  
 val\_dir,  
 target\_size=(48,48),  
 batch\_size=batch\_size,  
 color\_mode=**"grayscale"**,  
 class\_mode=**'categorical'**)  
  
*# Create the model*model = Sequential()  
  
model.add(Conv2D(32, kernel\_size=(3, 3), activation=**'relu'**, input\_shape=(48,48,1)))  
model.add(Conv2D(64, kernel\_size=(3, 3), activation=**'relu'**))  
model.add(MaxPooling2D(pool\_size=(2, 2)))  
model.add(Dropout(0.25))  
  
model.add(Conv2D(128, kernel\_size=(3, 3), activation=**'relu'**))  
model.add(MaxPooling2D(pool\_size=(2, 2)))  
model.add(Conv2D(128, kernel\_size=(3, 3), activation=**'relu'**))  
model.add(MaxPooling2D(pool\_size=(2, 2)))  
model.add(Dropout(0.25))  
  
model.add(Flatten())  
model.add(Dense(1024, activation=**'relu'**))  
model.add(Dropout(0.5))  
model.add(Dense(7, activation=**'softmax'**))  
  
*# If you want to train the same model or try other models, go for this***if** mode == **"train"**:  
 model.compile(loss=**'categorical\_crossentropy'**,optimizer=Adam(lr=0.0001, decay=1e-6),metrics=[**'accuracy'**])  
  
 model\_info = model.fit\_generator(  
 train\_generator,  
 steps\_per\_epoch=num\_train // batch\_size,  
 epochs=num\_epoch,  
 validation\_data=validation\_generator,  
 validation\_steps=num\_val // batch\_size)  
  
 plot\_model\_history(model\_info)  
 model.save\_weights(**'model.h5'**)  
  
*# emotions will be displayed on your face from the webcam feed***elif** mode == **"display"**:  
 model.load\_weights(**'model.h5'**)  
  
 *# prevents openCL usage and unnecessary logging messages* cv2.ocl.setUseOpenCL(**False**)  
  
 *# dictionary which assigns each label an emotion (alphabetical order)* emotion\_dict = {0: **"Angry"**, 1: **"Disgusted"**, 2: **"Fearful"**, 3: **"Happy"**, 4: **"Neutral"**, 5: **"Sad"**, 6: **"Surprised"**}  
  
 *# start the webcam feed* cap = cv2.VideoCapture(0)  
 **while True**:  
 *# Find haar cascade to draw bounding box around face* ret, frame = cap.read()  
 facecasc = cv2.CascadeClassifier(**'haarcascade\_frontalface\_default.xml'**)  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
 faces = facecasc.detectMultiScale(gray,scaleFactor=1.3, minNeighbors=5)  
  
 **for** (x, y, w, h) **in** faces:  
 cv2.rectangle(frame, (x, y-50), (x+w, y+h+10), (255, 0, 0), 2)  
 roi\_gray = gray[y:y + h, x:x + w]  
 cropped\_img = np.expand\_dims(np.expand\_dims(cv2.resize(roi\_gray, (48, 48)), -1), 0)  
 prediction = model.predict(cropped\_img)  
 maxindex = int(np.argmax(prediction))  
 cv2.putText(frame, emotion\_dict[maxindex], (x+20, y-60), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE\_AA)  
  
 *# show the output frame* cv2.imshow(**"Alex Corporations Press Q to Exit"**, frame)  
 key = cv2.waitKey(1) & 0xFF  
   
 *# if the `q` key was pressed, break from the loop* **if** key == ord(**"q"**):  
 **break** cap.release()  
 cv2.destroyAllWindows()

Admin side Views.py

**from** django.shortcuts **import** render  
**from** django.contrib **import** messages  
**from** users.models **import** UserRegistrationModel,UserImagePredictinModel  
**from** .utility.AlgorithmExecutions **import** KNNclassifier  
  
*# Create your views here.***def** AdminLoginCheck(request):  
 **if** request.method == **'POST'**:  
 usrid = request.POST.get(**'loginid'**)  
 pswd = request.POST.get(**'pswd'**)  
 print(**"User ID is = "**, usrid)  
 **if** usrid == **'admin' and** pswd == **'admin'**:  
 **return** render(request, **'admins/AdminHome.html'**)  
 **elif** usrid == **'Admin' and** pswd == **'Admin'**:  
 **return** render(request, **'admins/AdminHome.html'**)  
 **else**:  
 messages.success(request, **'Please Check Your Login Details'**)  
 **return** render(request, **'AdminLogin.html'**, {})  
  
  
**def** AdminHome(request):  
 **return** render(request, **'admins/AdminHome.html'**)  
  
  
**def** ViewRegisteredUsers(request):  
 data = UserRegistrationModel.objects.all()  
 **return** render(request, **'admins/RegisteredUsers.html'**, {**'data'**: data})  
  
  
**def** AdminActivaUsers(request):  
 **if** request.method == **'GET'**:  
 id = request.GET.get(**'uid'**)  
 status = **'activated'** print(**"PID = "**, id, status)  
 UserRegistrationModel.objects.filter(id=id).update(status=status)  
 data = UserRegistrationModel.objects.all()  
 **return** render(request, **'admins/RegisteredUsers.html'**, {**'data'**: data})  
  
**def** AdminStressDetected(request):  
 data = UserImagePredictinModel.objects.all()  
 **return** render(request, **'admins/AllUsersStressView.html'**, {**'data'**: data})  
  
**def** AdminKNNResults(request):  
 obj = KNNclassifier()  
 df, accuracy, classificationerror, sensitivity, Specificity, fsp, precision = obj.getKnnResults()  
 df.rename(  
 columns={**'Target'**: **'Target'**, **'ECG(mV)'**: **'Time pressure'**, **'EMG(mV)'**: **'Interruption'**, **'Foot GSR(mV)'**: **'Stress'**,  
 **'Hand GSR(mV)'**: **'Physical Demand'**, **'HR(bpm)'**: **'Performance'**, **'RESP(mV)'**: **'Frustration'**, },  
 inplace=**True**)  
 data = df.to\_html()  
 **return** render(request, **'admins/AdminKnnResults.html'**,  
 {**'data'**: data, **'accuracy'**: accuracy, **'classificationerror'**: classificationerror,  
 **'sensitivity'**: sensitivity, **"Specificity"**: Specificity, **'fsp'**: fsp, **'precision'**: precision})

All urls.py

*"""StressDetection URL Configuration  
  
The `urlpatterns` list routes URLs to views. For more information please see:  
 https://docs.djangoproject.com/en/2.0/topics/http/urls/  
Examples:  
Function views  
 1. Add an import: from my\_app import views  
 2. Add a URL to urlpatterns: path('', views.home, name='home')  
Class-based views  
 1. Add an import: from other\_app.views import Home  
 2. Add a URL to urlpatterns: path('', Home.as\_view(), name='home')  
Including another URLconf  
 1. Import the include() function: from django.urls import include, path  
 2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))  
"""***from** django.contrib **import** admin  
**from** django.urls **import** path  
**from** StressDetection **import** views **as** mainView  
**from** users **import** views **as** usr  
**from** admins **import** views **as** admins  
**from** django.contrib.staticfiles.urls **import** static  
**from** django.contrib.staticfiles.urls **import** staticfiles\_urlpatterns  
**from** django.conf **import** settings  
  
urlpatterns = [  
 path(**'admin/'**, admin.site.urls),  
 path(**""**, mainView.index, name=**"index"**),  
 path(**"index/"**, mainView.index, name=**"index"**),  
 path(**"logout/"**, mainView.logout, name=**"logout"**),  
 path(**"UserLogin/"**, mainView.UserLogin, name=**"UserLogin"**),  
 path(**"AdminLogin/"**, mainView.AdminLogin, name=**"AdminLogin"**),  
 path(**"UserRegister/"**, mainView.UserRegister, name=**"UserRegister"**),  
  
 *### User Side Views* path(**"UserRegisterActions/"**, usr.UserRegisterActions, name=**"UserRegisterActions"**),  
 path(**"UserLoginCheck/"**, usr.UserLoginCheck, name=**"UserLoginCheck"**),  
 path(**"UserHome/"**, usr.UserHome, name=**"UserHome"**),  
 path(**"UploadImageForm/"**, usr.UploadImageForm, name=**"UploadImageForm"**),  
 path(**"UploadImageAction/"**, usr.UploadImageAction, name=**"UploadImageAction"**),  
 path(**"UserEmotionsDetect/"**, usr.UserEmotionsDetect, name=**"UserEmotionsDetect"**),  
 path(**"UserLiveCameDetect/"**, usr.UserLiveCameDetect, name=**"UserLiveCameDetect"**),  
 path(**"UserKerasModel/"**, usr.UserKerasModel, name=**"UserKerasModel"**),  
 path(**"UserKnnResults/"**, usr.UserKnnResults, name=**"UserKnnResults"**),  
  
 *### Admin Side Views* path(**"AdminLoginCheck/"**, admins.AdminLoginCheck, name=**"AdminLoginCheck"**),  
 path(**"AdminHome/"**, admins.AdminHome, name=**"AdminHome"**),  
 path(**"ViewRegisteredUsers/"**, admins.ViewRegisteredUsers, name=**"ViewRegisteredUsers"**),  
 path(**"AdminActivaUsers/"**, admins.AdminActivaUsers, name=**"AdminActivaUsers"**),  
 path(**"AdminStressDetected/"**, admins.AdminStressDetected, name=**"AdminStressDetected"**),  
 path(**"AdminKNNResults/"**, admins.AdminKNNResults, name=**"AdminKNNResults"**),  
  
  
]  
  
urlpatterns += staticfiles\_urlpatterns()  
urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

**Base.html**

<!DOCTYPE **html**>  
{%load static%}  
<**html lang="en"**>  
<**head**>  
<**title**>Stress Feelings</**title**>  
<**meta charset="utf-8"**>  
<**meta http-equiv="X-UA-Compatible" content="IE=edge"**>  
<**meta name="description" content="Unicat project"**>  
<**meta name="viewport" content="width=device-width, initial-scale=1"**>  
<**link rel="stylesheet" type="text/css" href="{%static 'styles/bootstrap4/bootstrap.min.css'%}"**>  
<**link href="{%static 'plugins/font-awesome-4.7.0/css/font-awesome.min.css'%}" rel="stylesheet" type="text/css"**>  
<**link rel="stylesheet" type="text/css" href="{%static 'plugins/OwlCarousel2-2.2.1/owl.carousel.css'%}"**>  
<**link rel="stylesheet" type="text/css" href="{%static 'plugins/OwlCarousel2-2.2.1/owl.theme.default.css'%}"**>  
<**link rel="stylesheet" type="text/css" href="{%static 'plugins/OwlCarousel2-2.2.1/animate.css'%}"**>  
<**link rel="stylesheet" type="text/css" href="{%static 'styles/main\_styles.css'%}"**>  
<**link rel="stylesheet" type="text/css" href="{%static 'styles/responsive.css'%}"**>  
</**head**>  
<**body**>  
  
<**div class="super\_container"**>  
  
 *<!-- Header -->* <**header class="header"**>  
  
 *<!-- Header Content -->* <**div class="header\_container"**>  
 <**div class="container"**>  
 <**div class="row"**>  
 <**div class="col"**>  
 <**div class="header\_content d-flex flex-row align-items-center justify-content-start"**>  
 <**div class="logo\_container"**>  
 <**a href="{%url 'index'%}"**>  
 <**div class="logo\_text"**>Stress Detection in IT<**span**> Professionals</**span**></**div**>  
 </**a**>  
 </**div**>  
 <**nav class="main\_nav\_contaner ml-auto"**>  
 <**ul class="main\_nav"**>  
 <**li**><**a href="{%url 'index'%}"**>Home</**a**></**li**>  
 <**li**><**a href="{%url 'UserLogin'%}"**>Users</**a**></**li**>  
 <**li**><**a href="{%url 'AdminLogin'%}"**>Admin</**a**></**li**>  
 <**li**><**a href="{%url 'UserRegister'%}"**>Registrations</**a**></**li**>  
  
 </**ul**>  
 </**nav**>  
  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
  
 *<!-- Header Search Panel -->* <**div class="header\_search\_container"**>  
 <**div class="container"**>  
 <**div class="row"**>  
 <**div class="col"**>  
 <**div class="header\_search\_content d-flex flex-row align-items-center justify-content-end"**>  
 <**form action="#" class="header\_search\_form"**>  
 <**input type="search" class="search\_input" placeholder="Search" required="required"**>  
 <**button class="header\_search\_button d-flex flex-column align-items-center justify-content-center"**>  
 <**i class="fa fa-search" aria-hidden="true"**></**i**>  
 </**button**>  
 </**form**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**header**>  
  
 {%block contents%}  
  
 {%endblock%}  
  
  
  
 <**footer class="footer"**>  
 <**div class="footer\_background" style="**background-image:url({%static 'images/footer\_background.png'%})**"**></**div**>  
 <**div class="container"**>  
 <**div class="row copyright\_row"**>  
 <**div class="col"**>  
 <**div class="copyright d-flex flex-lg-row flex-column align-items-center justify-content-start"**>  
 <**div class="cr\_text"**>*<!-- Link back to Colorlib can't be removed. Template is licensed under CC BY 3.0. -->*Copyright **&copy;**<**script**>document.write(new Date().getFullYear());</**script**> All rights reserved | This template is made with <**i class="fa fa-heart-o" aria-hidden="true"**></**i**> by <**a href="#" target="\_blank"**>Alex Corporation</**a**>  
*<!-- Link back to Colorlib can't be removed. Template is licensed under CC BY 3.0. -->*</**div**>  
  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**footer**>  
</**div**>  
  
<**script src="{%static 'js/jquery-3.2.1.min.js'%}"**></**script**>  
<**script src="{%static 'styles/bootstrap4/popper.js'%}"**></**script**>  
<**script src="{%static 'styles/bootstrap4/bootstrap.min.js'%}"**></**script**>  
<**script src="{%static 'plugins/greensock/TweenMax.min.js'%}"**></**script**>  
<**script src="{%static 'plugins/greensock/TimelineMax.min.js'%}"**></**script**>  
<**script src="{%static 'plugins/scrollmagic/ScrollMagic.min.js'%}"**></**script**>  
<**script src="{%static 'plugins/greensock/animation.gsap.min.js'%}"**></**script**>  
<**script src="{%static 'plugins/greensock/ScrollToPlugin.min.js'%}"**></**script**>  
<**script src="{%static 'plugins/OwlCarousel2-2.2.1/owl.carousel.js'%}"**></**script**>  
<**script src="{%static 'plugins/easing/easing.js'%}"**></**script**>  
<**script src="{%static 'plugins/parallax-js-master/parallax.min.js'%}"**></**script**>  
<**script src="{%static 'js/custom.js'%}"**></**script**>  
</**body**>  
</**html**>

Index.html

{%extends 'base.html'%}  
{%load static%}  
  
{%block contents%}  
  
<**div class="home"**>  
 <**div class="home\_slider\_container"**>  
  
 *<!-- Home Slider -->* <**div class="owl-carousel owl-theme home\_slider"**>  
  
 *<!-- Home Slider Item -->* <**div class="owl-item"**>  
 <**div class="home\_slider\_background" style="**background-image:url({%static 'images/home\_slider\_1.jpg'%})**"**></**div**>  
 <**div class="home\_slider\_content"**>  
 <**div class="container"**>  
 <**div class="row"**>  
 <**div class="col text-center"**>  
 <**div class="home\_slider\_title"**>Stress Detection in IT Professionals </**div**>  
 <**div class="home\_slider\_subtitle"**>by Image Processing and Machine Learning</**div**>  
 <**div class="home\_slider\_form\_container"**>  
 <**p**>  
 <**font color="Black"**>The main motive of our project is to detect stress in the IT professionals using vivid Machine learning and Image processing techniques .Our system is an upgraded version of the old stress detection systems which excluded the live detection and the personal counseling but this system comprises of live detection and periodic analysis of employees and detecting physical as well as mental stress levels in his/her by providing them with proper remedies for managing stress by providing survey form periodically. Our system mainly focuses on managing stress and making the working environment healthy and spontaneous for the employees and to get the best out of them during working hours.</**font**>  
 </**p**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
{%endblock%}

Index.html

{%extends 'base.html'%}  
{%load static%}  
  
{%block contents%}  
  
<**div class="home"**>  
 <**div class="home\_slider\_container"**>  
  
 *<!-- Home Slider -->* <**div class="owl-carousel owl-theme home\_slider"**>  
  
 *<!-- Home Slider Item -->* <**div class="owl-item"**>  
 <**div class="home\_slider\_background" style="**background-image:url({%static 'images/home\_slider\_1.jpg'%})**"**></**div**>  
 <**div class="home\_slider\_content"**>  
 <**div class="container"**>  
 <**div class="row"**>  
 <**div class="col text-center"**>  
 <**div class="home\_slider\_title"**>User Register Form </**div**>  
 <**center**>  
 <**form action="{%url 'UserRegisterActions'%}" method="POST" class="text-primary comment\_form"**" style="width:100%">  
  
 {% csrf\_token %}  
 <**table**>  
 <**tr**>  
 <**td class="text-primary"**>User Name</**td**>  
 <**td**>{{form.name}}</**td**>  
 </**tr**>  
 <**tr**>  
 <**td**>Login ID</**td**>  
 <**td**>{{form.loginid}}</**td**>  
 </**tr**>  
 <**tr**>  
 <**td**>Password</**td**>  
 <**td**>{{form.password}}</**td**>  
 </**tr**>  
 <**tr**>  
 <**td**>Mobile</**td**>  
 <**td**>{{form.mobile}}</**td**>  
 </**tr**>  
 <**tr**>  
 <**td**>email</**td**>  
 <**td**>{{form.email}}</**td**>  
 </**tr**>  
 <**tr**>  
 <**td**>Locality</**td**>  
 <**td**>{{form.locality}}</**td**>  
 </**tr**>  
 <**tr**>  
 <**td**>Address</**td**>  
 <**td**>{{form.address}}</**td**>  
 </**tr**>  
 <**tr**>  
 <**td**>City</**td**>  
 <**td**>{{form.city}}</**td**>  
 </**tr**>  
 <**tr**>  
 <**td**>State</**td**>  
 <**td**>{{form.state}}</**td**>  
 </**tr**>  
 <**tr**>  
  
 <**td**>{{form.status}}</**td**>  
 </**tr**>  
<**tr**><**td**></**td**>  
 <**td**><**button type="submit" class="comment\_button trans\_200"**>Register</**button**></**td**>  
 </**tr**>  
  
  
 {% if messages %}  
 {% for message in messages %}  
 <**font color='GREEN'**> {{ message }}</**font**>  
 {% endfor %}  
 {% endif %}  
  
 </**table**>  
  
 </**form**>  
 </**center**>  
  
  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
</**div**>  
{%endblock%}

User login .html

{%extends 'base.html'%}  
{%load static%}  
  
{%block contents%}  
  
<**div class="home"**>  
 <**div class="home\_slider\_container"**>  
  
 *<!-- Home Slider -->* <**div class="owl-carousel owl-theme home\_slider"**>  
  
 *<!-- Home Slider Item -->* <**div class="owl-item"**>  
 <**div class="home\_slider\_background" style="**background-image:url({%static 'images/home\_slider\_1.jpg'%})**"**></**div**>  
 <**div class="home\_slider\_content"**>  
 <**div class="container"**>  
 <**div class="row"**>  
 <**div class="col text-center"**>  
 <**div class="home\_slider\_title"**>User Login Form </**div**>  
 <**div class="home\_slider\_subtitle"**></**div**>  
 <**div class="home\_slider\_form\_container"**>  
 <**p**>  
 <**center**>  
 <**form action="{%url 'UserLoginCheck'%}" method="POST" class="text-primary" style="**width:100%**"**>  
 {% csrf\_token %}  
 <**table**>  
 <**div class="form-group row"**>  
 <**div class="col-md-12"**>  
 <**input type="text" class="form-control" name="loginname" required placeholder="Enter Login Id"**>  
 </**div**>  
 </**div**>  
 <**div class="form-group row"**>  
 <**div class="col-md-12"**>  
 <**input type="password" class="form-control" name="pswd" required placeholder="Enter password"**>  
 </**div**>  
 </**div**>  
  
  
 <**tr**>  
 <**td**>  
 <**button class="btn btn-block btn-primary text-white py-3 px-5" style="**margin-left:20%;**"  
 type="submit"**>  
 Login  
 </**button**>  
 </**td**>  
 </**tr**>  
  
 {% if messages %}  
 {% for message in messages %}  
 <**font color='GREEN'**> {{ message }}</**font**>  
 {% endfor %}  
 {% endif %}  
  
 </**table**>  
  
 </**form**>  
 </**center**>  
  
 </**p**>  
  
  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
{%endblock%}

KNN Results.html

{%extends 'users/userbase.html'%}  
{%load static%}  
  
{%block contents%}  
<**div class="features"**>  
 <**div class="container"**>  
 <**div class="row"**>  
 <**div class="col"**>  
 <**div class="section\_title\_container text-center"**>  
 <**h2 class="section\_title"**>Knn Algorithm Results</**h2**>  
 <**h3**>Accuarcy <**font color="Green"**>{{accuracy}}</**font**></**h3**> <**br**/>  
 <**h3**>Classification Error <**font color="Green"**>{{classificationerror}}</**font**></**h3**>  
 <**h3**>Sensitivity <**font color="Green"**>{{sensitivity}}</**font**></**h3**>  
 <**h3**>Specificity <**font color="Green"**>{{Specificity}}</**font**></**h3**>  
 <**h3**>False positive rate Error <**font color="Green"**>{{fsp}}</**font**></**h3**>  
 <**h3**>Precision <**font color="Green"**>{{precision}}</**font**></**h3**>  
  
  
 </**div**>  
 <**center**>  
 <**h2**>Results table</**h2**>  
 <**font color="Black"**>  
 {{data | safe}}  
 </**font**>  
 </**center**>  
  
  
 </**div**>  
 </**div**>  
 <**div class="row features\_row"**>  
  
  
 </**div**>  
 </**div**>  
 </**div**>  
{%endblock%}