**Department of Information Technology**

**University of the Punjab**

**Gujranwala Campus**



**Project: Computer Vision**

**Project Title: Currency Detector**

**Submitted BY**

**Muhammad Sohaib Raza (BSE21033)**

**Danish Iqbal (BSE21025)**

**TO**

**Miss Nadia Yousaf**

**Dated: 04/09/2024**

**Project Documentation**

The "cuurrency flag" directory contains the following items:

.vscode: Likely contains settings for the Visual Studio Code editor.

pycache: Contains compiled Python files.

app.py: A Python file that might be the main script for the project.

imagesQuery: Possibly contains image-related data or queries.

static: Typically used for static files like CSS, JavaScript, or images in web applications.

Templates: Likely contains HTML templates for rendering web pages.

uploads: Might store files that are uploaded through the application.

I'll start by looking into the app.py file, as it's likely the main script of the project. ​​

The app.py file is a Flask-based web application that appears to involve image processing. Here's an overview of what the file contains so far:

**Imports**

Flask: For creating the web application.

cv2: OpenCV, used for computer vision tasks.

os: For handling file paths and directories.

requests: For making HTTP requests (though not yet used in the snippet).

numpy: For numerical operations.

secure\_filename: To safely handle filenames in uploads.

**Key Components**

Configuration:

UPLOAD\_FOLDER: Directory for storing uploaded files.

ALLOWED\_EXTENSIONS: Set of allowed file extensions for uploads.

ORB Detector:

Initializes an ORB (Oriented FAST and Rotated BRIEF) detector for feature detection.

Prepares to load a dataset of images from the ImagesQuery directory.

**Data Preparation:**

Loads images from the ImagesQuery folder.

Extracts and stores the class names (file names without extensions) for reference.

**Functionality:**

extract\_features(image): A function to extract keypoints and descriptors from an image using ORB.

Next, I'll provide a more detailed documentation based on the remaining parts of the app.py file. ​​

It seems like I can’t do more advanced data analysis right now. Please try again later. However, I can help answer any questions or provide a summary based on what we've reviewed so far. Let me know how you'd like to proceed! ​​

**Currency Detector Code**

**HTML Code**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Fake Currency Detector</title>

<style>

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

font-family: Arial, sans-serif;

background: url('/static/back.jpg') no-repeat center center fixed;

background-size: cover;

display: flex;

justify-content: center;

align-items: center;

flex-direction: column;

height: 100vh;

}

header {

background-color: #335;

color: #fff;

padding: 10px 0;

width: 100%;

position: fixed;

top: 0;

left: 0;

z-index: 1000;

}

.container1 {

max-width: 1200px;

margin: 0 auto;

padding: 0 20px;

display: flex;

align-items: center;

}

.logo {

display: flex;

align-items: center;

}

.logo img {

width:55px;

height: 50px; /\* Adjust size of the logo image as needed \*/

margin-right: 20px;

border-radius:50%;

}

.logo span {

font-size: 24px;

font-weight: bold;

}

.container {

text-align: center;

background: rgba(0, 0, 0, 0.5);

padding: 20px;

border-radius: 8px;

margin-top: 80px; /\* Add margin to push it below the header \*/

height: 300px;

display: flex;

justify-content: center;

align-items: center;

}

.content .card {

padding: 20px;

border-radius: 8px;

color: white;

}

.content .card h2 {

font-size: 22px;

margin-bottom: 10px;

}

.content .card p {

font-size: 14px;

margin-bottom: 20px;

}

.file-upload-label {

display: inline-block;

padding: 10px 20px;

font-size: 16px;

color: #fff;

background-color: #007bff;

border: none;

border-radius: 5px;

cursor: pointer;

text-align: center;

margin-right: 20px;

margin-left: 30px;

}

/\* Hide the default file input \*/

.file-upload-input {

display: none;

}

/\* Add styling to the file input wrapper \*/

.file-upload-wrapper {

display: inline-block;

position: relative;

}

</style>

</head>

<body>

<header>

<div class="container1">

<div class="logo">

<img src="/static/logo.png" alt="Logo"> <!-- Replace with your logo image path -->

<span>Fake Currency Detector</span>

</div>

</div>

</header>

<div class="container">

<div class="content">

<div class="card">

<h2>SELECT NOTES</h2>

<p>Please select note picture by clicking the button below!</p>

<form action="/" method="post" enctype="multipart/form-data">

<div class="file-upload-wrapper">

<label for="file-upload" class="file-upload-label">Select Note </label>

<input type="file" id="file-upload" name="file" class="file-upload-input">

</div>

<input type="submit" value="Detect" class="file-upload-label">

</form>

</div>

</div>

</div>

</body>

</html>

**Python Code**

from flask import Flask, render\_template, request, redirect, url\_for, send\_from\_directory

import cv2

import os

import requests

import numpy as np

from werkzeug.utils import secure\_filename

app = Flask(\_\_name\_\_)

# Configure the upload folder and allowed file extensions

UPLOAD\_FOLDER = 'uploads/'

ALLOWED\_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif', 'jfif'}

app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER

# Ensure the upload directory exists

os.makedirs(UPLOAD\_FOLDER, exist\_ok=True)

# Initialize ORB detector and load dataset

orb = cv2.ORB\_create(nfeatures=1000)

path = 'ImagesQuery'

ClassNames = []

images = []

# Load dataset images and associate with note names

myList = os.listdir(path)

for cl in myList:

    imgCur = cv2.imread(f"{path}/{cl}", 0)

    if imgCur is not None:

        images.append(imgCur)

        ClassNames.append(os.path.splitext(cl)[0])  # Store the filename without extension

def extract\_features(image):

    """Extract keypoints and descriptors from the image using ORB."""

    keypoints, descriptors = orb.detectAndCompute(image, None)

    return keypoints, descriptors

def findDes(images):

    """Find descriptors for all images in the dataset."""

    desList = []

    for img in images:

        \_, des = extract\_features(img)

        desList.append(des if des is not None else np.array([]))  # Handle missing descriptors

    return desList

def match\_image(input\_image\_descriptors, dataset\_descriptors):

    """Match the input image descriptors with the dataset and find the best match."""

    bf = cv2.BFMatcher(cv2.NORM\_HAMMING, crossCheck=True)

    best\_match\_index = -1

    max\_good\_matches = 0

    good\_match\_threshold = 30  # You can adjust this threshold for stricter matching

    for i, descriptors in enumerate(dataset\_descriptors):

        if len(descriptors) == 0 or len(input\_image\_descriptors) == 0:

            continue

        matches = bf.match(descriptors, input\_image\_descriptors)

        good\_matches = [m for m in matches if m.distance < 50]  # Only count matches with a low distance

        if len(good\_matches) > good\_match\_threshold:  # Only consider as a match if it meets the threshold

            return i  # Return immediately if a strong match is found

    return None

def detect\_currency(input\_image\_path):

    """Detect the specific Pakistani note based on the uploaded image."""

    input\_image = cv2.imread(input\_image\_path, 0)

    if input\_image is None:

        print("Error: Unable to load the input image.")

        return None

    \_, input\_descriptors = extract\_features(input\_image)

    if input\_descriptors is None or len(input\_descriptors) == 0:

        print("Error: No features detected in the input image.")

        return None

    desList = findDes(images)

    best\_match\_index = match\_image(input\_descriptors, desList)

    if best\_match\_index is not None:

        return ClassNames[best\_match\_index]  # Return the matched note name

    else:

        return None

@app.route('/', methods=['GET', 'POST'])

def index():

    """Render the home page with the flag of Pakistan."""

    if request.method == 'POST':

        if 'file' not in request.files:

            return redirect(request.url)

        file = request.files['file']

        if file.filename == '':

            return redirect(request.url)

        if file and allowed\_file(file.filename):

            filename = secure\_filename(file.filename)

            file\_path = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

            file.save(file\_path)

            # Process the image and detect currency

            note\_name = detect\_currency(file\_path)

            if note\_name:

                message = "Detected Currency : Pakistan"

            else:

                message = " "

            return render\_template('result.html', message=message, image\_path=filename)

        else:

            return redirect(request.url)  # Handle case where file extension is not allowed

    # Handle GET request

    return render\_template('index.html')  # Make sure you have this template file

@app.route('/uploads/<filename>')

def uploaded\_file(filename):

    """Serve uploaded files."""

    return send\_from\_directory(app.config['UPLOAD\_FOLDER'], filename)

def allowed\_file(filename):

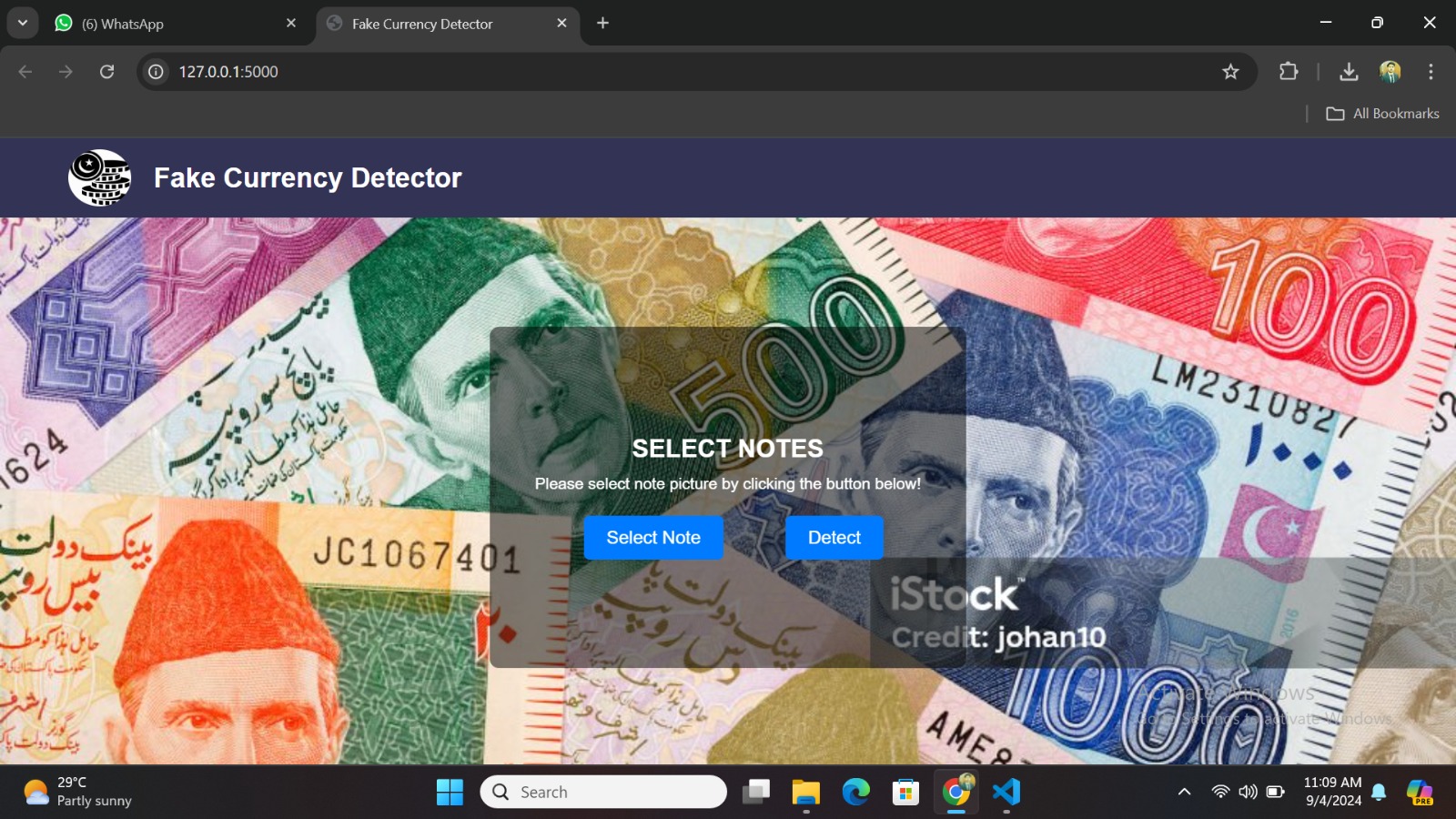
    """Check if the uploaded file has an allowed extension."""

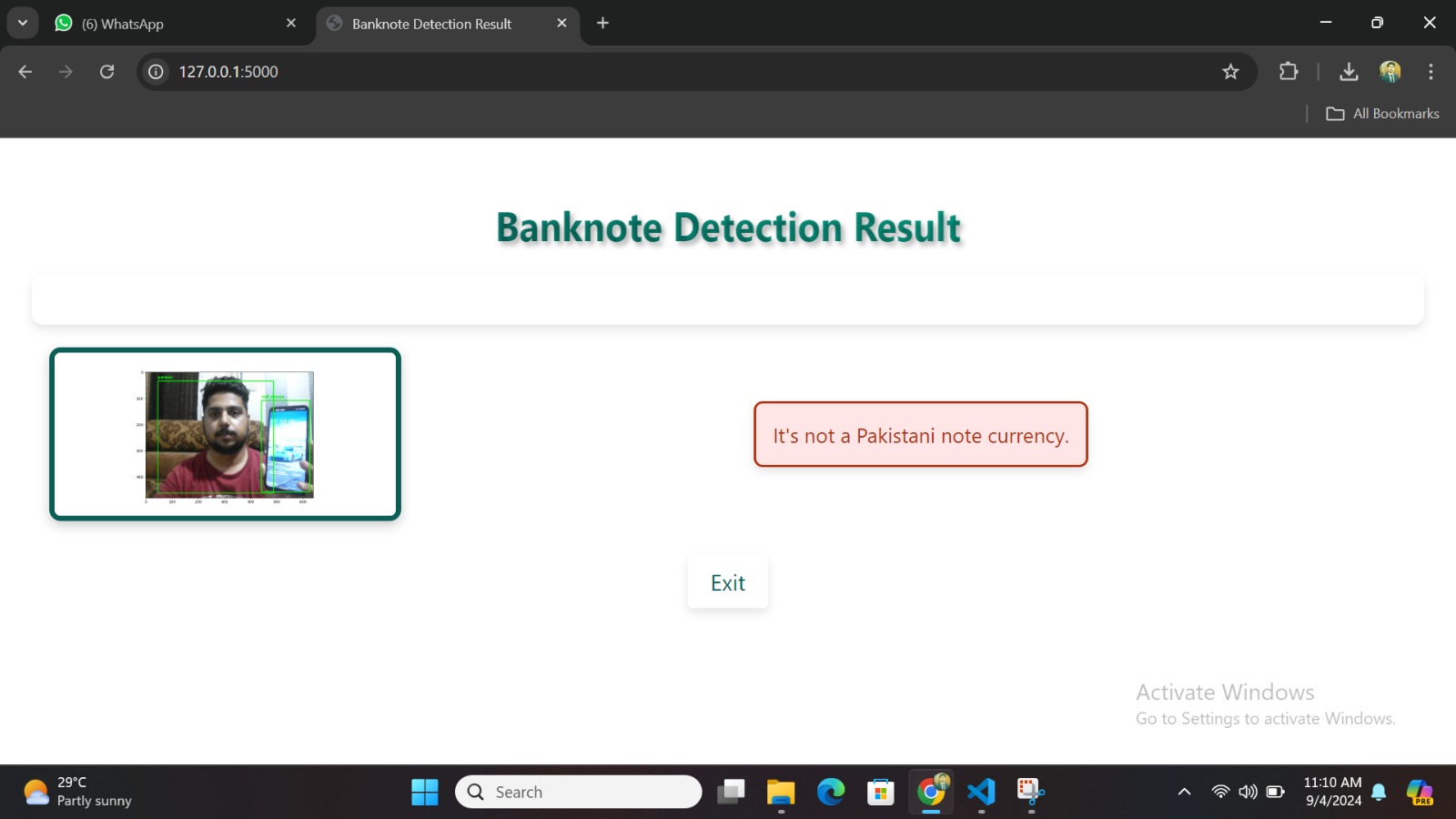
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED\_EXTENSIONS

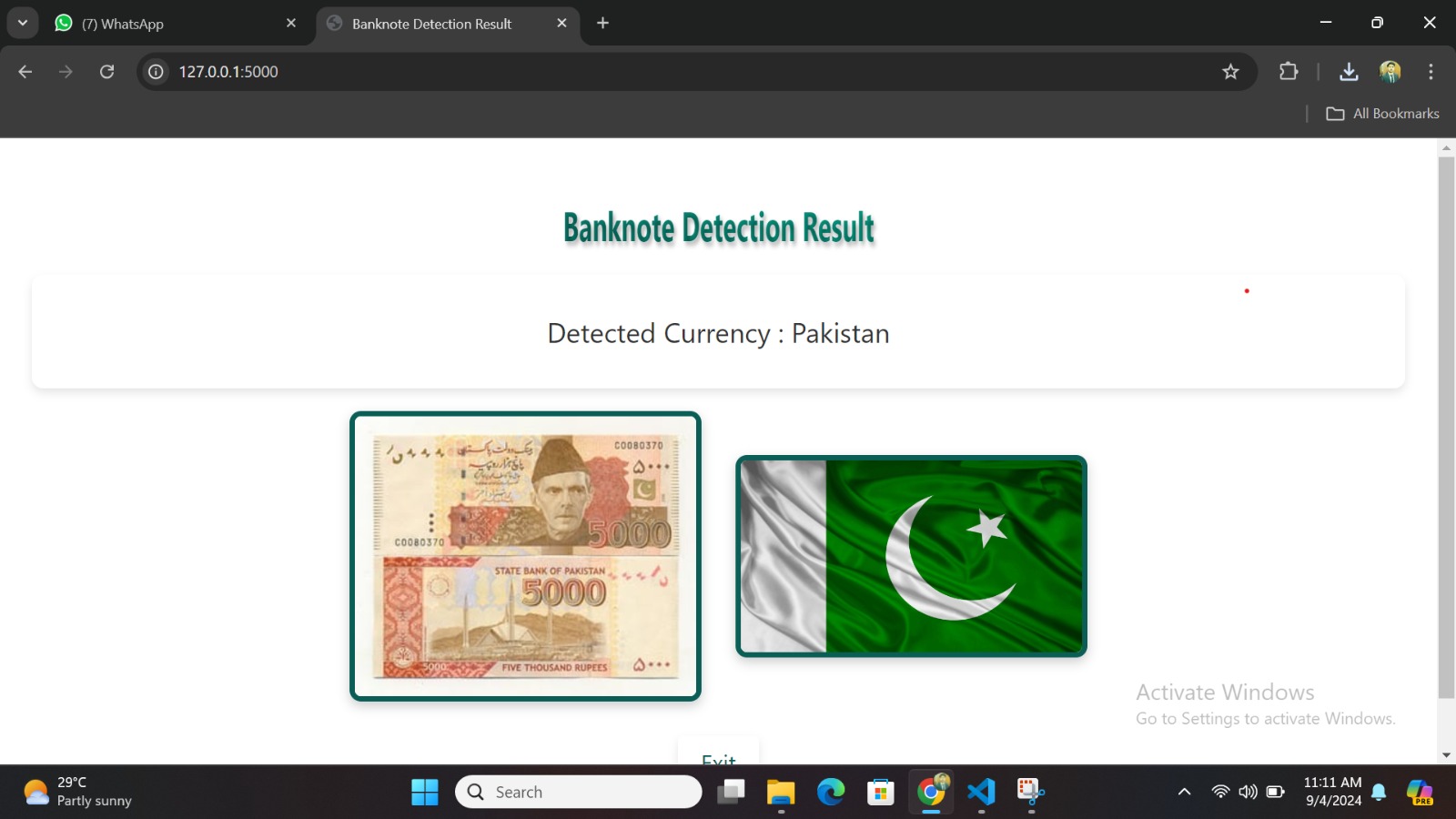
if \_\_name\_\_ == '\_\_main\_\_':

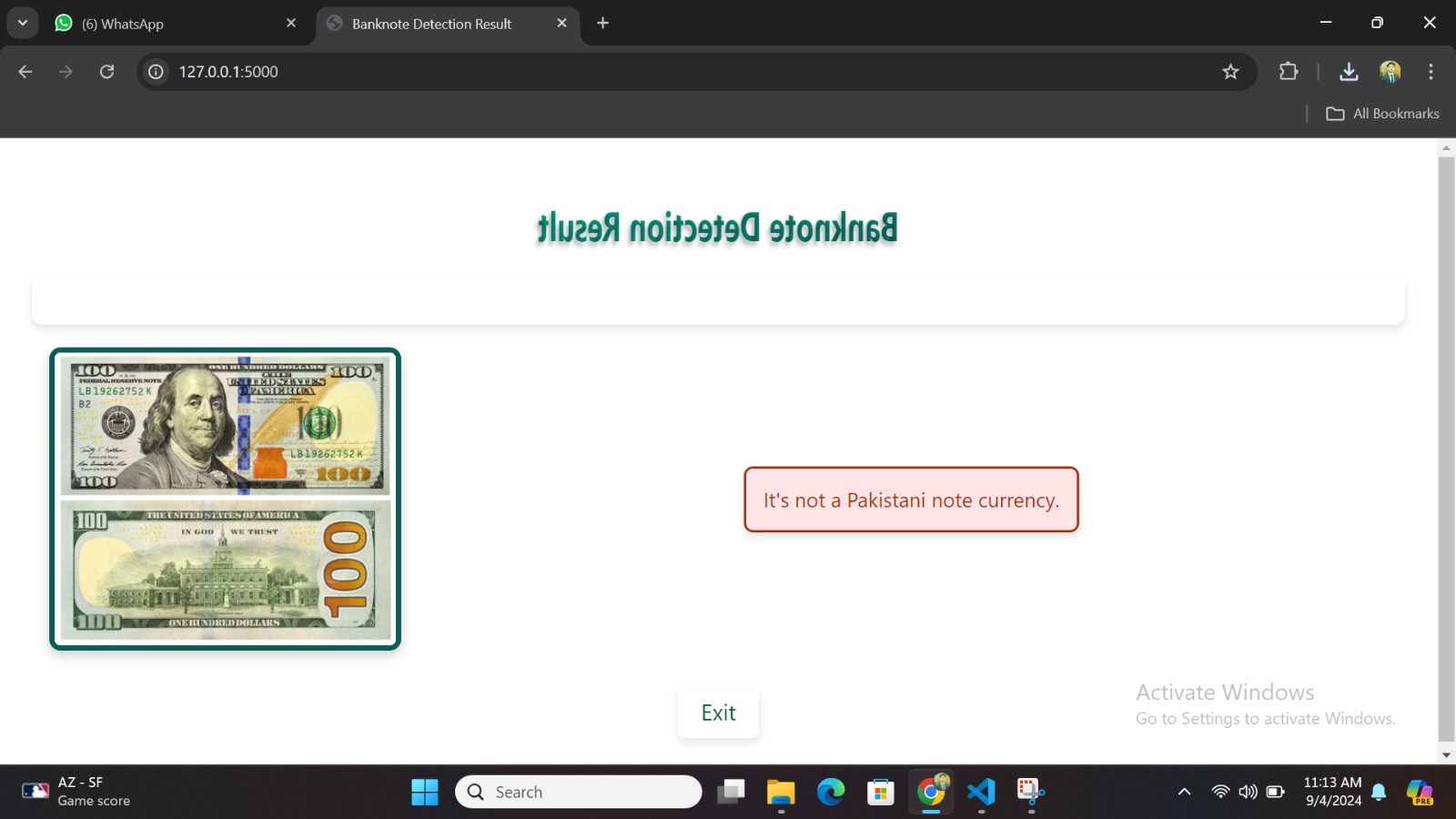
    app.run(debug=True)

**Output Detector**

****

****

****

****

**Working of Currency Detector**

The app.py script is a Flask application that detects currency by matching images against a dataset. Here’s an overview of how it works:

**1. Flask Setup and Configuration**

The application is built using the Flask framework and configured to handle image uploads.

Uploads are stored in the uploads folder, and allowed file types are png, jpg, jpeg, gif, and jfif.

**2. Dataset Loading**

The ORB (Oriented FAST and Rotated BRIEF) feature detector is initialized to detect features in images.

The script loads images from the ImagesQuery directory, which contains reference images of different currencies.

Each image is associated with a class name derived from the file names.

**3. Feature Extraction**

extract\_features(image): This function uses ORB to extract keypoints and descriptors from an image.

findDes(images): It computes descriptors for all images in the dataset.

**4. Image Matching**

The matching process uses a Brute-Force Matcher with the Hamming distance to compare the descriptors of the input image with the dataset descriptors.

The match\_image() function identifies the best match by finding the dataset image with the highest number of good matches based on a threshold.

**5. Detection Logic**

When a user uploads an image, the script extracts its features and compares them to the dataset using the matching logic.

The best match indicates the detected currency type.