SOURCE CODE

User Side views.py

```
from django.shortcuts import render, HttpResponse
from django.contrib import messages
from .forms import UserRegistrationForm
from .models import UserRegistrationModel, UserSearchUrlModel, CSRFResponse
import json
import subprocess
import pandas as pd
from .UserMachineLearningAlgorithms import MLConcepts


# 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, 'UsersRegister.html', {'form': form})
        else:
            messages.success(request, 'Email or Mobile Already Existed')
            print("Invalid form")
    else:
        form = UserRegistrationForm()
    return render(request, 'UsersRegister.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')
            # return render(request, 'user/userpage.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 UserPreProcessForm(request):
    return render(request, "users/UserPreProcessForm.html", {})


def UserCSRFProcessByBolt(request):
    if request.method == "POST":
        urlname = request.POST.get("urlname")
        depth = request.POST.get("depth")
        UserSearchUrlModel.objects.create(urlname=urlname, depthfecth=depth)
        command = "python Bolt-master/bolt.py" + " -u " + urlname + " -l" + " " +
depth
        print("path " + command)
        subprocess.call(command)
        f = open('./db/hashes.json', )
        data = json.load(f)
        # print("Data is ",len(data))
        mydict = {}
        for i in data:
            keys = i.keys();
            # print("fu=",keys['regex'])
            # print("fus",keys['matches'])
            for x in keys:
                regex = i['regex']
                matches = i['matches']
                CSRFResponse.objects.create(regex=regex, matches=matches,
urlname=urlname)

    data = CSRFResponse.objects.filter(urlname=urlname)
    return render(request, "users/CSRFProcess.html", {"data": data})


def UserMitchProcess(request):
    f = open('./media/dataset/dataset.json', )
    data = json.load(f)
    mydict = {}
    for i in data:
        keys = i.keys()
        data = i['data']
        website = i['website']
        i = 0
        for x in data:
            #print("X value = ",x)
            i = i+1
            mydict.update({i:x})
        #mydict.update({data: website})
        #for x,y in keys.items():
            #data = keys.get('data')
```

```
            #website = i['website']
            #print(y)

            #print(data,"<==>",website)

    return render(request, "users/MitchProcessone.html", {"data": mydict})

def UserMachineLearning(request):
    df = pd.read_csv('./media/dataset/features_matrix.csv', sep=',', delimiter=None,
header='infer', names=None, index_col=None, usecols=None, squeeze=False, prefix=None,
mangle_dupe_cols=True, dtype=None, engine=None, converters=None, true_values=None,
false_values=None, skipinitialspace=False, skiprows=None, skipfooter=0, nrows=None,
na_values=None, keep_default_na=True, na_filter=True, verbose=False,
skip_blank_lines=True, parse_dates=False, infer_datetime_format=False,
keep_date_col=False, date_parser=None, dayfirst=False, cache_dates=True,
iterator=False, chunksize=None, compression='infer', thousands=None, decimal='.',
lineterminator=None, quotechar='"', quoting=0, doublequote=True, escapechar=None,
comment=None, encoding=None, dialect=None, error_bad_lines=True, warn_bad_lines=True,
delim_whitespace=False, low_memory=True, memory_map=False, float_precision=None)
    obj = MLConcepts()
    post_dict = obj.startPOSTProcess(df)
    get_dict = obj.startGETProcess(df)
    option_dict = obj.startOPTIONProcess(df)
    return
render(request,"users/UserMachineLearning.html",{'post_dict':post_dict,'get_dict':get
_dict,"option_dict":option_dict})
```

userMachineLearningAlgorithm.py

```
from sklearn.model_selection import train_test_split
class MLConcepts:
    def startPOSTProcess(self,df):
        print(df.head())
        df = df[['numOfParams', 'numOfBools',
'numOfIds','numOfBlobs','reqLen','isPOST']]
        #df_get = df[['numOfParams', 'numOfBools', 'numOfIds', 'numOfBlobs',
'reqLen', 'isGET']]
        X = df[['numOfParams', 'numOfBools', 'numOfIds','numOfBlobs','reqLen']]
        y = df[['isPOST']]
        X_train,X_test, y_train, y_test = train_test_split(X, y,
test_size=1/3,random_state=0)
        from sklearn.ensemble import RandomForestClassifier
        regressor = RandomForestClassifier()
        regressor.fit(X_train, y_train)
        # Predecting The test Result
        y_pred = regressor.predict(X_test)
        # Need to implement Accuracy, Precession and recall
        from sklearn.metrics import accuracy_score
        accuracy = accuracy_score(y_pred.round(), y_test)
        print('POST Accuracy=', accuracy)
        from sklearn.metrics import precision_score
        precision = precision_score(y_pred.round(), y_test)
        print("POST Precession=", precision)
        from sklearn.metrics import recall_score
```

```
        recall = recall_score(y_pred.round(), y_test)
        print("POST Recall=", recall)
        post_dict =
{"post_accuracy":accuracy,"post_precision":precision,"post_recall":recall}
        return post_dict

    def startGETProcess(self,df):
        print(df.head())
        df = df[['numOfParams', 'numOfBools',
'numOfIds','numOfBlobs','reqLen','isGET']]
        X = df[['numOfParams', 'numOfBools', 'numOfIds','numOfBlobs','reqLen']]
        y = df[['isGET']]
        X_train,X_test, y_train, y_test = train_test_split(X, y,
test_size=1/3,random_state=0)
        from sklearn.ensemble import RandomForestClassifier
        regressor = RandomForestClassifier()
        regressor.fit(X_train, y_train)
        # Predecting The test Result
        y_pred = regressor.predict(X_test)
        # Need to implement Accuracy, Precession and recall
        from sklearn.metrics import accuracy_score
        accuracy = accuracy_score(y_pred.round(), y_test)
        print('GET Accuracy=', accuracy)
        from sklearn.metrics import precision_score
        precision = precision_score(y_pred.round(), y_test)
        print("GET Precession=", precision)
        from sklearn.metrics import recall_score
        recall = recall_score(y_pred.round(), y_test)
        print("GET Recall=", recall)
        get_dict = {"get_accuracy": accuracy, "get_precision":precision,
"get_recall": recall}
        return get_dict

    def startOPTIONProcess(self,df):
        print(df.head())
        df = df[['numOfParams', 'numOfBools',
'numOfIds','numOfBlobs','reqLen','isOPTIONS']]
        X = df[['numOfParams', 'numOfBools', 'numOfIds','numOfBlobs','reqLen']]
        y = df[['isOPTIONS']]
        X_train,X_test, y_train, y_test = train_test_split(X, y,
test_size=1/3,random_state=0)
        from sklearn.ensemble import RandomForestClassifier
        regressor = RandomForestClassifier()
        regressor.fit(X_train, y_train)
        # Predecting The test Result
        y_pred = regressor.predict(X_test)
        # Need to implement Accuracy, Precession and recall
        from sklearn.metrics import accuracy_score
        accuracy = accuracy_score(y_pred.round(), y_test)
        print('OPTION Accuracy=', accuracy)
        from sklearn.metrics import precision_score
        precision = precision_score(y_pred.round(), y_test)
        print("OPTION Precession=", precision)
        from sklearn.metrics import recall_score
        recall = recall_score(y_pred.round(), y_test)
```

```
        print("OPTION Recall=", recall)
        ooption_dict = {"option_accuracy": accuracy, "option_precision":precision,
"option_recall": recall}
        return ooption_dict
```

models.py

```
from django.db import models

# Create your models here.

# 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 = 'Registrations'

class UserSearchUrlModel(models.Model):
    id = models.AutoField(primary_key=True)
    urlname = models.CharField(max_length=250)
    depthfecth = models.IntegerField()
    c_date = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.urlname
    class Meta:
        db_table = "UserSearchUrls"

class CSRFResponse(models.Model):
    id = models.AutoField(primary_key=True)
    regex = models.CharField(max_length=10000)
    matches = models.CharField(max_length=100000)
    urlname = models.CharField(max_length=1000)
    c_date = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.urlname
    class Meta:
        db_table = "csrftables"
```

getting all csrfs:

```
from core.colors import green, yellow, end, run, good, info, bad, white, red

lightning = '\033[93;5m⚡\033[0m'


def banner():
    print ('''
     %s⚡ %sBOLT%s  ⚡%s
    ''' % (yellow, white, yellow, end))


banner()

try:
    import concurrent.futures
    try:
        from fuzzywuzzy import fuzz, process
    except:
        import os
        print ('%s fuzzywuzzy library is not installed, installing now.' % info)
        os.system('pip3 install fuzzywuzzy')
        print ('%s fuzzywuzzy has been installed, please restart Bolt.' % info)
        quit()
except:
    print ('%s Bolt is not compatible with python 2. Please run it with python 3.' %
bad)

import argparse
import json
import random
import re
import statistics

from core.entropy import isRandom
from core.datanize import datanize
from core.prompt import prompt
from core.photon import photon
from core.tweaker import tweaker
from core.evaluate import evaluate
from core.ranger import ranger
from core.zetanize import zetanize
from core.requester import requester
from core.utils import extractHeaders, strength, isProtected, stringToBinary,
longestCommonSubstring

parser = argparse.ArgumentParser()
parser.add_argument('-u', help='target url', dest='target')
parser.add_argument('-t', help='number of threads', dest='threads', type=int)
parser.add_argument('-l', help='levels to crawl', dest='level', type=int)
parser.add_argument('--delay', help='delay between requests',
                    dest='delay', type=int)
parser.add_argument('--timeout', help='http request timeout',
                    dest='timeout', type=int)
```

```
parser.add_argument('--headers', help='http headers',
                    dest='add_headers', nargs='?', const=True)
args = parser.parse_args()

if not args.target:
    print('\n' + parser.format_help().lower())
    quit()

if type(args.add_headers) == bool:
    headers = extractHeaders(prompt())
elif type(args.add_headers) == str:
    headers = extractHeaders(args.add_headers)
else:
    from core.config import headers

target = args.target
delay = args.delay or 0
level = args.level or 2
timeout = args.timeout or 20
threadCount = args.threads or 2

allTokens = []
weakTokens = []
tokenDatabase = []
insecureForms = []

print (' %s Phase: Crawling %s[%s1/6%s]%s' %
       (lightning, green, end, green, end))
dataset = photon(target, headers, level, threadCount)
allForms = dataset[0]
print ('\r%s Crawled %i URL(s) and found %i form(s).%-10s' %
       (info, dataset[1], len(allForms), ' '))
print (' %s Phase: Evaluating %s[%s2/6%s]%s' %
       (lightning, green, end, green, end))

evaluate(allForms, weakTokens, tokenDatabase, allTokens, insecureForms)

if weakTokens:
    print ('%s Weak token(s) found' % good)
    for weakToken in weakTokens:
        url = list(weakToken.keys())[0]
        token = list(weakToken.values())[0]
        print ('%s %s %s' % (info, url, token))

if insecureForms:
    print ('%s Insecure form(s) found' % good)
    for insecureForm in insecureForms:
        url = list(insecureForm.keys())[0]
        action = list(insecureForm.values())[0]['action']
        form = action.replace(target, '')
        if form:
            print ('%s %s %s[%s%s%s]%s' %
                   (bad, url, green, end, form, green, end))

print (' %s Phase: Comparing %s[%s3/6%s]%s' %
```

```
       (lightning, green, end, green, end))
uniqueTokens = set(allTokens)
if len(uniqueTokens) < len(allTokens):
    print ('%s Potential Replay Attack condition found' % good)
    print ('%s Verifying and looking for the cause' % run)
    replay = False
    for url, token in tokenDatabase:
        for url2, token2 in tokenDatabase:
            if token == token2 and url != url2:
                print ('%s The same token was used on %s%s%s and %s%s%s' %
                       (good, green, url, end, green, url2, end))
                replay = True
    if not replay:
        print ('%s Further investigation shows that it was a false positive.')

with open('./db/hashes.json') as f:
    hashPatterns = json.load(f)

if not allTokens:
    print ('%s No CSRF protection to test' % bad)
    quit()

aToken = allTokens[0]
matches = []
for element in hashPatterns:
    pattern = element['regex']
    if re.match(pattern, aToken):
        for name in element['matches']:
            matches.append(name)
if matches:
    print ('%s Token matches the pattern of following hash type(s):' % info)
    for name in matches:
        print ('    %s>%s %s' % (yellow, end, name))


def fuzzy(tokens):
    averages = []
    for token in tokens:
        sameTokenRemoved = False
        result = process.extract(token, tokens, scorer=fuzz.partial_ratio)
        scores = []
        for each in result:
            score = each[1]
            if score == 100 and not sameTokenRemoved:
                sameTokenRemoved = True
                continue
            scores.append(score)
        average = statistics.mean(scores)
        averages.append(average)
    return statistics.mean(averages)


try:
    similarity = fuzzy(allTokens)
    print ('%s Tokens are %s%i%%%s similar to each other on an average' %
```

```
            (info, green, similarity, end))
except statistics.StatisticsError:
    print ('%s No CSRF protection to test' % bad)
    quit()


def staticParts(allTokens):
    strings = list(set(allTokens.copy()))
    commonSubstrings = {}
    for theString in strings:
        strings.remove(theString)
        for string in strings:
            commonSubstring = longestCommonSubstring(theString, string)
            if commonSubstring not in commonSubstrings:
                commonSubstrings[commonSubstring] = []
            if len(commonSubstring) > 2:
                if theString not in commonSubstrings[commonSubstring]:
                    commonSubstrings[commonSubstring].append(theString)
                if string not in commonSubstrings[commonSubstring]:
                    commonSubstrings[commonSubstring].append(string)
    return commonSubstrings


result = {k: v for k, v in staticParts(allTokens).items() if v}

if result:
    print ('%s Common substring found' % info)
    print (json.dumps(result, indent=4))

simTokens = []

print (' %s Phase: Observing %s[%s4/6%s]%s' %
       (lightning, green, end, green, end))
print ('%s 100 simultaneous requests are being made, please wait.' % info)


def extractForms(url):
    response = requester(url, {}, headers, True, 0).text
    forms = zetanize(url, response)
    for each in forms.values():
        localTokens = set()
        inputs = each['inputs']
        for inp in inputs:
            value = inp['value']
            if value and match(r'^[\w\-_]+$', value):
                if strength(value) > 10:
                    simTokens.append(value)


while True:
    sample = random.choice(tokenDatabase)
    goodToken = list(sample.values())[0]
    if len(goodToken) > 0:
        goodCandidate = list(sample.keys())[0]
        break
```

```
threadpool = concurrent.futures.ThreadPoolExecutor(max_workers=30)
futures = (threadpool.submit(extractForms, goodCandidate)
           for goodCandidate in [goodCandidate] * 30)
for i in concurrent.futures.as_completed(futures):
    pass

if simTokens:
    if len(set(simTokens)) < len(simTokens):
        print ('%s Same tokens were issued for simultaneous requests.' % good)
    else:
        print (simTokens)
else:
    print ('%s Different tokens were issued for simultaneous requests.' % info)

print (' %s Phase: Testing %s[%s5/6%s]%s' %
       (lightning, green, end, green, end))

parsed = ''
print ('%s Finding a suitable form for further testing. It may take a while.' % run)
for url, forms in allForms[0].items():
    found = False
    parsed = datanize(forms, tolerate=True)
    if parsed:
        found = True
        break
    if found:
        break

if not parsed:
    candidate = list(random.choice(tokenDatabase).keys())[0]
    parsed = datanize(candidate, headers, tolerate=True)
    print (parsed)

origGET = parsed[0]
origUrl = parsed[1]
origData = parsed[2]

print ('%s Making a request with CSRF token for comparison.' % run)
response = requester(origUrl, origData, headers, origGET, 0)
originalCode = response.status_code
originalLength = len(response.text)
print ('%s Status Code: %s' % (info, originalCode))
print ('%s Content Length: %i' % (info, originalLength))
print ('%s Checking if the resonse is dynamic.' % run)
response = requester(origUrl, origData, headers, origGET, 0)
secondLength = len(response.text)
if originalLength != secondLength:
    print ('%s Response is dynamic.' % info)
    tolerableDifference = abs(originalLength - secondLength)
else:
    print ('%s Response isn\'t dynamic.' % info)
    tolerableDifference = 0

print ('%s Emulating a mobile browser' % run)
```

```
print ('%s Making a request with mobile browser' % run)
headers['User-Agent'] = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows CE; PPC;
240x320)'
response = requester(origUrl, {}, headers, True, 0).text
parsed = zetanize(origUrl, response)
if isProtected(parsed):
    print ('%s CSRF protection is enabled for mobile browsers as well.' % bad)
else:
    print ('%s CSRF protection isn\'t enabled for mobile browsers.' % good)

print ('%s Making a request without CSRF token parameter.' % run)

data = tweaker(origData, 'remove')
response = requester(origUrl, data, headers, origGET, 0)
if response.status_code == originalCode:
    if str(originalCode)[0] in ['4', '5']:
        print ('%s It didn\'t work' % bad)
    else:
        difference = abs(originalLength - len(response.text))
        if difference <= tolerableDifference:
            print ('%s It worked!' % good)
else:
    print ('%s It didn\'t work' % bad)

print ('%s Making a request without CSRF token parameter value.' % run)
data = tweaker(origData, 'clear')

response = requester(origUrl, data, headers, origGET, 0)
if response.status_code == originalCode:
    if str(originalCode)[0] in ['4', '5']:
        print ('%s It didn\'t work' % bad)
    else:
        difference = abs(originalLength - len(response.text))
        if difference <= tolerableDifference:
            print ('%s It worked!' % good)
else:
    print ('%s It didn\'t work' % bad)


seeds = ranger(allTokens)

print ('%s Checking if tokens are checked to a specific length' % run)

for index in range(len(allTokens[0])):
    data = tweaker(origData, 'replace', index=index, seeds=seeds)
    response = requester(origUrl, data, headers, origGET, 0)
    if response.status_code == originalCode:
        if str(originalCode)[0] in ['4', '5']:
            break
        else:
            difference = abs(originalLength - len(response.text))
            if difference <= tolerableDifference:
                print ('%s Last %i chars of token aren\'t being checked' %
                       (good, index + 1))
    else:
```

```
        break

print ('%s Generating a fake token.' % run)

data = tweaker(origData, 'generate', seeds=seeds)
print ('%s Making a request with the self generated token.' % run)

response = requester(origUrl, data, headers, origGET, 0)
if response.status_code == originalCode:
    if str(originalCode)[0] in ['4', '5']:
        print ('%s It didn\'t work' % bad)
    else:
        difference = abs(originalLength - len(response.text))
        if difference <= tolerableDifference:
            print ('%s It worked!' % good)
else:
    print ('%s It didn\'t work' % bad)

print (' %s Phase: Analysing %s[%s6/6%s]%s' %
       (lightning, green, end, green, end))

binary = stringToBinary(''.join(allTokens))
result = isRandom(binary)
for name, result in result.items():
    if not result:
        print ('%s %s : %s%s%s' % (good, name, green, 'non-random', end))
    else:
        print ('%s %s : %s%s%s' % (bad, name, red, 'random', end))
```

Admin side Views.py

```
from django.shortcuts import render,HttpResponse
from django.contrib import messages
from users.models import UserRegistrationModel,CSRFResponse
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger
import pandas as pd
# Create your views here.
def AdminLoginCheck(request):
    if request.method == 'POST':
        usrid = request.POST.get('loginname')
        pswd = request.POST.get('pswd')
        print("User ID is = ", usrid)
        if 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 AdminViewUsers(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 adminviewallcsrfs(request):
    data_list = CSRFResponse.objects.all()
    page = request.GET.get('page', 1)

    paginator = Paginator(data_list, 60)
    try:
        users = paginator.page(page)
    except PageNotAnInteger:
        users = paginator.page(1)
    except EmptyPage:
        users = paginator.page(paginator.num_pages)

    return render(request, 'admins/viewAllCSRFS.html', {'users': users})

def PostRequestdata(request):
    df = pd.read_csv('./media/dataset/features_matrix.csv', sep=',', delimiter=None,
header='infer', names=None,
                     index_col=None, usecols=None, squeeze=False, prefix=None,
mangle_dupe_cols=True, dtype=None,
                     engine=None, converters=None, true_values=None,
false_values=None, skipinitialspace=False,
                     skiprows=None, skipfooter=0, nrows=None, na_values=None,
keep_default_na=True, na_filter=True,
                     verbose=False, skip_blank_lines=True, parse_dates=False,
infer_datetime_format=False,
                     keep_date_col=False, date_parser=None, dayfirst=False,
cache_dates=True, iterator=False,
                     chunksize=None, compression='infer', thousands=None,
decimal='.', lineterminator=None,
                     quotechar='"', quoting=0, doublequote=True, escapechar=None,
comment=None, encoding=None,
                     dialect=None, error_bad_lines=True, warn_bad_lines=True,
delim_whitespace=False, low_memory=True,
                     memory_map=False, float_precision=None)
    data = df[['numOfParams', 'numOfBools', 'numOfIds', 'numOfBlobs', 'reqLen',
'isPOST']]
    data = data.to_html()
    #print(data)

    return render(request, "admins/PostviewData.html",{"data":data})

def GetRequestdata(request):
    df = pd.read_csv('./media/dataset/features_matrix.csv', sep=',', delimiter=None,
header='infer', names=None,
                     index_col=None, usecols=None, squeeze=False, prefix=None,
```

```
mangle_dupe_cols=True, dtype=None,
                     engine=None, converters=None, true_values=None,
false_values=None, skipinitialspace=False,
                     skiprows=None, skipfooter=0, nrows=None, na_values=None,
keep_default_na=True, na_filter=True,
                     verbose=False, skip_blank_lines=True, parse_dates=False,
infer_datetime_format=False,
                     keep_date_col=False, date_parser=None, dayfirst=False,
cache_dates=True, iterator=False,
                     chunksize=None, compression='infer', thousands=None,
decimal='.', lineterminator=None,
                     quotechar='"', quoting=0, doublequote=True, escapechar=None,
comment=None, encoding=None,
                     dialect=None, error_bad_lines=True, warn_bad_lines=True,
delim_whitespace=False, low_memory=True,
                     memory_map=False, float_precision=None)
    data = df[['numOfParams', 'numOfBools', 'numOfIds', 'numOfBlobs', 'reqLen',
'isGET']]
    data = data.to_html()

    return render(request, "admins/GetviewData.html", {"data": data})
```

All Ursl.py

```
"""WebVulnerability 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 WebVulnerability 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("index/", mainView.logout, name="index"),
    path("", 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 Views #####
    path("UserRegisterActions/", usr.UserRegisterActions,
name="UserRegisterActions"),
    path("UserLoginCheck/", usr.UserLoginCheck, name="UserLoginCheck"),
    path("UserHome/",usr.UserHome, name="UserHome"),
    path("UserPreProcessForm/", usr.UserPreProcessForm, name="UserPreProcessForm"),
    path("UserCSRFProcessByBolt/", usr.UserCSRFProcessByBolt,
name="UserCSRFProcessByBolt"),
    path("UserMitchProcess/", usr.UserMitchProcess, name="UserMitchProcess"),
    path("UserMachineLearning/", usr.UserMachineLearning,
name="UserMachineLearning"),



    #####Admin Side Views #######
    path("AdminLoginCheck/", admins.AdminLoginCheck, name="AdminLoginCheck"),
    path("AdminHome/", admins.AdminHome, name="AdminHome"),
    path("AdminViewUsers/", admins.AdminViewUsers, name="AdminViewUsers"),
    path("AdminActivaUsers/", admins.AdminActivaUsers, name="AdminActivaUsers"),
    path("adminviewallcsrfs/", admins.adminviewallcsrfs, name="adminviewallcsrfs"),
    path("PostRequestdata/", admins.PostRequestdata, name="PostRequestdata"),
    path("GetRequestdata/", admins.GetRequestdata, name="GetRequestdata"),




]
urlpatterns += staticfiles_urlpatterns()
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Base.html

```
{%load static%}
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
fit=no">
    <meta name="description" content="">
    <meta name="author" content="TemplateMo">
    <link
href="https://fonts.googleapis.com/css?family=Poppins:100,200,300,400,500,600,700,800
,900&display=swap" rel="stylesheet">
    <title>Finance Business HTML5 Template</title>
    <!-- Bootstrap core CSS -->
    <link href="{%static 'vendor/bootstrap/css/bootstrap.min.css'%}"
rel="stylesheet">
    <!-- Additional CSS Files -->
    <link rel="stylesheet" href="{%static 'assets/css/fontawesome.css'%}">
    <link rel="stylesheet" href="{%static 'assets/css/templatemo-finance-
business.css'%}">
    <link rel="stylesheet" href="{%static 'assets/css/owl.css'%}">
```

```
  </head>

  <body>
    <div id="preloader">
        <div class="jumper">
            <div></div>
            <div></div>
            <div></div>
        </div>
    </div>
    <header class="">
      <nav class="navbar navbar-expand-lg">
        <div class="container">
          <a class="navbar-brand" href="index.html"><h2>Web Vulnerability</h2></a>
          <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarResponsive" aria-controls="navbarResponsive" aria-expanded="false"
aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
          </button>
          <div class="collapse navbar-collapse" id="navbarResponsive">
            <ul class="navbar-nav ml-auto">
              <li class="nav-item">
                <a class="nav-link" href="{%url 'index'%}">Home</a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="{%url 'UserLogin'%}">User</a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="{%url 'AdminLogin'%}">Admin</a>
              </li>

              <li class="nav-item">
                <a class="nav-link" href="{%url 'UserRegister'%}">Registrations</a>
              </li>
            </ul>
          </div>
        </div>
      </nav>
    </header>

    <!-- Page Content -->
    <!-- Banner Starts Here -->
    {%block contents%}

    {%endblock%}


    <div class="services">
      <div class="container">
        <div class="row">
          <div class="col-md-12">
            <div class="section-heading">
              <h2>Web Vulnerability  <em>Detection</em></h2>
              <span>use supervised learning to automatically train a classifier
which partitions selected web objects of interest, e.g.,
```

```
HTTP requests, HTTP responses or cookies, based on
the web application semantics. For example, in the case
of CSRF detection, the classifier would be used to
identify security-sensitive HTTP requests</span>
            </div>
          </div>
                  </div>
      </div>
    </div>
    <div class="sub-footer">
      <div class="container">
        <div class="row">
          <div class="col-md-12">
            <p>Copyright &copy; 2020 Alex and Co., Ltd.

            - Design: <a rel="nofollow noopener" href="Google"
target="_blank">TemplateMo</a></p>
          </div>
        </div>
      </div>
    </div>

    <!-- Bootstrap core JavaScript -->
    <script src="{% static 'vendor/jquery/jquery.min.js'%}"></script>
    <script src="{% static 'vendor/bootstrap/js/bootstrap.bundle.min.js'%}"></script>

    <!-- Additional Scripts -->
    <script src="{% static 'assets/js/custom.js'%}"></script>
    <script src="{% static 'assets/js/owl.js'%}"></script>
    <script src="{% static 'assets/js/slick.js'%}"></script>
    <script src="{% static 'assets/js/accordions.js'%}"></script>

    <script language = "text/Javascript">
      cleared[0] = cleared[1] = cleared[2] = 0; //set a cleared flag for each field
      function clearField(t){                  //declaring the array outside of the
      if(! cleared[t.id]){                     // function makes it static and
global
          cleared[t.id] = 1;  // you could use true and false, but that's more typing
          t.value='';         // with more chance of typos
          t.style.color='#fff';
          }
      }
    </script>

  </body>
</html>
```

UserRegistrations.html

```
{%extends 'base.html'%}

{%block contents%}

<div class="main-banner header-text" id="top">
```

```
        <div class="Modern-Slider">
          <div class="item item-1">
            <div class="img-fill">
                <center>
                <div class="text-content"><br/><br/><br/><br/>
                  <h4>User Registration Here</h4>
                  <p>
                      <form action="{%url 'UserRegisterActions'%}" method="POST"
class="text-primary" style="width:100%">
                                        {% csrf_token %}
                                        <table>
                                            <tr>
                                                <td class="text-primary">Customer
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></td>
                                                <td>{{form.status}}</td>
                                            </tr>

                                            <tr>
                                                <td>
```

```
                                        <button class="btn btn-primary my-2
my-sm-0" style="margin-left:20%;"
                                                type="submit">
                                            Register
                                        </button>
                                    </td>
                                </tr>

                                {% if messages %}
                                {% for message in messages %}
                                <font color='GREEN'> {{ message }}</font>
                                {% endfor %}
                                {% endif %}

                            </table>

                        </form>
                    </p>
                </div>
                    </center>
            </div>
          </div>
        </div>
    </div>


{%endblock%}
```