**URL:**

from django.urls import path

from . import views

urlpatterns = [path("index.html", views.index, name="index"),

               path("UserLogin.html", views.UserLogin, name="UserLogin"),

               path("UserLoginAction", views.UserLoginAction, name="UserLoginAction"),

           path("Register.html", views.Register, name="Register"),

               path("RegisterAction", views.RegisterAction, name="RegisterAction"),

               path("RunRandom", views.RunRandom, name="RunRandom"),

               path("RunDT.html", views.RunDT, name="RunDT"),

               path("RunSVM", views.RunSVM, name="RunSVM"),

               path("Chatbot", views.Chatbot, name="Chatbot"),

           path("ChatData", views.ChatData, name="ChatData"),

]

**Views.py:**

from django.shortcuts import render

from django.template import RequestContext

from django.contrib import messages

from django.http import HttpResponse

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import pickle

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics import accuracy\_score

from sklearn import svm

from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import confusion\_matrix

import seaborn as sns

import pymysql

import random

global precision, recall, fscore, accuracy, uname

X = np.load("model/X.txt.npy")

Y = np.load("model/Y.txt.npy")

indices = np.arange(X.shape[0])

np.random.shuffle(indices)

X = X[indices]

Y = Y[indices]

with open('model/tfidf.txt', 'rb') as file:

    tfidf = pickle.load(file)

file.close()

X = tfidf.fit\_transform(X).toarray()

print(X.shape)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2)

if os.path.exists('model/svm.txt'):

    with open('model/svm.txt', 'rb') as file:

        svm\_cls = pickle.load(file)

    file.close()

else:

    svm\_cls = svm.SVC()

    svm\_cls.fit(X\_train, y\_train)

    with open('model/svm.txt', 'wb') as file:

        pickle.dump(svm\_cls, file)

    file.close()

if os.path.exists('model/rf.txt'):

    with open('model/rf.txt', 'rb') as file:

        rf\_cls = pickle.load(file)

    file.close()

else:

    rf\_cls = RandomForestClassifier()

    rf\_cls.fit(X\_train, y\_train)

    with open('model/rf.txt', 'wb') as file:

        pickle.dump(rf\_cls, file)

    file.close()

if os.path.exists('model/dt.txt'):

    with open('model/dt.txt', 'rb') as file:

        dt\_cls = pickle.load(file)

    file.close()

else:

    dt\_cls = RandomForestClassifier()

    dt\_cls.fit(X\_train, y\_train)

    with open('model/dt.txt', 'wb') as file:

        pickle.dump(dt\_cls, file)

    file.close()

def RunSVM(request):

    if request.method == 'GET':

        global precision, recall, fscore, accuracy

        global X\_train, X\_test, y\_train, y\_test

        predict = svm\_cls.predict(X\_test)

        acc = accuracy\_score(y\_test,predict)\*100

        p = precision\_score(y\_test,predict,average='macro') \* 100

        r = recall\_score(y\_test,predict,average='macro') \* 100

        f = f1\_score(y\_test,predict,average='macro') \* 100

        precision.append(p)

        recall.append(r)

        fscore.append(f)

        accuracy.append(acc)

        output = ""

        output+='<tr><td><font size="" color="black">Random Forest</td>'

        output+='<td><font size="" color="black">'+str(accuracy[0])+'</td>'

        output+='<td><font size="" color="black">'+str(precision[0])+'</td>'

        output+='<td><font size="" color="black">'+str(recall[0])+'</td>'

        output+='<td><font size="" color="black">'+str(fscore[0])+'</td>'

        output+='<tr><td><font size="" color="black">Decision Tree</td>'

        output+='<td><font size="" color="black">'+str(accuracy[1])+'</td>'

        output+='<td><font size="" color="black">'+str(precision[1])+'</td>'

        output+='<td><font size="" color="black">'+str(recall[1])+'</td>'

        output+='<td><font size="" color="black">'+str(fscore[1])+'</td>'

        output+='<tr><td><font size="" color="black">SVM</td>'

        output+='<td><font size="" color="black">'+str(accuracy[2])+'</td>'

        output+='<td><font size="" color="black">'+str(precision[2])+'</td>'

        output+='<td><font size="" color="black">'+str(recall[2])+'</td>'

        output+='<td><font size="" color="black">'+str(fscore[2])+'</td>'

        LABELS = ['Normal URL','Phishing URL']

        conf\_matrix = confusion\_matrix(y\_test, predict)

        plt.figure(figsize =(6, 6))

        ax = sns.heatmap(conf\_matrix, xticklabels = LABELS, yticklabels = LABELS, annot = True, cmap="viridis" ,fmt ="g");

        ax.set\_ylim([0,2])

        plt.title("SVM Confusion matrix")

        plt.ylabel('True class')

        plt.xlabel('Predicted class')

        plt.show()

        context= {'data':output}

        return render(request, 'ViewOutput.html', context)

def RunDT(request):

    if request.method == 'GET':

        global precision, recall, fscore, accuracy

        global X\_train, X\_test, y\_train, y\_test

        predict = dt\_cls.predict(X\_test)

        acc = accuracy\_score(y\_test,predict)\*100

        p = precision\_score(y\_test,predict,average='macro') \* 100

        r = recall\_score(y\_test,predict,average='macro') \* 100

        f = f1\_score(y\_test,predict,average='macro') \* 100

        precision.append(p)

        recall.append(r)

        fscore.append(f)

        accuracy.append(acc)

        output = ""

        output+='<tr><td><font size="" color="black">Random Forest</td>'

        output+='<td><font size="" color="black">'+str(accuracy[0])+'</td>'

        output+='<td><font size="" color="black">'+str(precision[0])+'</td>'

        output+='<td><font size="" color="black">'+str(recall[0])+'</td>'

        output+='<td><font size="" color="black">'+str(fscore[0])+'</td>'

        output+='<tr><td><font size="" color="black">Decision Tree</td>'

        output+='<td><font size="" color="black">'+str(accuracy[1])+'</td>'

        output+='<td><font size="" color="black">'+str(precision[1])+'</td>'

        output+='<td><font size="" color="black">'+str(recall[1])+'</td>'

        output+='<td><font size="" color="black">'+str(fscore[1])+'</td>'

        LABELS = ['Normal URL','Phishing URL']

        conf\_matrix = confusion\_matrix(y\_test, predict)

        plt.figure(figsize =(6, 6))

        ax = sns.heatmap(conf\_matrix, xticklabels = LABELS, yticklabels = LABELS, annot = True, cmap="viridis" ,fmt ="g");

        ax.set\_ylim([0,2])

        plt.title("Decision Tree Confusion matrix")

        plt.ylabel('True class')

        plt.xlabel('Predicted class')

        plt.show()

        context= {'data':output}

        return render(request, 'ViewOutput.html', context)

def RunRandom(request):

    if request.method == 'GET':

        global precision, recall, fscore, accuracy, rf\_cls

        global X\_train, X\_test, y\_train, y\_test

        precision = []

        recall = []

        fscore = []

        accuracy = []

        predict = rf\_cls.predict(X\_test)

        acc = accuracy\_score(y\_test,predict)\*100

        p = precision\_score(y\_test,predict,average='macro') \* 100

        r = recall\_score(y\_test,predict,average='macro') \* 100

        f = f1\_score(y\_test,predict,average='macro') \* 100

        precision.append(p)

        recall.append(r)

        fscore.append(f)

        accuracy.append(acc)

        output = ""

        output+='<tr><td><font size="" color="black">Random Forest</td>'

        output+='<td><font size="" color="black">'+str(accuracy[0])+'</td>'

        output+='<td><font size="" color="black">'+str(precision[0])+'</td>'

        output+='<td><font size="" color="black">'+str(recall[0])+'</td>'

        output+='<td><font size="" color="black">'+str(fscore[0])+'</td>'

        LABELS = ['Normal URL','Phishing URL']

        conf\_matrix = confusion\_matrix(y\_test, predict)

        plt.figure(figsize =(6, 6))

        ax = sns.heatmap(conf\_matrix, xticklabels = LABELS, yticklabels = LABELS, annot = True, cmap="viridis" ,fmt ="g");

        ax.set\_ylim([0,2])

        plt.title("Random Forest Confusion matrix")

        plt.ylabel('True class')

        plt.xlabel('Predicted class')

        plt.show()

        context= {'data':output}

        return render(request, 'ViewOutput.html', context)

def getData(arr):

    data = ""

    for i in range(len(arr)):

        arr[i] = arr[i].strip()

        if len(arr[i]) > 0:

            data += arr[i]+" "

    return data.strip()

def PredictAction(url\_input):

    global rf\_cls, tfidf

    test = []

    arr = url\_input.split("/")

    output = "Unable to predict"

    if len(arr) > 0:

        data = getData(arr)

        #print(data)

        test.append(data)

        test = tfidf.transform(test).toarray()

        #print(test)

        #print(test.shape)

        predict = rf\_cls.predict(test)

        #print(predict)

        predict = predict[0]

        if predict == 0:

            output = "Genuine URL"

        if predict == 1:

            output = "Contains PHISHING URL"

    return output

def getAccountNo(question):

    acc = "none"

    arr = question.split(" ")

    for i in range(len(arr)):

        try:

            acc = int(arr[i].strip())

            break

        except Exception:

            pass

    return acc

def getBalance(acc\_no):

    output = "invalid account no"

    con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database = 'EvilChatbot',charset='utf8')

    deposit = 0

    debit = 0

    balance = 0

    with con:

        cur = con.cursor()

        cur.execute("select sum(transaction\_amount) from account where acc\_no='"+str(acc\_no)+"' and transaction\_type='deposit'")

        rows = cur.fetchall()

        for row in rows:

            if row[0] is not None:

                deposit = float(row[0])

            break

    with con:

        cur = con.cursor()

        cur.execute("select sum(transaction\_amount) from account where acc\_no='"+str(acc\_no)+"' and transaction\_type='debit'")

        rows = cur.fetchall()

        for row in rows:

            if row[0] is not None:

                debit = float(row[0])

            break

    if deposit > 0:

        balance = deposit - debit

        output = "Your current balance is : "+str(balance)

    return output

def getDebit(acc\_no):

    output = "Transaction No, Transaction Type, Transaction Amount, Balance, Transaction Date\n"

    con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database = 'EvilChatbot',charset='utf8')

    with con:

        cur = con.cursor()

        cur.execute("select \* from account where acc\_no='"+str(acc\_no)+"' and transaction\_type='debit'")

        rows = cur.fetchall()

        for row in rows:

            output += row[0]+", "+row[1]+", "+row[2]+", "+row[3]+", "+row[4]+"\n"

    return output

def getDeposit(acc\_no):

    output = "Transaction No, Transaction Type, Transaction Amount, Balance, Transaction Date\n"

    con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database = 'EvilChatbot',charset='utf8')

    with con:

        cur = con.cursor()

        cur.execute("select \* from account where acc\_no='"+str(acc\_no)+"' and transaction\_type='deposit'")

        rows = cur.fetchall()

        for row in rows:

            output += row[0]+", "+row[1]+", "+row[2]+", "+row[3]+", "+row[4]+"\n"

    return output

def ChatData(request):

    if request.method == 'GET':

        global email

        links= ['http://www.google.com', 'http://mail.google.com', 'aws.amazon.com', 'http://www.yahoo.com', 'http://allrecipes.com/video/466/parmesan-cheese-twists/detail.aspx?prop24=VH\_Brands',

                'http://askubuntu.com/questions/239450/nvidia-7800-gtx-drivers-not-working-for-12-04', 'http://bestblackhatforum.com/Thread-Become-VIP-Is-Your-Best-Bet-Why?action=lastpost']

        link = links[random.randint(0,len(links))]

        malicious\_output = PredictAction(link)

        question = request.GET.get('mytext', False)

        question = question.strip("\n").strip()

        question = question.lower()

        acc\_no = getAccountNo(question)

        print(str(acc\_no)+" ====")

        output = "Did not get your question. Please retry"

        if "balance" in question:

            output = getBalance(acc\_no)

        if "debit" in question:

            output = getDebit(acc\_no)

        if "deposit" in question:

            output = getDeposit(acc\_no)

        return HttpResponse("Chatbot: "+output+"#"+link+"#"+malicious\_output, content\_type="text/plain")

def index(request):

    if request.method == 'GET':

       return render(request, 'index.html', {})

def Predict(request):

    if request.method == 'GET':

       return render(request, 'Predict.html', {})

def UserLogin(request):

    if request.method == 'GET':

        return render(request, 'UserLogin.html', {})

def Chatbot(request):

    if request.method == 'GET':

        return render(request, 'Chatbot.html', {})

def UserLoginAction(request):

    if request.method == 'POST':

        global uname

        user = request.POST.get('t1', False)

        password = request.POST.get('t2', False)

        index = 0

        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database = 'EvilChatbot',charset='utf8')

        with con:

            cur = con.cursor()

            cur.execute("select \* FROM register")

            rows = cur.fetchall()

            for row in rows:

                if row[0] == user and password == row[1]:

                    uname = user

                    index = 1

                    break

        if index == 1:

            context= {'data':'Welcome '+user}

            return render(request, 'UserScreen.html', context)

        else:

            context= {'data':'Invalid Login'}

            return render(request, 'UserLogin.html', context)

def Register(request):

    if request.method == 'GET':

        return render(request, 'Register.html', {})

def RegisterAction(request):

    if request.method == 'POST':

        username = request.POST.get('t1', False)

        password = request.POST.get('t2', False)

        contact = request.POST.get('t3', False)

        email = request.POST.get('t4', False)

        address = request.POST.get('t5', False)

        status = "none"

        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database = 'EvilChatbot',charset='utf8')

        with con:

            cur = con.cursor()

            cur.execute("select username FROM register where username='"+username+"'")

            rows = cur.fetchall()

            for row in rows:

                status = "Username Already Exists"

                break

        if status == "none":

            db\_connection = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database = 'EvilChatbot',charset='utf8')

            db\_cursor = db\_connection.cursor()

            student\_sql\_query = "INSERT INTO register(username,password,contact,email,address) VALUES('"+username+"','"+password+"','"+contact+"','"+email+"','"+address+"')"

            db\_cursor.execute(student\_sql\_query)

            db\_connection.commit()

            print(db\_cursor.rowcount, "Record Inserted")

            if db\_cursor.rowcount == 1:

                status = 'Signup Process Completed'

        context= {'data': status}

        return render(request, 'Register.html', context)

**Settings.py**

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.mysql',

        'NAME': 'EvilChatbot',

        'HOST': '127.0.0.1',

        'PORT': '3306',

        'USER': 'root',

        'PASSWORD': 'root',

    'OPTIONS': {

          'autocommit': True,

        },

    }

}

**Manage.py :**

#!/usr/bin/env python

import os

import sys

if \_\_name\_\_ == '\_\_main\_\_':

    os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'Chatbot.settings')

    try:

        from django.core.management import execute\_from\_command\_line

    except ImportError as exc:

        raise ImportError(

            "Couldn't import Django. Are you sure it's installed and "

            "available on your PYTHONPATH environment variable? Did you "

            "forget to activate a virtual environment?"

        ) from exc

    execute\_from\_command\_line(sys.argv)

**CHAPTER 5**

**UML DIAGRAMS**

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

**GOALS:** The Primary goals in the design of the UML are as follows:

* Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
* Provide extendibility and specialization mechanisms to extend the core concepts.
* Be independent of particular programming languages and development process.
* Provide a formal basis for understanding the modeling language.
* Encourage the growth of OO tools market.
* Support higher level development concepts such as collaborations, frameworks, patterns and components.
* Integrate best practices.

**Class diagram**

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram was capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.



Figure-5.1: Class Diagram

**Sequence Diagram**

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows, as parallel vertical lines (“lifelines”), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

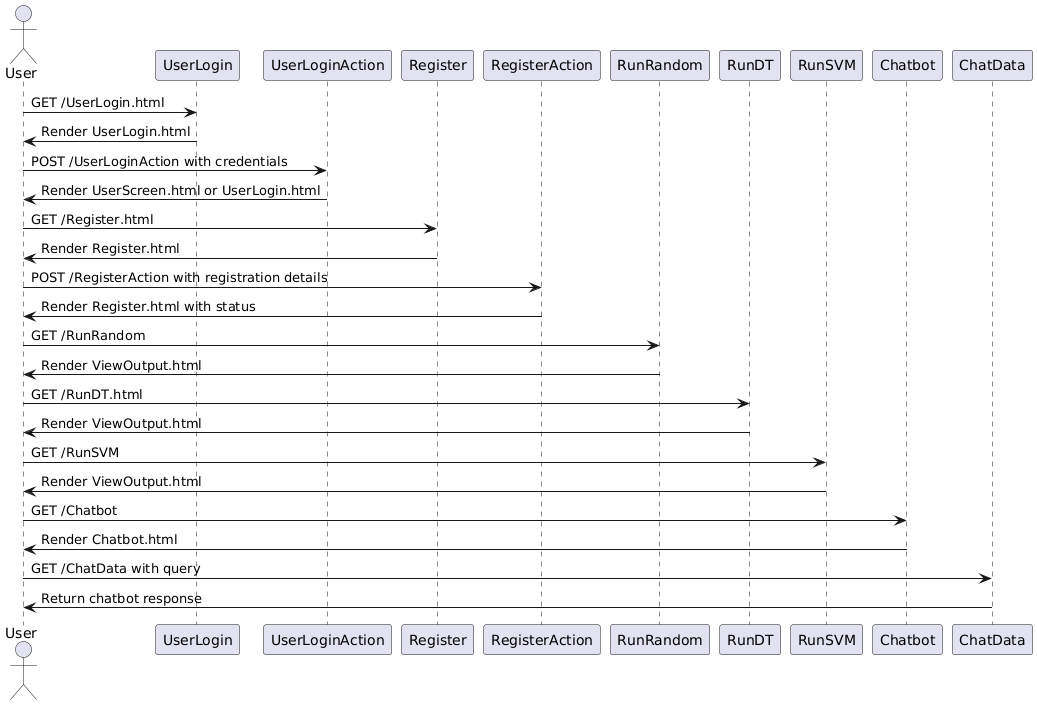


Figure-5.2: Sequence Diagram

**Activity diagram**

Activity diagrams are graphical representations of Workflows of stepwise activities and actions with support for choice, iteration, and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

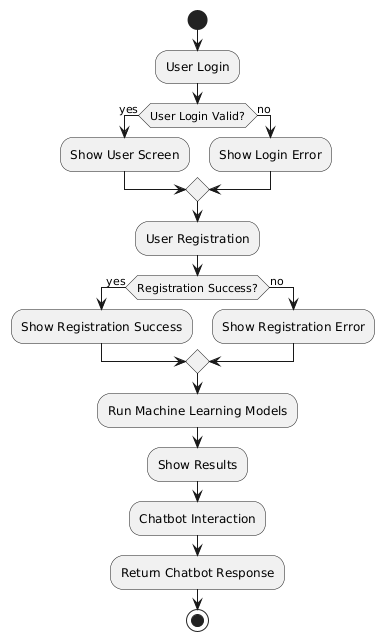


Figure-5.3: Activity Diagram

**Data flow diagram**

A data flow diagram (DFD) is a graphical representation of how data moves within an information system. It is a modeling technique used in system analysis and design to illustrate the flow of data between various processes, data stores, data sources, and data destinations within a system or between systems. Data flow diagrams are often used to depict the structure and behavior of a system, emphasizing the flow of data and the transformations it undergoes as it moves through the system.

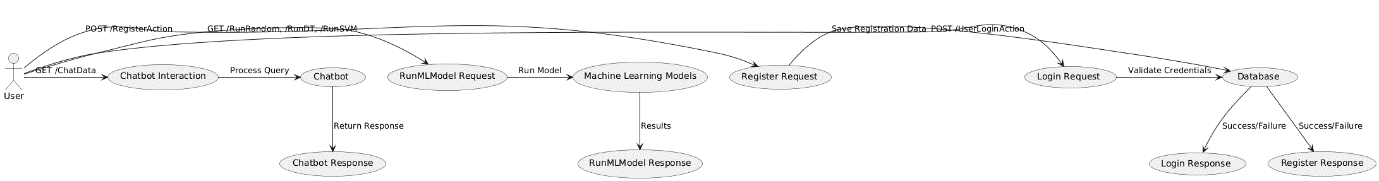


Figure-5.4: Dataflow Diagram

**Component diagram:** Component diagram describes the organization and wiring of the physical components in a system.

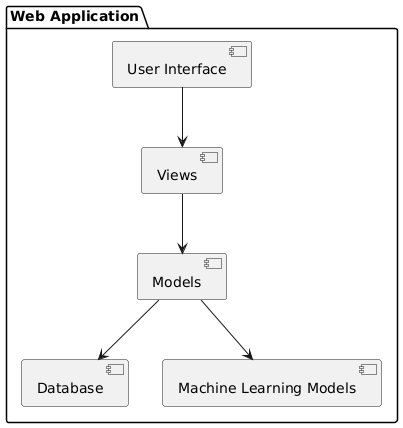


Figure-5.5: Component Diagram

**Use Case diagram:** A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

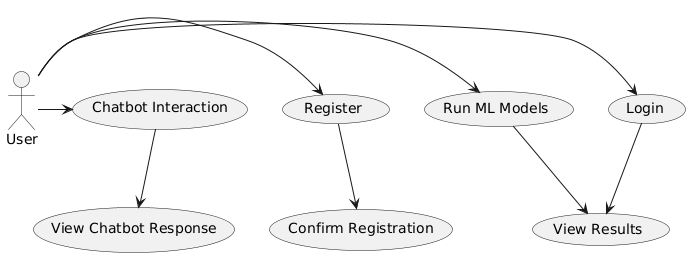


Figure-5.6: use case diagram

**Deployment Diagram:**

A deployment diagram in UML illustrates the physical arrangement of hardware and software components in the system. It visualizes how different software artifacts, such as data processing scripts and model training components, are deployed across hardware nodes and interact with each other, providing insight into the system’s infrastructure and deployment strategy.

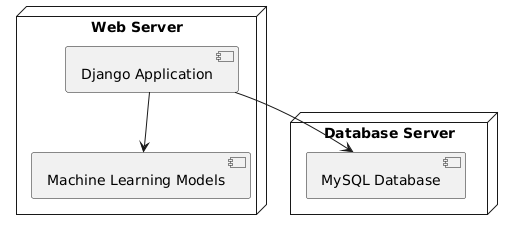


Figure-5.7: DeploymentDiagram

**Architectural Block Diagram**

An architectural block diagram offers a high-level view of a system’s structure, showcasing the main components and their interactions. It represents how major modules, such as data sources, processing units, and evaluation components, are organized and how they communicate with each other to accomplish the system’s objectives. This diagram helps in understanding the overall design and flow of the system.

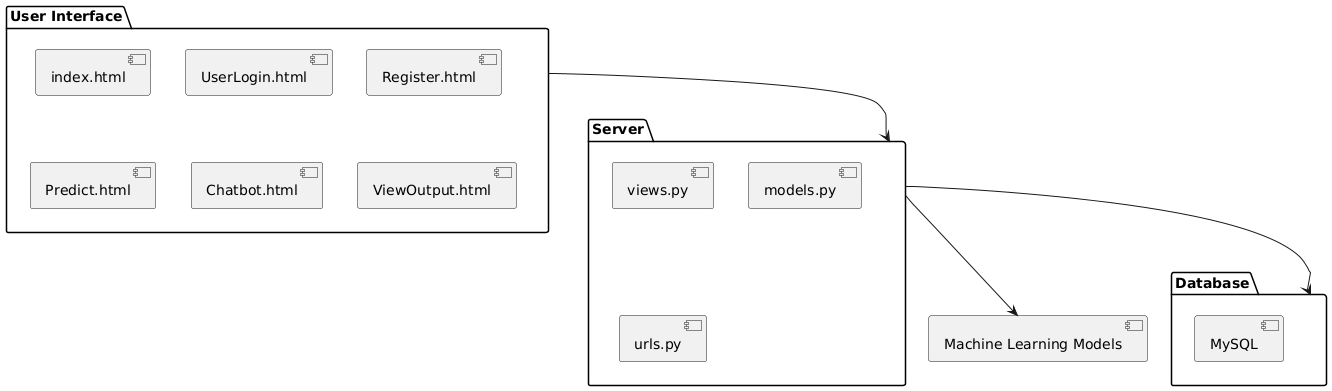


Figure-5.8: architectural block diagram