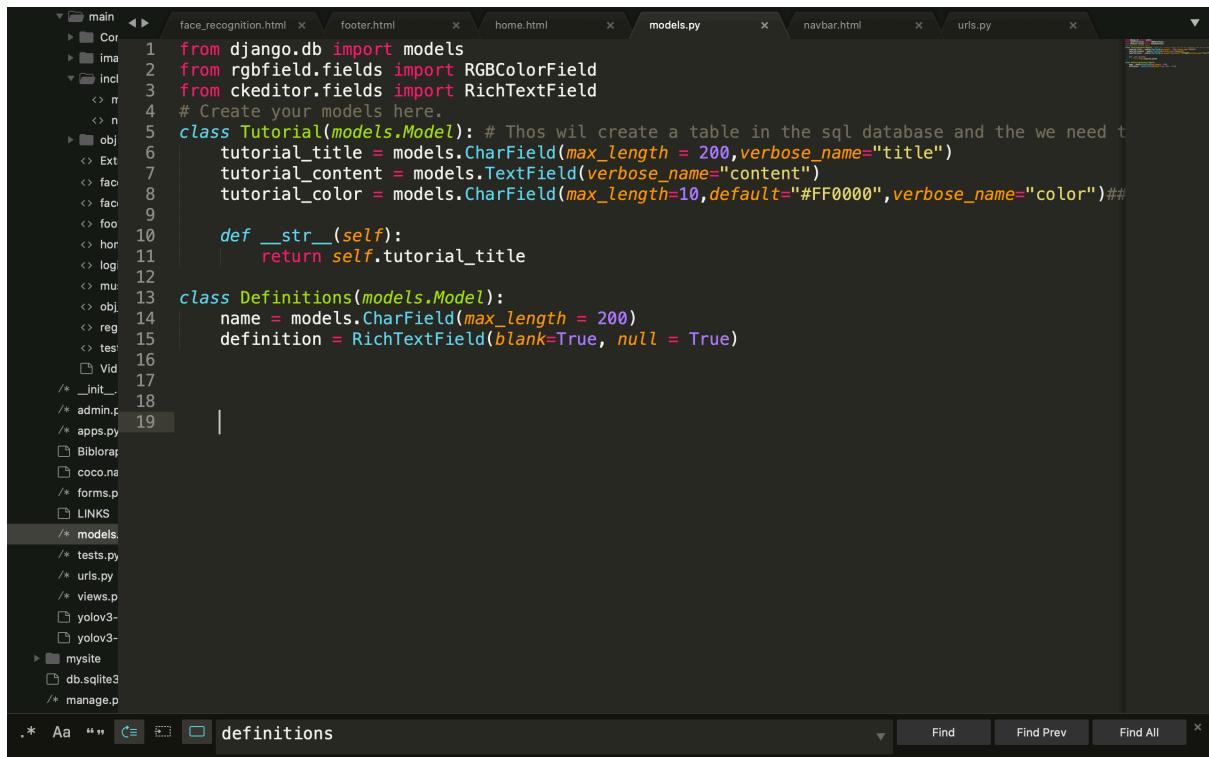


Criterion C: Development

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:



The screenshot shows a code editor with multiple tabs open at the top: face_recognition.html, footer.html, home.html, models.py (which is the active tab), navbar.html, and urls.py. The sidebar on the left shows a project structure with folders like main, Cor, incl, obj, Ext, fac, foo, hor, logi, mu, obj, reg, tes, Vid, __init__.py, admin.py, apps.py, Bibliotek, coco.na, forms.py, LINKS, models.py, tests.py, urls.py, views.py, yolov3-, and mysite. The db.sqlite3 file is also listed under mysite. The models.py file contains the following Python code:

```
1  from django.db import models
2  from rgbfield.fields import RGBColorField
3  from ckeditor.fields import RichTextField
4  # Create your models here.
5  class Tutorial(models.Model): # This will create a table in the sql database and the we need to
6      tutorial_title = models.CharField(max_length = 200, verbose_name="title")
7      tutorial_content = models.TextField(verbose_name="content")
8      tutorial_color = models.CharField(max_length=10, default="#FF0000", verbose_name="color")##
9
10     def __str__(self):
11         return self.tutorial_title
12
13 class Definitions(models.Model):
14     name = models.CharField(max_length = 200)
15     definition = RichTextField(blank=True, null = True)
```

models.py file contains all the classes for the database, class Tutorial(models.Model) and class Definitions(models.Model) both inherit from models.Model to use its 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.

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

app_name = 'main'

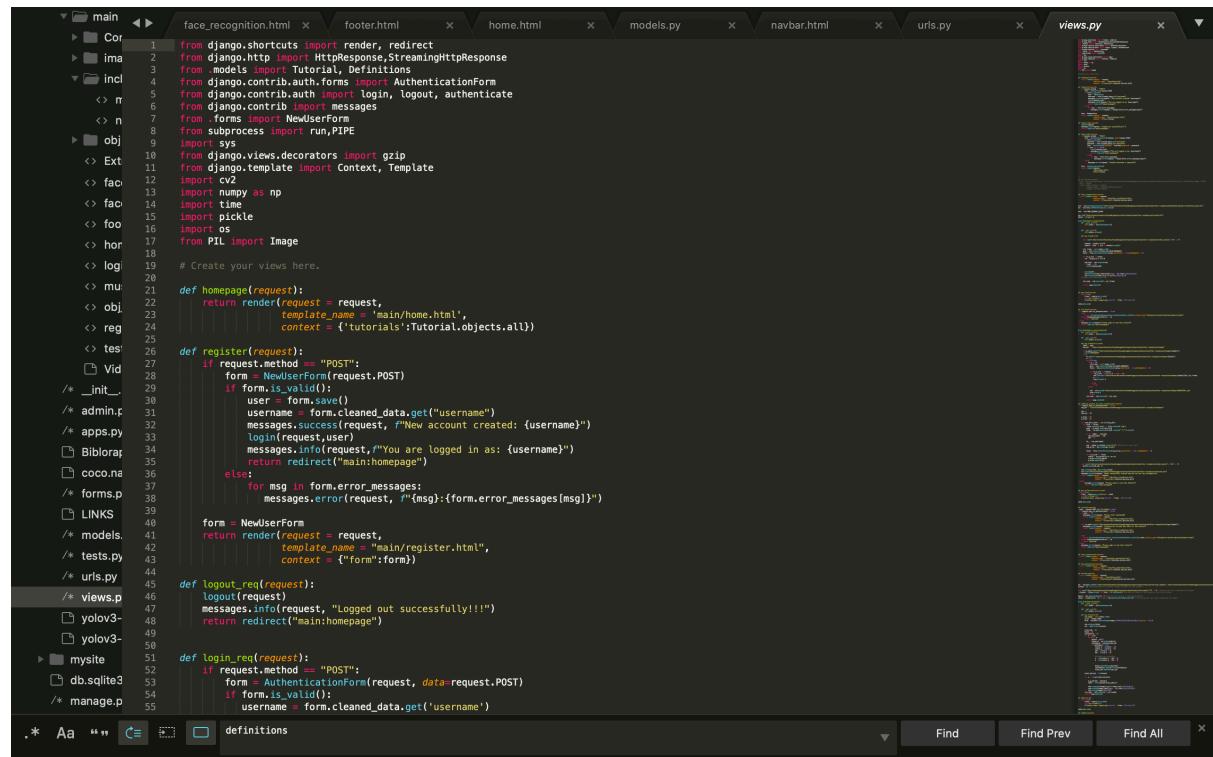
urlpatterns = [

    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:



```
from django.shortcuts import render, redirect
from django.http import HttpResponseRedirect
from models import Tutorial, Definitions
from django.contrib import messages
from forms import NewUserForm
from subprocess import run,PIPE
import sys
from django.views.decorators import gzip
from django.template import Context, Template
import cv2
import numpy as np
import time
import pickle
import os
from PIL import Image
# Create your views here.

def homepage(request):
    return render(request = request,
                  template_name = "main/home.html",
                  context = {'tutorials':Tutorial.objects.all})

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,"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]}')

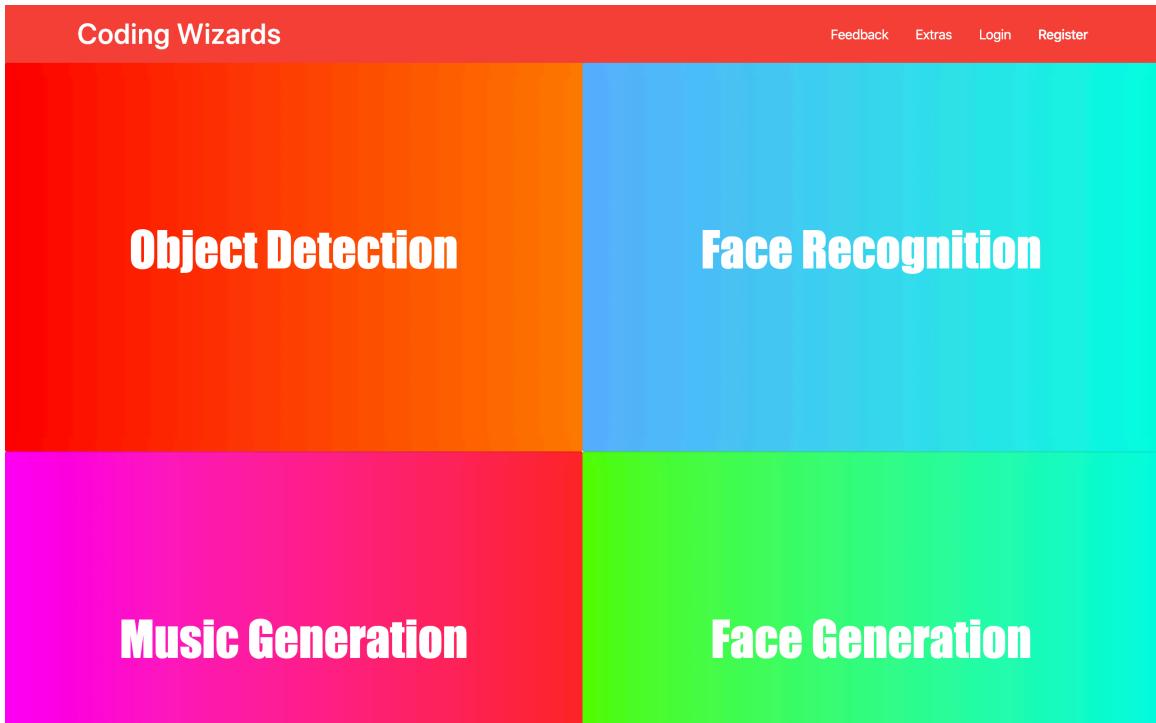
    form = NewUserForm
    return render(request = request,
                  template_name = "main/register.html",
                  context = {"form":form})

def logout_req(request):
    logout(request)
    messages.info(request, "Logged out successfully!!!")
    return redirect("main:homepage")

def login_req(request):
    if request.method == "POST":
        form = AuthenticationForm(request, data=request.POST)
        if form.is_valid():
            username = form.cleaned_data.get('username')
```

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.

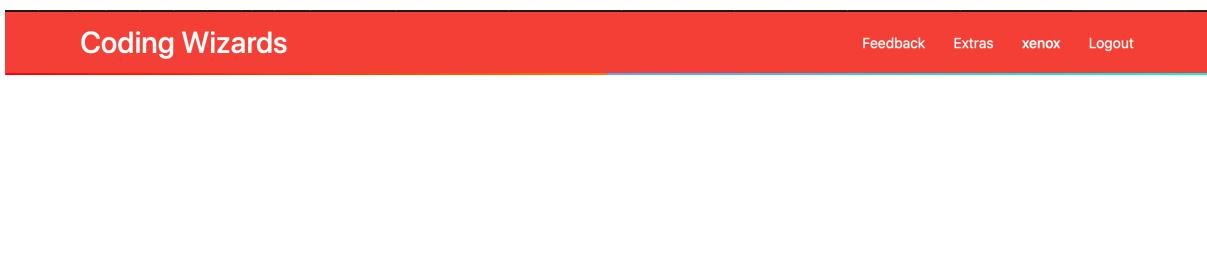
Home Page:



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 navbar.html and footer.html files.



```

1 <head>
2   <link href = ".../home.css" rel="stylesheet">
3
4 </head>
5
6 <div>
7   <nav style="background-color: #f44336;">
8     <div class="nav-wrapper" style="box-shadow: 0px 3px #9E9E9E;">
9       <a href="#" class="brand-logo" style = "padding-left:80px"><strong>Coding Wizards</strong></a>
10      <ul id="nav-mobile" class="right hide-on-med-and-down">
11        <li><a href="https://docs.google.com/forms/d/e/1FAIpQLSeWhAG7ZTm52vrl18XwQ1kUNzpNOYZryMal9JG6yplsxoDCdg/viewform?usp=sf_link">Feedback</a></li>
12        <li><a href="Extras">Extras</a></li>
13
14       {% if user.is_authenticated %}
15         <li><strong><a href="#">{{user.username}}</a></strong></li>
16         <li><a href="#" style = "padding-right:80px">Logout</a></li>
17
18       {% else %}
19         <li><a href="login">Login</a></li>
20         <li><strong><a href="register" style = "padding-right:80px">Register</a></strong></li>
21
22       {% endif %}
23     </ul>
24   </div>
25 </nav>
26 </div>
27
28
29
30

```

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 %})



```

1  <head>
2    % load static
3    <link href = "% static 'main/css/prism.css' %" rel="stylesheet">
4    <link href = "% static 'main/css/obj_detection.css' %" rel="stylesheet">
5    <link href = "% static 'main/css/materialize-loading-bar.css' %" rel="stylesheet" type="text/css" />
6    <script src="% static 'tinymce.js' %" type="text/javascript" />
7
8    
9    <link href = "% static 'main/css/materialize.css' %" rel="stylesheet">
10   <!-- Compiled and minified JavaScript -->
11   <script src="https://unpkg.com/materialize@1.0.0/dist/js/materialize.min.js"></script>
12
13   <script src="https://fonts.googleapis.com/icon?family=Material+Icons"></script>
14   <link href="https://fonts.googleapis.com/icon?family=Material+Icons"> rel="stylesheet"
15
16  </head>
17
18  <body>
19
20
21  (% include "main/includes/navbar.html" %)
22
23
24
25  (% include "main/includes/messages.html" %)
26  (% block content %)
27  (% endblock %)
28
29
30
31  <footer class="page-footer" style="background-color: #F0E6EE;">
32    <div class="container">
33      <div class="row l6 s12">
34        <div><p>This is the End!!</p></div>
35        <div><a href="#">Contact: shubham1200@gmail.com</a></div>
36        <div><a href="https://docs.google.com/forms/d/e/1FAIpQLSehAG7Tm52vlr18Xw01kUNzpN0YzryMaL9JG6ypslxoDCdg/viewform?usp=sf_link" target="blank">Feedback</a></div>
37
38        <div><a href="#">Instagram</a></div>
39        <div><a href="#">Twitter</a></div>
40        <div><a href="#">Facebook</a></div>
41
42      </div>
43
44      <div class="col l4 offset-l2 s12">
45        <div><a href="#">Links</a></div>
46        <div><a href="#">Object Detection</a></div>
47        <div><a href="#">Face Recognition</a></div>
48        <div><a href="#">GMS for faces</a></div>
49
50      </div>
51
52    </div>
53
54  </div>
55
56  <div class="footer-copyright">
57    <div><a href="#">Copyright Coding Wizz</a></div>
58    <div>© 2019 Copyright Coding Wizz</div>
59
60  </div>
61
62  </div>
63
64  <script> src = "% static 'tinymce/js/prism.js' %" </script>
65

```

Line 35, Column 54

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:

Coding Wizards

* Required

Email address *

Your email

Did you like our website *

yes

no

was it easy to navigate *

Yes

No

Figure(4): Google Forms

```

<a href="#">Homepage</a> <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/1FAIpQLSehAG7Tm52vlr18Xw01kUNzpN0YzryMaL9JG6ypslxoDCdg/viewform?usp=sf_link">Feedback</a></li>
  <li><a href="#">Extras</a></li>

```

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:

The screenshot shows a web application interface with a red header bar containing the text "Coding Wizards". Below the header, the main content area has a title "Definitions of all the terms" in large bold black font. The content is organized into sections with pink-colored titles: "AI (Artificial Intelligence) :" and "Layers :". Each section contains a brief description. Below these, there is another section titled "Computer Vision :" with a link to "You can learn more about computer vision from here:-" followed by links to "Ted" and "Udemy".

The screenshot shows a code editor with an open file named "Extras.html". The code is a template for a web page, specifically for displaying definitions. It includes imports for "main/footer.html" and "main/navbar.html". The main content starts with a large heading "Definitions of all the terms". Below this, a loop iterates over "definitions" and generates a table structure. The table has two columns: one for the term name and one for its definition. The code uses CSS inline styles for the table and its rows. The code editor interface shows line numbers from 1 to 28, and various navigation and search tools at the bottom.

```

<% extends "main/footer.html" %>
<head>
</head>
<body>
<% block content %>
<div class="container">
<h2 style="color: black; text-align: center; font-family: Impact;" name="Definitions"> Definitions of all the terms </h2>
<% for def in definitions %>
<table style="border: 0; padding: 20px; border-spacing: 0px; position: relative;">
<tr>
<td style = "height: 100px; width: 25%; "><i><p style="margin: auto; color: #F0E6EE; font-family: Andale Mono;">{{ def.name | safe }} </p></i></td>
<td style = "height: 100px; width: 80%; "><p style="margin: auto; color: #696969; font-family: Andale Mono;">{{ def.definition | safe }} </p></td>
</tr>
</table>
<% endfor %>
</div>
<% endblock %>
</body>

```

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:

The screenshot shows the login page of a website named "Coding Wizards". The header is red with the site name "Coding Wizards" on the left and links for "Feedback", "Extras", "Login", and "Register" on the right. Below the header is a form with two input fields: "Username:" and "Password:", each with a horizontal line below it for text entry. To the right of the "Username:" field is a small placeholder text "Username:". To the right of the "Password:" field is a small placeholder text "Password:". Below the password field is a green "LOGIN" button with white text. At the bottom of the form area, there is a link "Don't have an account yettt??????" followed by a green "SIGN UP" button. The main content area below the form has a red background. On the left side of this area, the text "This is the End!!!" is displayed in bold black font. Below it are several links: "Contact: codingwizards@gmail.com", "Feedback", "Instagram", "Twitter", and "Facebook". On the right side, under the heading "Links", are links to "Object Detection", "Face Recognition", "GANS for music", and "GANS for faces". At the very bottom of the page, a thin red bar contains the copyright notice "© 2019 Copyright Coding Wizz".

```

main
└── Content
    ├── face_recognition.html
    └── footer.html
    └── login.html
        1  {%extends "main/footer.html" %} 
        2
        3  {% block content %}
        4      <br/>
        5      <br/>
        6  <div class="container">
        7      <form method="POST">
        8          {% csrf_token %}<!-- protects from cross site request forgery attacks-->
        9          {{form.as_p}}
       10         <button class="btn waves-effect waves-light" type="submit" name="action"> Login </button>
       11     </form>
       12     <br/>
       13     <br/>
       14     <br/>
       15     Don't have an account yettt????? <a href="/register" style="color:white"> <button class="btn
       16         waves-effect waves-light"> Sign up </a>
       17     </div>
       18     <br/>
       19     <br/>
       20     <br/>
       21     <br/>
       22     <br/>
       23  {% endblock %}
       24
       25

```

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:

Coding Wizards

Feedback Extras Login Register

Username:

Required. 150 characters or fewer. Letters, digits and @./+/-/_ only.

Email:

|

Password:

Your password can't be too similar to your other personal information.

Your password must contain at least 8 characters.

Your password can't be a commonly used password.

Your password can't be entirely numeric.

Password confirmation:

Enter the same password as before, for verification.

SUBMIT

Already have an account??? NO problem hed over to our [LOGIN PAGE](#)

```
1  {%extends "main/footer.html" %}          register.html
2
3  {% block content %}                      views.py
4      <br/>
5      <br/>
6      <div class="container">
7          <form method="POST">
8              {% csrf_token %} <!-- protects from cross site request forgery attacks-->
9              {{form.as_p}}
10
11             <button class="btn waves-effect waves-light" type="submit" name="action"> Submit</button>
12
13         </form>
14
15         <br/>
16         <br/>
17         <br/>
18         Already have an account??? NO problem hed over to our <a href="login"> <button class="btn waves-effect
19             waves-light" type="submit" name="action"> Login page </button></a>
20
21     </div>
22     <br/>
23     <br/>
24     {% 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.

```

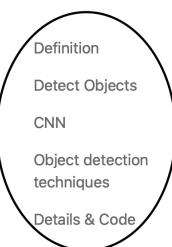
def register(request):
    if request.method == "POST":
        form = NewUserForm(request.POST)
        if form.is_valid():
            user = form.save()
            username = form.cleaned_data.get("username")
            messages.success(request, f"New account created: {username}")
            login(request, 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:



Object Detection

→ **Anchor**

Definition

Ever wondered!!! how does your car drives on its own? or how does your google lens knows what object you are showing to it? Well Its all Object Detection. It comes under a field of AI (Artificial Intelligence) called **Computer Vision**. Object Detection can detect any object (on which it is trained on) in an image. Well-researched domains of object detection include face detection and pedestrian detection. It can even be used to help blind people know about nearby objects, It can also be used for surveillance and much much more...

```

<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. { ``

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)

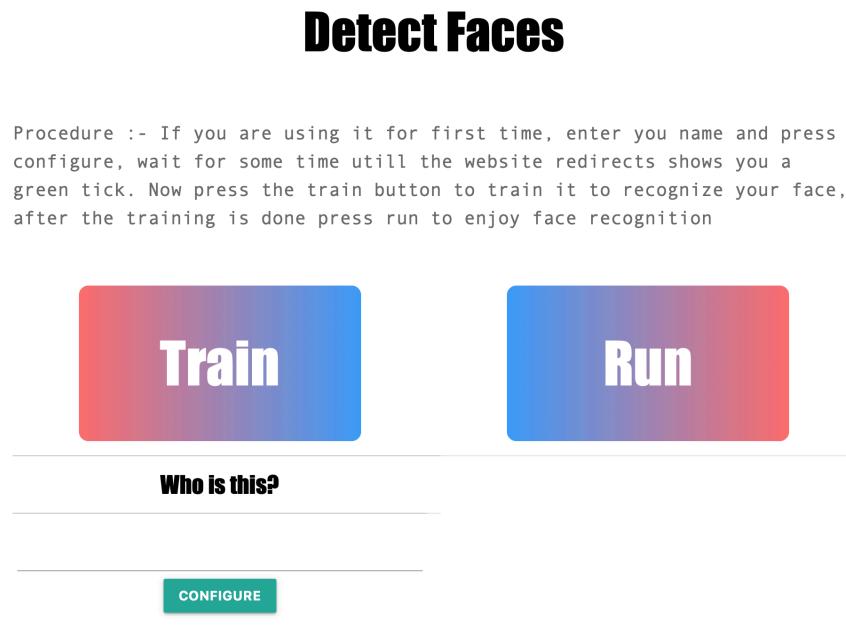
    total_objects = len(boxex)

    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/main/Face recognition/Images"

        if os.path.isdir(f'/Users/xenox/Documents/Coaaadinggg/site/mysite/main/Face recognition/Images/{label}'):
            print("WORKINNN")
        else:
            os.mkdir(f'/Users/xenox/Documents/Coaaadinggg/site/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/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/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(path).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:

Generate Music

Please give the generator 30 seconds.

GENERATE

DOWNLOAD

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.

```
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
```

```
def gen_m():
    ''' function to generate music'''
    #loading the notes files created while training
    with open('/Users/xenox/Documents/Cooaddinggg/site/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 integers 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, 1, 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)) #Adding 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/Cooaddinggg/site/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 predicting 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()*.

```
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 it as *HttpResponse()* and returns it to be downloaded in the download folder of the user.

Generate Images:

Generate Images

Definition

Detect Objects

CNN

Object detection
techniques

Details



Generate

Convolution neural networks

Convolution neural networks enables computers to see the real world. It uses various *layers* to recognize various patterns available in the image.

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.

```

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

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

    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:

```

def download_img(request):
    file="/Users/xenox/Documents/Coaaadinggg/site/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

[Github Link](#)

Definition

Detect Objects

CNN

Object detection
techniques

Details & Code

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

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

    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)
```

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:

You are logged in as: xenox

Music Generated successfully

invalid username or password

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.