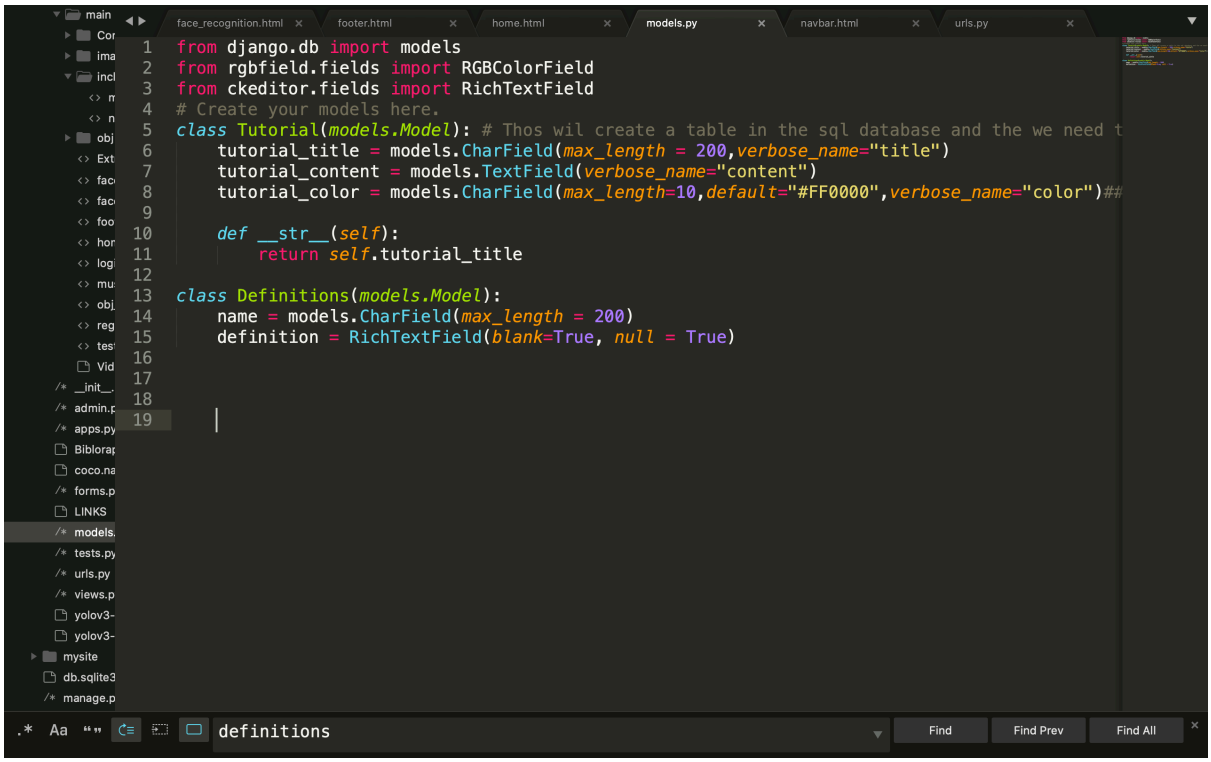
Criterion C: Development

Coding Wizards website allows users to learn about various techniques related to Artificial Intelligence. The website starts with displaying the homepage consisting various AI techniques and nav-bar with login, register, feedback and extras options. I have used Django and materialize.css to design the website. Django provides me with model-view-controller architecture and makes it easier for me to build a scalable website as it automatically handles the database, also it provides a admin page to add/remove item from database. So first I would like to make this architecture clearer

Models:



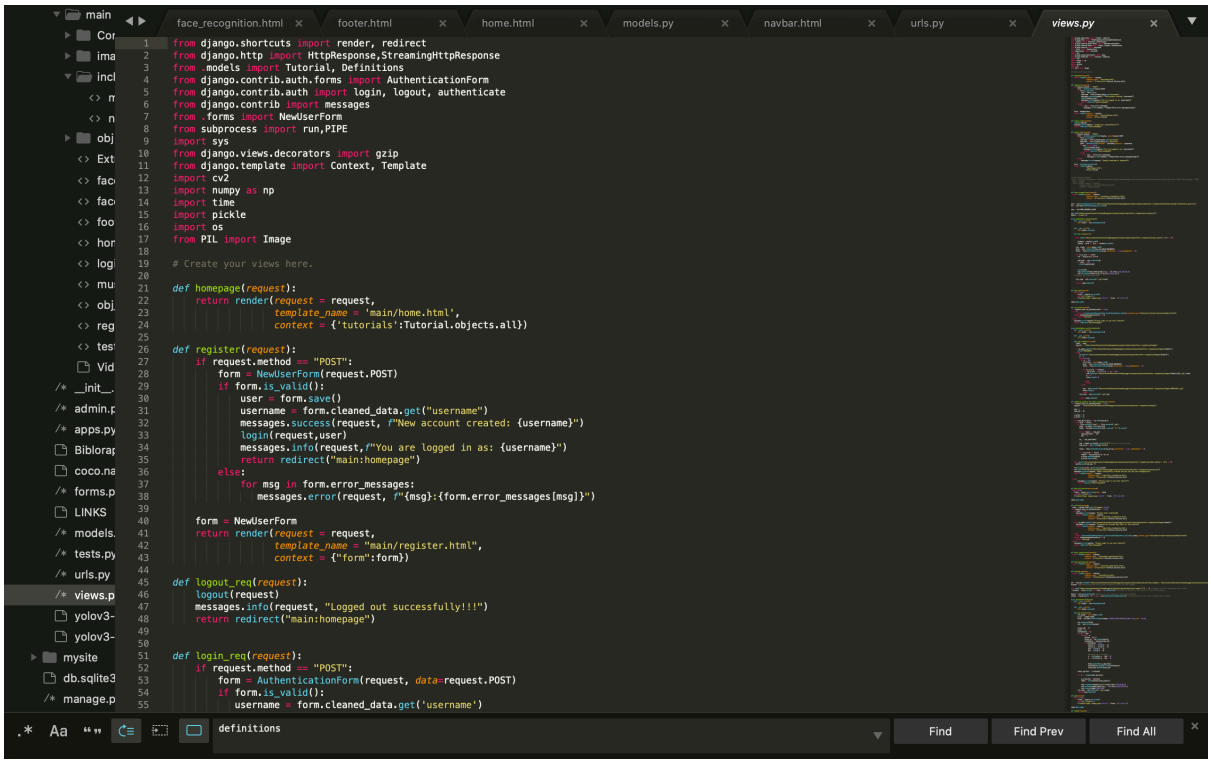
models.py file contains all the classes for the database, class Tutorial(models.Model) and class Definitions(models.Model) both inherits from models.Model to use it's methods. These classes have variables to represent different columns of the database for data of different data types. Variable's name is used to access individual item stored in the column

Controller:

urls.py file creates path for all the actions, it is a bridge between templates (where user interacts) and views.py. Template makes request from certain functionalities or links available in urls.py file which then calls for respective functions/actions in views.py file.

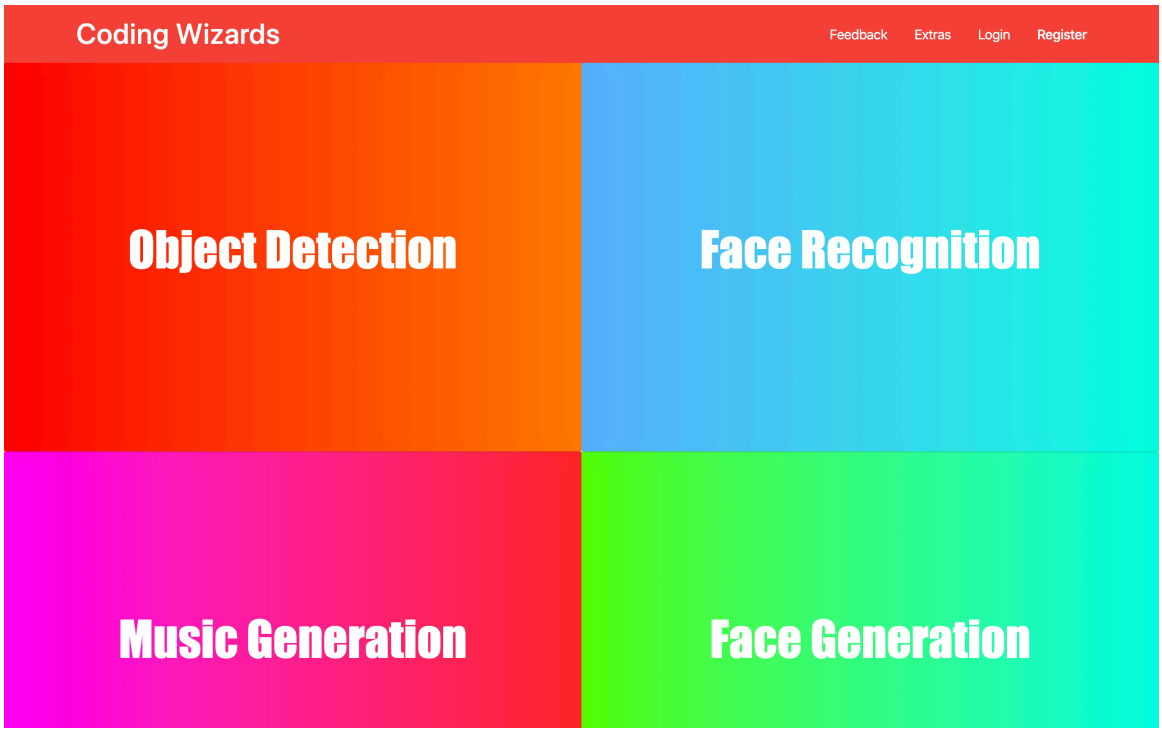
```python
from django.urls import path
from .import views

app_name = 'main'

urlpatterns = [


    path("obj_detection",views.obj_detection, name="obj_detection"),
    path("face_recognition",views.face_recognition, name="face_recognition"),
    path("music_generation",views.music_generation, name="music_generation"),
    path("face_generation",views.face_generation, name="face_generation"),
    path("", views.homepage, name = "homepage"),
    path("homepage", views.homepage, name = "homepage"),
    path("register", views.register, name="register"),
    path("logout",views.logout_req,name = "logout"),
    path("login",views.login_req,name = "login_req"),
    path("Extras",views.extras,name = 'extras'),
    path("object",views.index, name="index"),
    path("collect",views.collect, name="collect"),
    path("run_face",views.run_face, name="run_face"),
    path("updating_weights_for_face_recognition",views.updating_weights_for_face_recognition, name="updating_weights_for_face_recognition"),



]
```

Views:



views.py file performs functions asked by the template (users) such as login/logout, face recognition, object detection, displaying messages etc. It also provides template with the relevant html files and data from models.

Home Page:



Homepage has four options which users can choose from to navigate to whatever AI technique they might be interested in. The design adapts to various screen sizes using materialize.css.

Nav-bar and Footer:

As the nav-bar and footer are available in all the pages, I have common code for it in nav-bar.html and footer.html files.

face_recognition.html | footer.html | messages.html | navbar.html | urls.py

```
<head>
   <link href = "../home.css" rel="stylesheet">

</head>

  <div>
    <nav style="background-color: #f44336;">
      <div class="nav-wrapper" style="box-shadow: 0px 3px #9E9E9E;">
        <a href="homepage" class="brand-logo" style = "padding-left:80px"><strong>Coding Wizards</strong></a>
        <ul id="nav-mobile" class="right hide-on-med-and-down">
          <li><a href="https://docs.google.com/forms/d/
          e/1FAIpQLSeWhAG7ZTm52vlr18XwQ1kUNzpNOYZryMal9JG6ypslxoDCdg/viewform?usp=sf_link">Feedback</a></li>
          <li><a href="Extras">Extras</a></li>

          {% if user.is_authenticated %}

          <li><strong><a href="#!">{{user.username}}</a></strong></li>
          <li><a href="logout" style = "padding-right:80px">Logout</a></li>

          {% else %}
           <li><a href="login">Login</a></li>
          <li><strong><a href="register" style = "padding-right:80px">Register</a></strong></li>

           {% endif %}
        </ul>
      </div>
    </nav>
<div>
```

.* Aa “” TRAIN Find Find Prev Find All

Navbar contains buttons for login, register, logout, feedback and extras. It also has code to check for login status to hide or display login/logout options

({% if user.is_authenticated %})

This is the End!!!

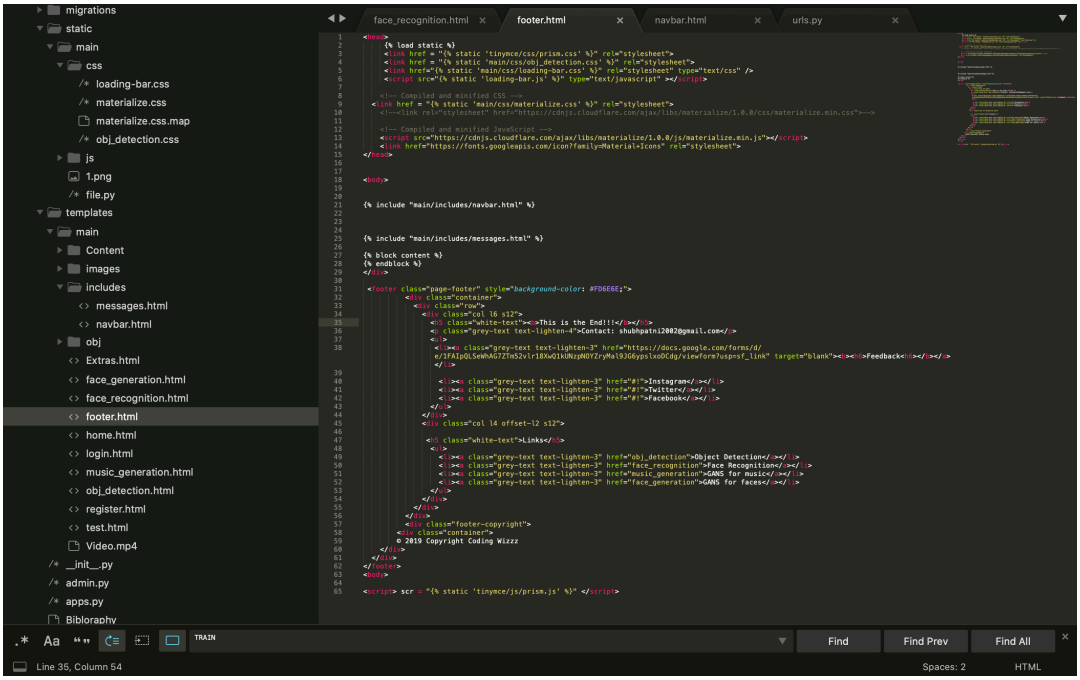
Contact: codingwizards@gmail.com

Feedback

Instagram
Twitter
Facebook

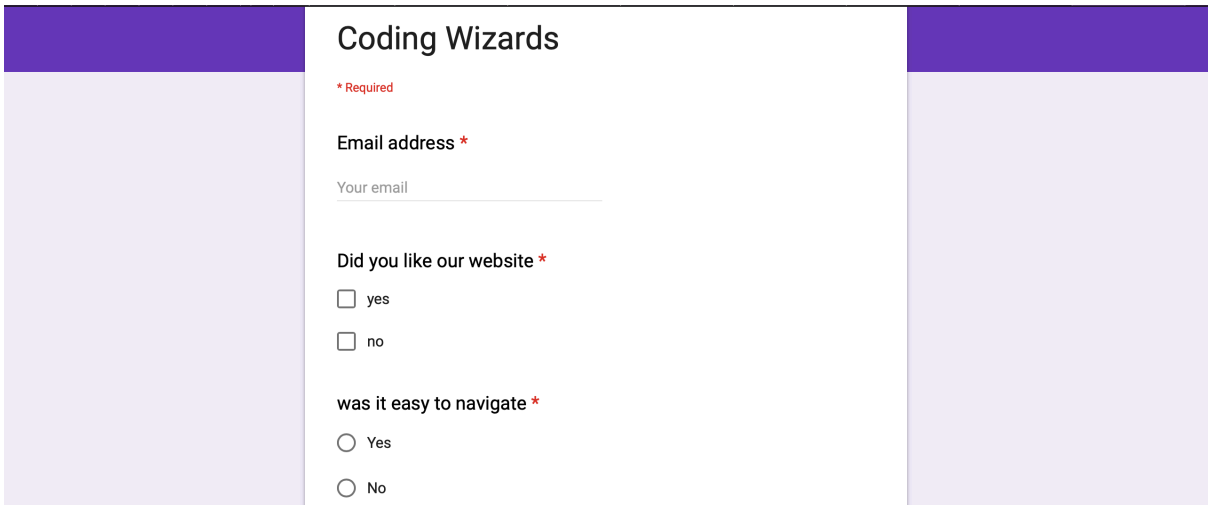
Links

Object Detection
Face Recognition
GANS for music
GANS for faces

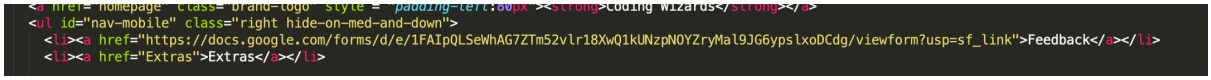


In footer.html file a line calls for navbar.html and messages.html to make the code more usable and modular *({% include "main/includes/navbar.html" %}) ({% include "main/includes/messages.html" %})*. .All the other HTML files include navbar.html and footer.html to use their features.

Feedback:

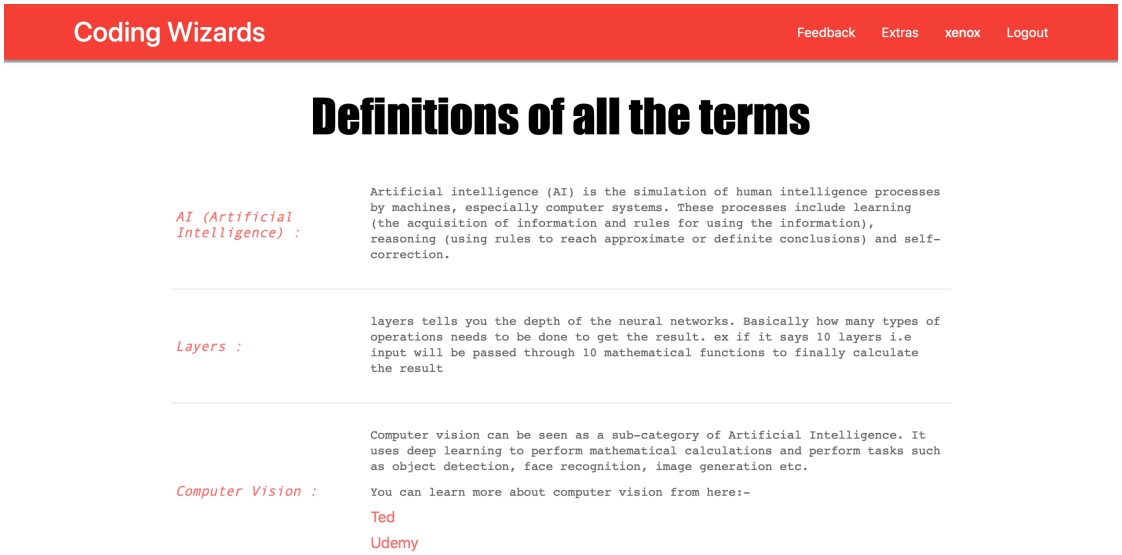


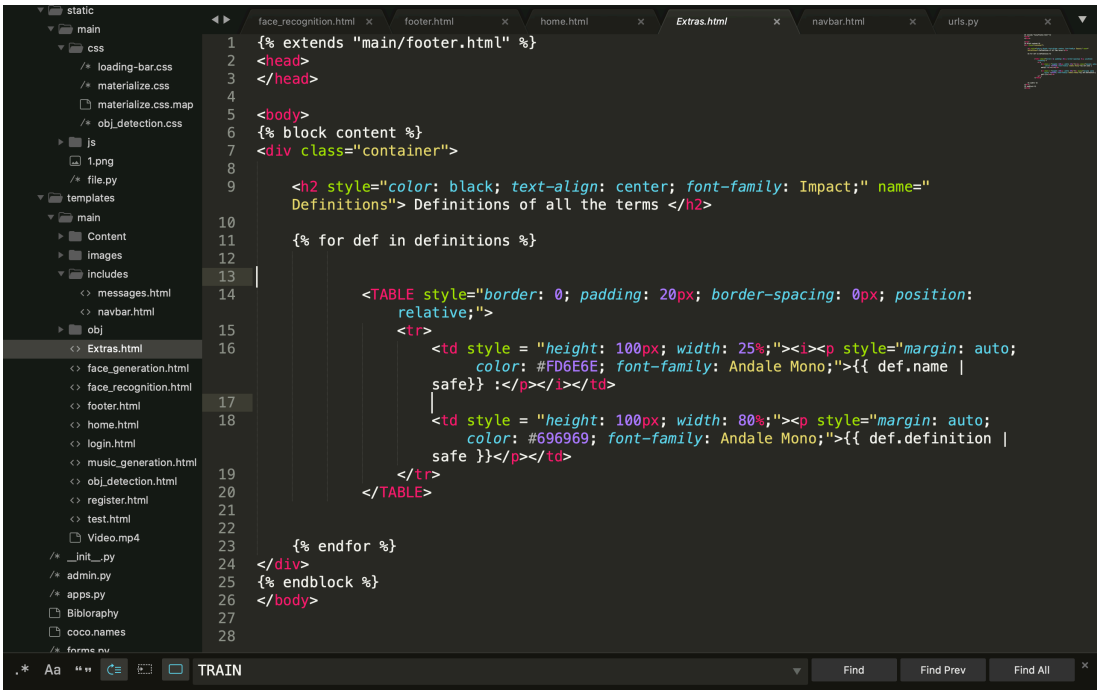
Figure(4): Google Forms



Feedback form by Google allows my client to know what can be improved in the website and keeps him in touch to his students and other users of the website.

Extras:





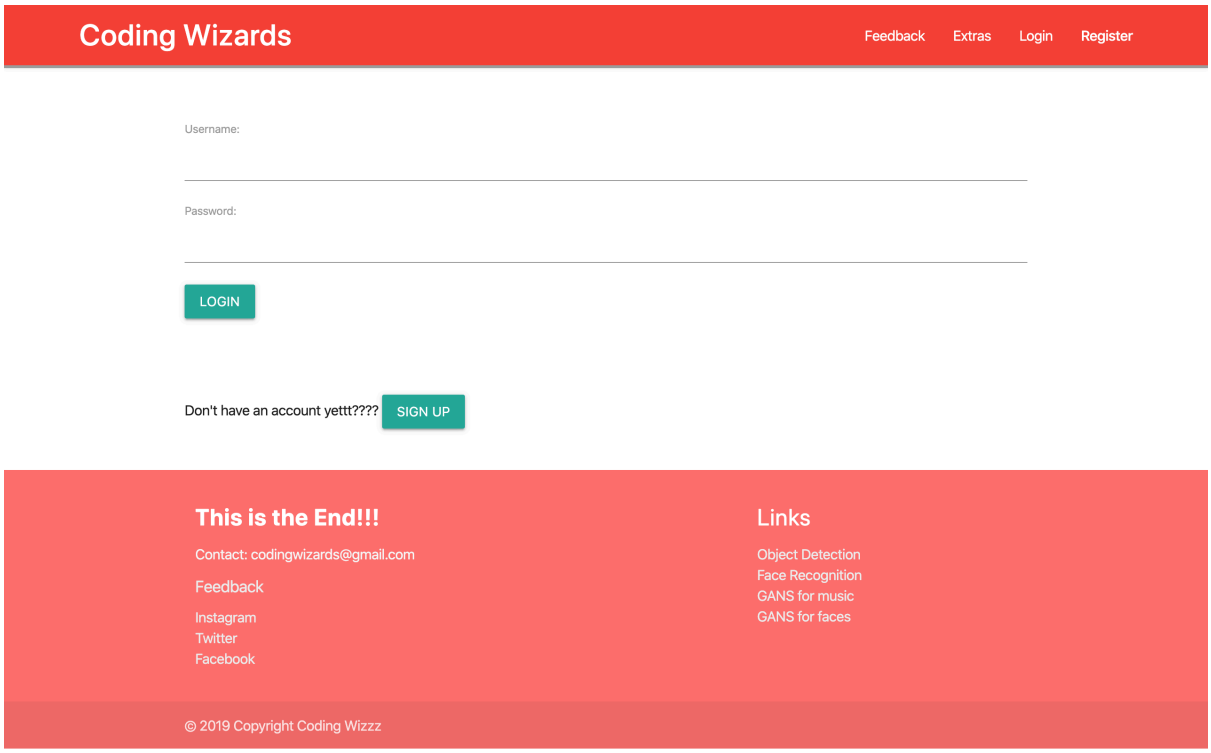
Extras contains definitions of all the hard terminologies used in the website and allows the user to learn more about them. It has code *({% for def in definitions %})* to loop through all

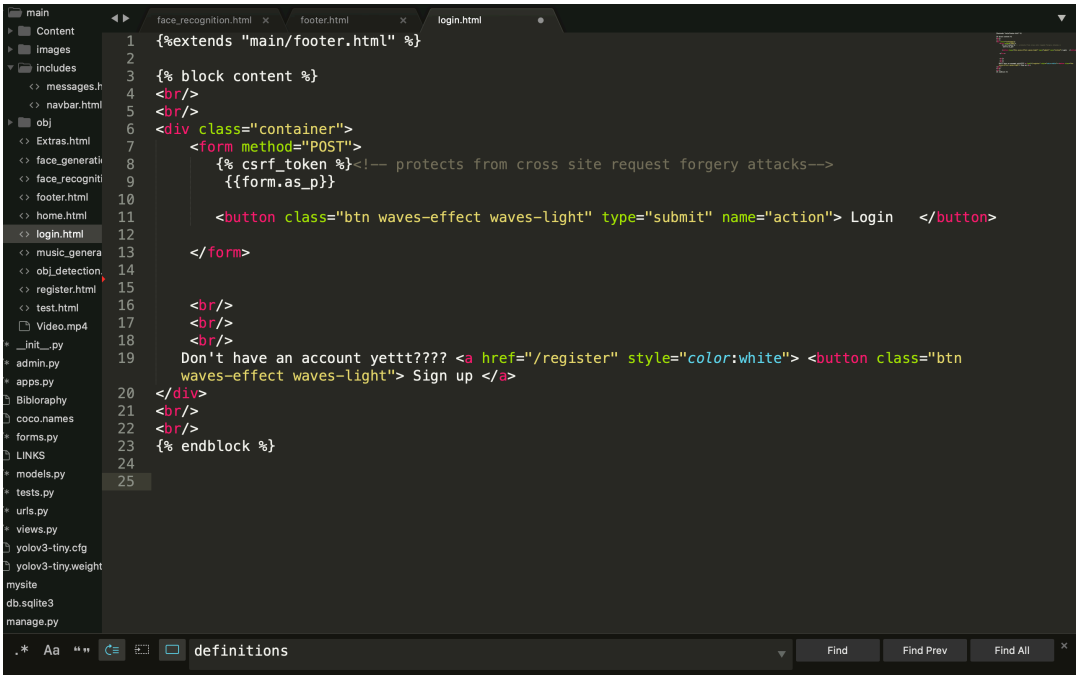
the definitions stored in Definitions model in the database and then it displays all of them with same structure and design inside the <*Table*> tag.

```
def extras(request):
    return render(request = request,
                  template_name = 'main/Extras.html',
                  context = {'definitions':Definitions.objects.all})
```

extras() function is called when a user clicks on any definition and this function passes the data from Definitions model to the extras.html to display them.

Login:



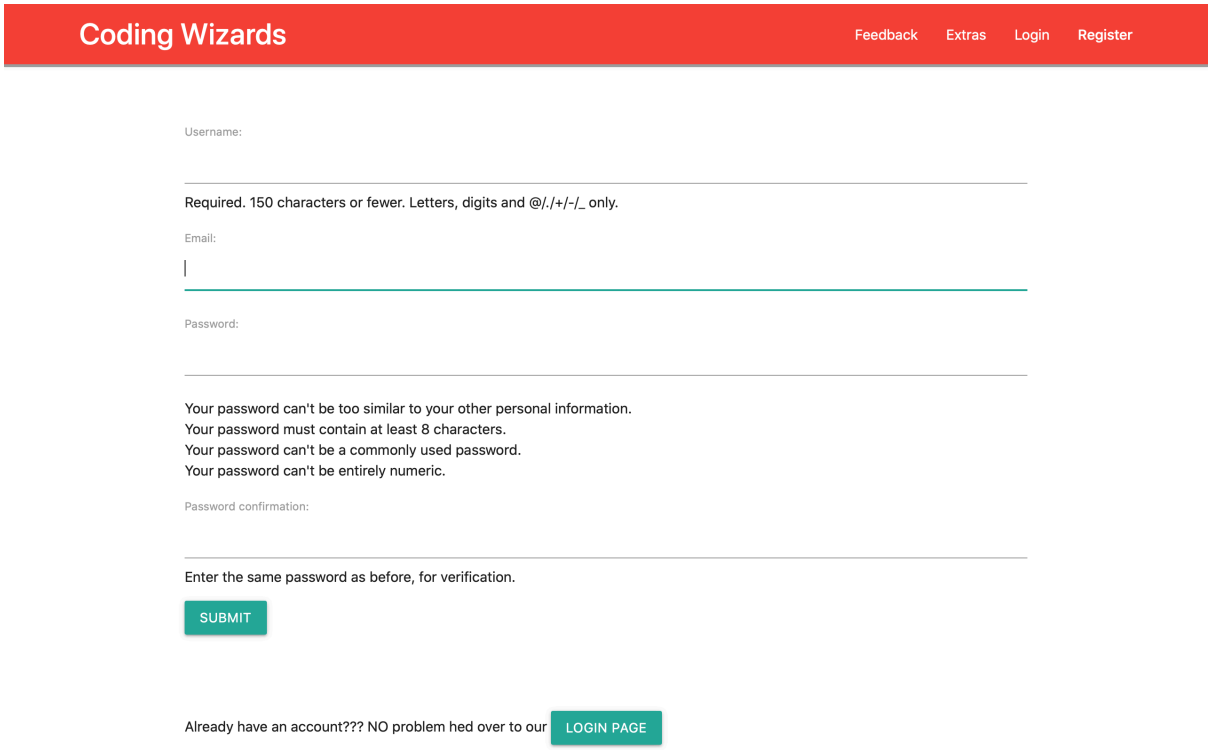


login.html allows user to login to the website. *<form>* tag takes the input from the user and sends it to view.py file to login the user. *{% csrf_token %}* protects against all cross site request forgery attacks.

```
def login_req(request):
    if request.method == "POST":
        form = AuthenticationForm(request, data=request.POST)
        if form.is_valid():
            username = form.cleaned_data.get('username')
            password = form.cleaned_data.get('password')
            user = authenticate(username = username,password = password)
            if user is not None:
                login(request,user)
                messages.info(request,f"You are logged in as: {username}")
                return redirect("main:homepage")
            else:
                for msg in form.error_messages:
                    messages.error(request, f"{msg}:{form.error_messages[msg]}")
        else:
            messages.error(request, "invalid username or password")

    form = AuthenticationForm()
    return render(request,
                  "main/login.html",
                  {"form":form})
```

login_req function checks the user login credentials and authenticates them if the info exists in database. Otherwise a message is shown to the user stating the issues in authentication.

Register:



register.html | views.py | music_generation.html | obj_detection.html | tets.py

```
{%extends "main/footer.html" %}

{% block content %}
<br/>
<br/>
<div class="container">
    <form method="POST">

        {% csrf_token %} <!-- protects from cross site request forgery attacks-->
        {{form.as_p}}

       <button class="btn waves-effect waves-light" type="submit" name="action"> Submit</button>

    </form>


    <br/>
    <br/>
    <br/>
    Already have an account??? NO problem hed over to our <a href="login"> <button class="btn waves-effect
    waves-light" type="submit" name="action"> Login page </button></a>
</div>
<br/>
<br/>
{% endblock %}
```

The html file has code to create a temple to let user register, the form tag uses Django's default form layout with materialize.css design to get the user's username, email and password.

```python
def register(request):
    if request.method == "POST":
        form = NewUserForm(request.POST)
        if form.is_valid():
            user = form.save()
            username = form.cleaned_data.get("username")
            messages.success(request, f"New account created: {username}")
            login(request,user)
            messages.info(request,f"You are logged in as: {username}")
            return redirect("main:homepage")
        else:
            for msg in form.error_messages:
                messages.error(request, f"{msg}:{form.error_messages[msg]}")

    form = NewUserForm
    return render(request = request,
                  template_name = "main/register.html",
                  context = {"form":form})
```

This function takes the data from register.html and checks if the data is authentic using *if form.is_valid()*, otherwise it displays an error message to informing the user about the error *messages.error()*

Anchors:

```
<div style="width: 200px; position: fixed; padding: 20px; margin: 20px; font-size: 17px;" class="sidebar">

    <p><a href="#definition" style="color: #696969;">Definition</a>
    <p><a href="#detect_objects" style="color: #696969;">Detect Objects</a>
    <p><a href="#CNN" style="color: #696969;">CNN</a>
    <p><a href="#various_object_detection_techniques" style="color: #696969;">Object detection techniques</a>
    <p><a href="#details" style="color: #696969;">Details & Code</a>


</div>
```

Anchors are internal links which allow users to easily navigate to various Headlines within the page. Href attribute of the <a> tag contains the name of the portion of page which user wishes to visit. { *<a href="#definition"*}

Transfer Learning:

```
face = cv2.CascadeClassifier('/Users/xenox/Documents/Coaaadinggg/site/mysite/mysite/main/Face recognition/data/haarcascade_frontalface_alt2.xml')
rec = cv2.face.LBPHFaceRecognizer_create()

font = cv2.FONT_HERSHEY_PLAIN

rec.read("/Users/xenox/Documents/Coaaadinggg/site/mysite/mysite/main/Face recognition/trained.yml")
labels = {'name':1}
```

```
    rec.train(x_train, np.array(y_train))
    rec.save("/Users/xenox/Documents/Coaaadinggg/site/mysite/mysite/main/Face recognition/trained.yml")
    messages.success(request, "Model successfully trained now you can use Face Recognition")
    return render(request = request,
                  template_name = 'main/face_recognition.html',
                  context = {'tutorials':Tutorial.objects.all})
else:
        messages.error(request, "Please Login to use this feature")
        return redirect("main:homepage")
```

It is a technique which uses per-trained models to do the classification, the model already contains features which can be modified by further training on specific data sets, *rec.train()* function trains the model and *rec.save()* saves it. It reduces training time, computation and creates better and more useful models.

Object Detection:

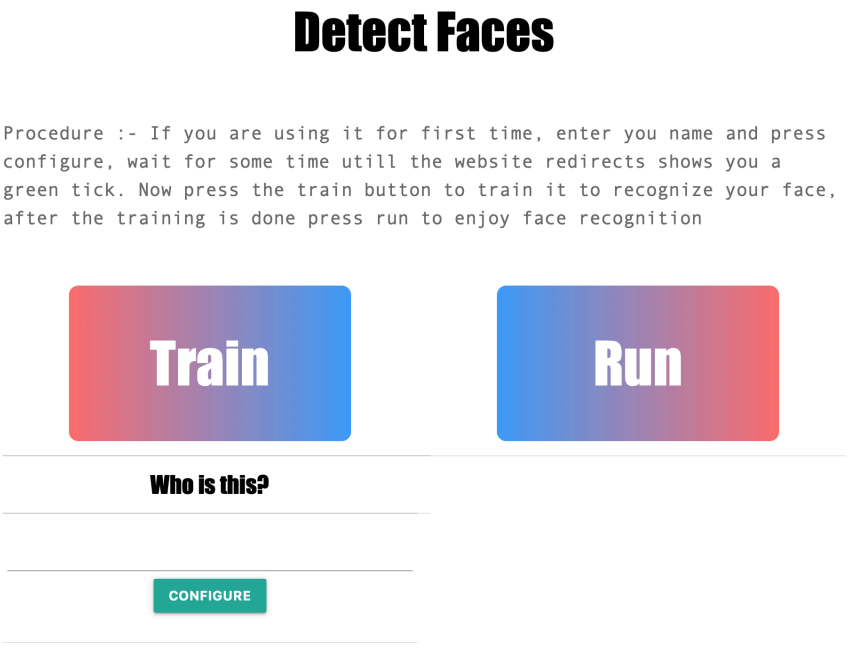
```
def gen(camera):
    while True:
        frame = camera.get_frame()
        yield(b'--frame\r\n'
        b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

@gzip.gzip_page

def index(request):
  if request.user.is_authenticated == True:
    try:
        return StreamingHttpResponse(gen(VideoCamera()),content_type="multipart/x-mixed-replace;boundary=frame")
    except HttpResponseServerError as e:
        return "aborted"
  else:
    messages.error(request,"Please Login to use this feature")
    return redirect("main:homepage")
```

Run button in the browser initiates the object detection by calling a function (index()) in views.py. Object detection uses lite version of YOLOv3 (You Only Look Once) algorithm and pre-trained weights provided by YOLO are used. Lite version of the main algorithm is used to make it faster and to be able to recognise objects in real time.

```
    def get_frame(self):
        ret,image = self.video.read()
        h,w,c = image.shape
        blob = cv2.dnn.blobFromImage(image,0.00392,(320,320),(0,0,0),True,crop = False)

        net.setInput(blob)
        out = net.forward(output)

        class_ids = []
        boxex = []
        confidences = []
        for o in out:
            for e in o:
                points = e[5:]
                class_id = np.argmax(points)
                confidence = points[class_id]
                if confidence > 0.6:
                    center_x = int(e[0] * w)
                    center_y = int(e[1] * h)
                    wid = int(e[2] * w)
                    hei = int(e[3] * h)

                    #Drawing the rectangles
                    x = int(center_x - wid / 2)
                    y = int(center_y - hei / 2)


                    boxex.append([x,y,wid,hei])
                    confidences.append(float(confidence))
                    class_ids.append(class_id)

        total_objects = len(boxex)

        for i in range(total_objects):

            x,y,wid,hei = boxex[i]
            label = str(classes[class_ids[i]])

            cv2.rectangle(image,(x,y),(x+wid,y+hei),(255,0,0),2)
            cv2.putText(image,label,(x,y + 20),font,1,(0,255,0),2)
            cv2.resize(image,(300,300))
        ret,jpeg = cv2.imencode('.jpg',image)
        return jpeg.tobytes()
```

After the object has been detected a rectangle is drawn around it with a text field on top stating which object it is. Then the image is passed in binary form and then displayed in the website.

Face Recognition:



It can detect faces of multiple people In a video frame and label them according to their names. I have used transfer learning for this technique to use a pre-trained face recognition model to save training time and get better predictions.

```
class VideoCamera_collect(object):
    def __init__(self):
        self.video = cv2.VideoCapture(0)

    def __del__(self):
        self.video.release()

    def get_frame(self,name):
      label = name
      img_dir = "/Users/xenox/Documents/Coaaadinggg/site/mysite/mysite/main/Face recognition/Images"

      if os.path.isdir(f'/Users/xenox/Documents/Coaaadinggg/site/mysite/mysite/main/Face recognition/Images/{label}'):
        print("WORKINNN")
      else:
          os.mkdir(f'/Users/xenox/Documents/Coaaadinggg/site/mysite/mysite/main/Face recognition/Images/{label}')
          a = 1
          while(True):
            if a <= 20:
              ret,frame = self.video.read()
              gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
              faces = face.detectMultiScale(gray,scaleFactor = 1.5,minNeighbors = 5)

              for (x,y,w,h) in (faces):
                if (x,y,w,h) != (0,0,0,0) and a <= 20:
                  cv2.imwrite(f'/Users/xenox/Documents/Coaaadinggg/site/mysite/mysite/main/Face recognition/Images/{label}/{a}.jpg',frame)
                  a += 1
                  time.sleep(0.5)

                else:
                  break
            else:

              img = cv2.imread("/Users/xenox/Documents/Coaaadinggg/site/mysite/mysite/main/Face recognition/Images/COMPLETED.jpg")
              time.sleep(1)
              break
          ret,jpeg = cv2.imencode('.jpg',img)

          return jpeg.tobytes()
```

New user has to input their name and press configure to start the data collection process, in this process images of the user is taken through webcam and saved in a folder with name as name entered by the user. *get_frame()* function first checks if the folder exists, otherwise the folder is created using *os.mkdir(),* now face is detected in each frame using *faces = face.detectMultiScale* and if there is a face then image is saved inside the folder using *cv2.imwrite().* After the data collection process user is shown a completed sign to indicate that the process has been completed.

```
def updating_weights_for_face_recognition(request):
  if request.user.is_authenticated == True:
    img_dir = "/Users/xenox/Documents/Coaaadinggg/site/mysite/mysite/main/Face recognition/Images/"

    ide =0
    lab_ids = {}

    x_train = []
    y_train = []

    for root,dirs,files in os.walk(img_dir):
      for file in files:
        if file.endswith('png') or file.endswith('jpg'):
          path = os.path.join(root,file)
          label = os.path.basename(root).replace(" ","-").lower()

          if not label in lab_ids:
            lab_ids[label] = ide
            ide += 1

          id_ = lab_ids[label]

          img = Image.open(path).convert("L") #converts to gray scale
          img_array = np.array(img,"uint8")

          faces = face.detectMultiScale(img_array,scaleFactor = 1.5, minNeighbors = 5)

          for (x,y,w,h) in faces:
            region = img_array[y:y+h, x:x+w]
            x_train.append(region)
            y_train.append(id_)

    with open('/Users/xenox/Documents/Coaaadinggg/site/mysite/mysite/main/Face recognition/label.pickle', 'wb') as f:
      pickle.dump(lab_ids, f)

    rec.train(x_train, np.array(y_train))
    rec.save("/Users/xenox/Documents/Coaaadinggg/site/mysite/mysite/main/Face recognition/trained.yml")
    messages.success(request, "Model successfully trained now you can use Face Recognition")
    return render(request = request,
                  template_name = 'main/face_recognition.html',
                  context = {'tutorials':Tutorial.objects.all})
  else:
        messages.error(request, "Please Login to use this feature")
        return redirect("main:homepage")
```

To train the model with new weights *updating_weights_for_face_recognition()* function takes the images and and detects faces using face.detectMultiScale() method. Now this face is stored in x_train and its corresponding label in y_train to later train the model. New names is saved in labels file to store the names of all the people in the current image dataset pickle.dump(). Model is trained and weights are saved in a yml file rec.save(). Page is refreshed and a message is shown to the user that the model has been trained.

Music Generation:

This technique allows users to generate music from scratch just by a click of a button. I have used LSTM (Long Short Term Memory) networks to generate music as they keep track of weights at each time step to generate sequence data.

```
def gen_m():
  ''' function to generate music'''
  #loading the notes files created while training
  with open('/Users/xenox/Documents/Coaaadinggg/site/mysite/mysite/main/music/data/notes', 'rb') as f: #opening the file as variable f
    notes = pickle.load(f) #Getting the notes in notes variable

  pitch_names = sorted(set(item for item in notes)) #sorting all the notes
  n = len(set(notes)) #getting the total number of individual notes

  inp, norm_inp = normalize(notes, pitch_names, n) #getting input in categorical format and also the normalized input

  model = make_model(norm_inp,n) #Getting the model from the funcion
  pred = gen_notes(model,inp,pitch_names,n)
  song = make_midi(pred)

def normalize(notes,pitch_names,n):
  #prepare the data for neural network
  #Also map the inputs to intigers to use in the network

  note_int = dict((note, number) for number, note in enumerate(pitch_names)) #Mapping numbers to notes and storing in the variabel

  l = 100 #min number of inuts req to make predictions
  inp = [] #to collect the input in integer form

  for i in range(0,len(notes) - l): #looping through all notes
    sequence_inp = notes[i:i+l] #taking the notes from i to i+l i.e 100 + i. to get the next notes
    seq_out = notes[i+l]
    inp.append([note_int[x] for x in sequence_inp]) #getting the categorical number of hte note from note_int

  n_patterns = len(inp) #number of patterns

  norm_inp = np.reshape(inp, (n_patterns, l, 1)) #Reshaping the array to a tensor so that we can feed it in our network
  norm_inp = norm_inp / float(n) #normalizing the input

  return (inp,norm_inp)

def make_model(inp,n):

  model = Sequential() #making object of sequential class
  model.add(LSTM(512, input_shape = (inp.shape[1], inp.shape[2]),return_sequences = True)) #adding a LSTM network and filling in the re
  model.add(Dropout(0.3)) #Addng dropout layer, regularization
  model.add(LSTM(512,return_sequences = True))
  model.add(Dropout(0.3))
  model.add(LSTM(512))
  model.add(Dense(256)) #basic deep neural network
  model.add(Dropout(0.3))
  model.add(Dense(n))
  model.add(Activation('softmax')) #to get the number of outputs same as number of individual notes
  model.compile(loss = 'categorical_crossentropy', optimizer = 'rmsprop') #loss function for back-propogation

  model.load_weights('/Users/xenox/Documents/Coaaadinggg/site/mysite/mysite/main/music/weights.hdf5') #loading the pre-trained weights

  return model
```

When user clicks generate button gen_m() function is called. It uses pre-trained weights of LSTM network to generate notes given 100 previous notes. It normalises the input i.e notes and sends it to make_model() function, in this function we create the model using *model=Sequential()* and add different layers such as LSTM, Dense, Activation etc, using *model.add().* Then pre-trained weights are loaded using *model.load_weights()* and the model is returned

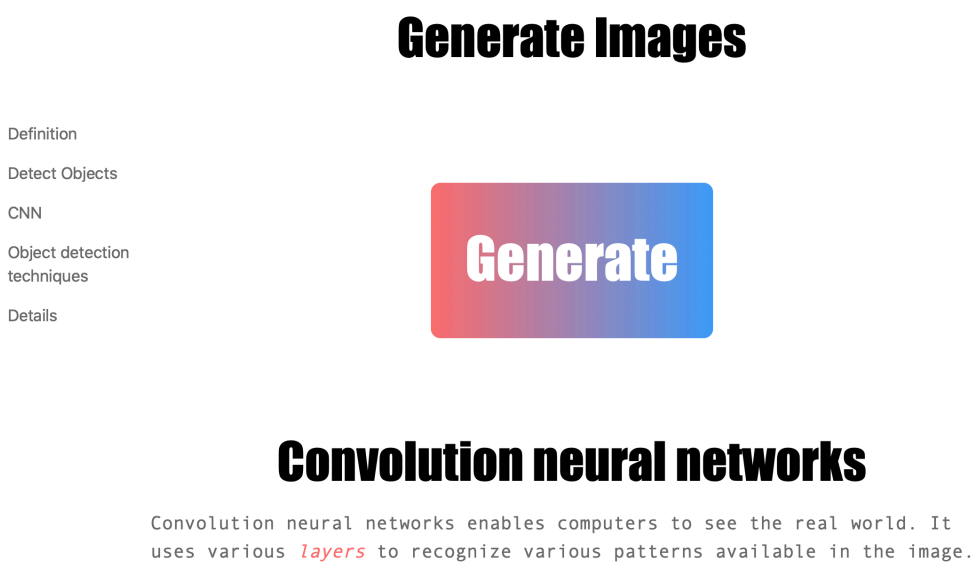
```
def gen_notes(model,inp,pitch_names,n):
  #pick a random seq from inp to start generation
  first = np.random.randint(0,len(inp) - 1)
  int_note = dict((number, note) for number, note in enumerate(pitch_names)) #getting the notes from number to co

  start = inp[first] #get the first categorical note
  pred = [] #to store all the predictions

  for index in range(100): #to generate next 500 notes
    pred_inp = np.reshape(start,(1,len(start),1)) #changing into tensor
    pred_inp = pred_inp / float(n) #normalizing

    prediction = model.predict(pred_inp,verbose = 0) #to not display anything in console while  predcting next no

    most_likely = np.argmax(prediction) #get the max of all outputs
    res = int_note[most_likely] #get the note for the categorical variable
    pred.append(res) #adding the reslt to the final array of predictions

    start.append(most_likely) #adding the pred to start to continue the loop
    start = start[1:len(start)]

  return pred

def make_midi(prediction_output):
    """ convert the output from the prediction to notes and create a midi file
        from the notes """
    offset = 0
    output_notes = []

    # create note and chord objects based on the values generated by the model
    for pattern in prediction_output:
        # pattern is a chord
        if ('.' in pattern) or pattern.isdigit():
            notes_in_chord = pattern.split('.')
            notes = []
            for current_note in notes_in_chord:
                new_note = note.Note(int(current_note))
                new_note.storedInstrument = instrument.Piano()
                notes.append(new_note)
            new_chord = chord.Chord(notes)
            new_chord.offset = offset
            output_notes.append(new_chord)
        # pattern is a note
        else:
            new_note = note.Note(pattern)
            new_note.offset = offset
            new_note.storedInstrument = instrument.Piano()
            output_notes.append(new_note)

        # increase offset each iteration so that notes do not stack
        offset += 0.5

    midi_stream = stream.Stream(output_notes)

    midi_stream.write('midi', fp='main/music/test_output.mid')
```

gen_notes() function takes the model and 100 input sample to generate new music *model.predict()* and save it in an array. This array is then passed to make_midi() function which arranges the notes in a sequence and creates a midi file which is then saved *midi_stream.write().*

```python
def playAudioFile(request):
    file="main/music/test_output.mid"
    f = open(file,"rb")
    response = HttpResponse()
    response.write(f.read())
    response['Content-Type'] ='audio/midi'
    response['Content-Length'] =os.path.getsize(file)
    return response
```

This function retrieves the music from the folder and loads is as *HttpResponse()* and returns it to be downloaded in the download folder of the user.

Generate Images:



Generate button in the page allows users to generate random images of 25 different things. This can be used by many artists to get inspiration and develop their own art.

```python
def gen_image():
  D = Discriminator()
  G = Generator()

# load weights
  D.load_state_dict(torch.load('/Users/xenox/Documents/Coaaadinggg/site/mysite/mysite/main/Gen_image/weights/weight_D.pth',map_location
  G.load_state_dict(torch.load('/Users/xenox/Documents/Coaaadinggg/site/mysite/mysite/main/Gen_image/weights/weight_G.pth',map_location

  batch_size = 25 #number of images generate in a batch
  latent_size = 100

  fixed_noise = torch.randn(batch_size, latent_size, 1, 1)
  fake_images = G(fixed_noise)

  fake_images_np = fake_images.cpu().detach().numpy()
  fake_images_np = fake_images_np.reshape(fake_images_np.shape[0], 3, 32, 32)
  fake_images_np = fake_images_np.transpose((0, 2, 3, 1))
  images = []
  for i in range(batch_size):
    images.append(fake_images_np[i])

  vertical = np.vstack(tuple(images))
  vertical *= 255.0/vertical.max()
  cv2.imwrite("generated.png", vertical)
  download_img()
```

The code uses Generator and Discriminator classes to create GAN's architecture. It then uses pre-trained weights to generate fake images. As the images are generated in range -1 to 1, they need to be converted to 0 to 255 which is done using *vertical *= 255.0/ vertical.max()*

View Images:

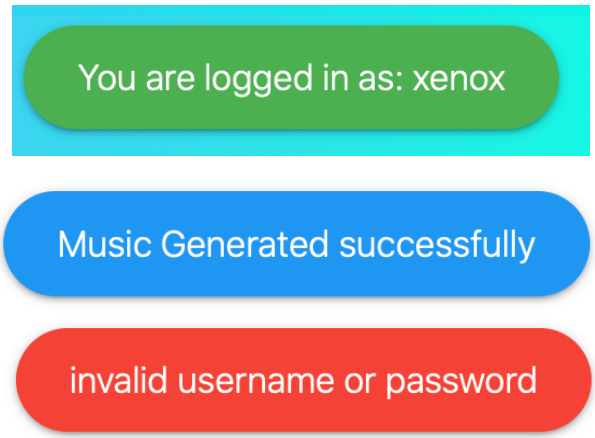
```python
def download_img(request):
  file="/Users/xenox/Documents/Coaaadinggg/site/mysite/mysite/generated.jpeg"
  f = open(file,"rb")
  response = HttpResponse()
  response.write(f.read())
  response['Content-Type'] ='image/jpeg'
  response['Content-Length'] =os.path.getsize(file)
  return response
```

This code allows users to view images generated using the generator in a new window

Code Snippet:

Code panel is available for all the techniques to let the user know what code is running behind the scenes. Code for all the techniques are stored in database and is received whenever user visits the page

Messages:



There are 3 types of alter messages, Red for any error, Green for success and Blue for info.

```
{% if messages %}
    {% for m in messages%}
        {% if m.tags == "success" %}
    <script> M.toast({html: "{{m}}", classes:"blue rounded", displayLength:3000}); </script>
        {% elif m.tags == "info" %}
                <script> M.toast({html: "{{m}}", classes:"green rounded", displayLength:3000}); </script>
        {% elif m.tags == "warning" %}
            <script> M.toast({html: "{{m}}", classes:"orange rounded", displayLength:3000}); </script>
         {% elif m.tags == "error" %}
            <script> M.toast({html: "{{m}}", classes:"red rounded", displayLength:3000}); </script>
         {% endif %}
    {% endfor %}
{% endif %}
```

Above code is to create template for messages and shows different messages depending on

The type of message to be displayed.