|  |
| --- |
| ECONOMICAL MOBIWISE INSIGHT SYSTEM (EMWIS) |
| Implementation and Coding - Report |

Coding Standards and Guidelines

### General Rules

* Write simple, clean, and easy-to-understand code.
* Avoid repeating the same code (use functions where needed).
* Add short comments if a piece of code is not obvious.
* Always handle errors properly — don't crash the app.

### Python (Flask Backend)

* **Indentation:** Always use 4 spaces (no tabs).
* **Naming:**
  + Variables and functions → snake\_case (e.g., process\_order, user\_name)
  + Classes → PascalCase (e.g., OrderManager)
  + Constants → UPPER\_CASE (e.g., MAX\_LOGIN\_ATTEMPTS)
* **File Organization:** Group similar files (routes, models, utils) separately.
* **Comments:** Write short descriptions for important functions and tricky parts.
* **Error Handling:** Use try-except blocks especially for database and API calls.
* **Security:**
  + Validate all user inputs.
  + Keep passwords, API keys, and DB credentials secret (use environment variables).

### Database (Oracle DB via PyODBC)

* Always use **parameterized queries** (no direct string building).
* Prefer writing query columns clearly instead of SELECT \*.
* Table and column names should be **UPPERCASE** (e.g., USER\_ACCOUNT, MOBILE\_NAME).
* Always close database connections properly (use finally block).

### Frontend (HTML, CSS, JavaScript)

* **HTML:**
  + Use proper tags like <header>, <footer>, <section>, etc.
  + IDs and classes should be kebab-case (e.g., mobile-card, order-history).
* **CSS:**
  + Use external CSS files, no inline styles.
  + Follow consistent colors and theme (light blue, dark blue branding).
  + Make sure pages look good on mobile and desktop.
* **JavaScript:**
  + Use camelCase for variables and functions (e.g., loadCartItems()).
  + Handle API/AJAX responses properly (success, error, loading states).

### File Naming and Structure

* **Python files:** snake\_case.py (e.g., order\_routes.py)
* **HTML templates:** snake\_case.html (e.g., track\_order.html)
* **Static files:**
  + CSS → /static/css/
  + JS → /static/js/
  + Images → /static/images/

### Commenting

* Comment briefly what a function does.
* Add notes if a part of the code is tricky.
* No need to comment obvious things.

### Git (Version Control)

* Make regular commits with clear messages.
* Example commit message: Fixed cart update bug, Added forgot password feature.
* Use branches for new features if needed (feature/cart-improvements).

## Code Section

### Environment Setup Code

**requirements.txt**

Flask

pyodbc

requests

pywhatkit

python-jose

apscheduler

flask-caching

flask-compress

pygame

**Python Environment Setup Example**

pip install -r requirements.txt

### Folder Structure Overview

mobiwiseinsight/

│

├── app.py # Main Flask app

├── programs/ # Core Python modules (AI, chatbot, compare, etc.)

├── static/ # Static files (CSS, JS, images)

├── templates/ # HTML templates

├── Admin\_Authenticity\_Form/ # Admin auth module

├── User\_Authenticity\_Form/ # User auth module

├── Mobile\_Form/ # Mobile management

├── Discount\_Form/ # Discount management

├── Order\_Form/ # Order management

├── scripts/ # SQL scripts

└── requirements.txt # Python dependencies

### Flask App Initialization Code

**app.py**

from flask import Flask

from flask\_compress import Compress

from flask\_caching import Cache

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

Compress(app)

cache = Cache(app, config={'CACHE\_TYPE': 'simple'})

### Database Connection Code

**app.py**

import pyodbc

dsn = 'ODBC\_2007'

user = 'arjun\_christoph'

password = '2004'

conn = pyodbc.connect(f"DSN={dsn};UID={user};PWD={password}", timeout=10)

### User Authentication Code

**User\_Authenticity\_Form/app.py**

from flask import request, redirect

import jwt

SECRET\_KEY = "your\_super\_secret\_key"

def decode\_token(token):

try:

return jwt.decode(token, SECRET\_KEY, algorithms=["HS256"])

except jwt.ExpiredSignatureError:

return None

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

def login():

username = request.form['username']

password = request.form['password']

# Validate against DB

# If valid:

token = jwt.encode({'username': username}, SECRET\_KEY, algorithm="HS256")

resp = make\_response(redirect('/dashboard'))

resp.set\_cookie('jwt', token)

return resp

Signup and Forgot Password are similarly handled in their respective routes.

### Admin Console Functionalities

**Mobile\_Form/app.py**

@app.route('/fetch\_mobile\_buffer', methods=['POST'])

def fetch\_mobile\_buffer():

    data = request.get\_json()

    mobile\_id = data.get('mobile\_id')

    cursor = connection.cursor()

    try:

*# Check if the mobile ID exists in the buffer table*

        cursor.execute("SELECT \* FROM Mobiles\_Buffer WHERE MobileID = ?", (mobile\_id,))

        record = cursor.fetchone()

        if record:

*# Get column names from the cursor description*

            columns = [column[0] for column in cursor.description]

            response\_data = dict(zip(columns, record))

*# Debugging: Log the fetched data*

            print(f"Record fetched: {record}")

            print(f"Columns fetched: {columns}")

*# Safely convert fields*

            response\_data['Price'] = float(response\_data.get('Price', 0))  *# Default to 0 if Price is missing*

            response\_data['UserRating'] = float(response\_data.get('UserRating', 0))  *# Convert rating to float*

            return jsonify(response\_data), 200

        else:

            return jsonify({"error": f"No record found in the buffer table for Mobile ID: {mobile\_id}"}), 404

    except pyodbc.Error as e:

        return jsonify({"error": f"Database error: {str(e)}"}), 500

    finally:

        cursor.close()

@app.route('/save\_mobile\_buffer', methods=['POST'])

def save\_mobile\_buffer():

*"""*

*Save a record into the Mobiles\_Buffer table.*

*If the MobileID already exists, return an error.*

*"""*

    data = request.get\_json()

*# Extract form fields*

    mobile\_id = data.get('mobile\_id')

    model = data.get('model')

    brand = data.get('brand')

    price = data.get('price')

    ram = data.get('ram')

    storage = data.get('storage')

    battery = data.get('battery')

    display = data.get('display')

    processor = data.get('processor')

    front\_camera = data.get('front\_camera')

    rear\_camera = data.get('rear\_camera')

    release\_date = data.get('release\_date')

    user\_rating = data.get('user\_rating')

    description = data.get('description')

    images = data.get('images')

    if not all([mobile\_id, model, brand, price, ram, storage, battery, display, processor, front\_camera, rear\_camera, release\_date, user\_rating, description, images]):

        return jsonify({"error": "All fields are required."}), 400

    cursor = connection.cursor()

    try:

*# Check if the MobileID already exists in the buffer table*

        cursor.execute("SELECT 1 FROM Mobiles\_Buffer WHERE MobileID = ?", (mobile\_id,))

        if cursor.fetchone():

            return jsonify({"error": f"Mobile ID {mobile\_id} already exists in the buffer table."}), 409

*# Insert the new record into the buffer table*

        cursor.execute("""

            INSERT INTO Mobiles\_Buffer (MobileID, Model, Brand, Price, RAM, Storage, Battery, Display, Processor, FrontCamera, RearCamera, ReleaseDate, UserRating, Description, Images)

            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, TO\_DATE(?, 'YYYY-MM-DD'), ?, ?, ?)

        """, (mobile\_id, model, brand, price, ram, storage, battery, display, processor, front\_camera, rear\_camera, release\_date, user\_rating, description, images))

        connection.commit()

        return jsonify({"success": f"Mobile ID {mobile\_id} successfully saved to the buffer table."}), 200

    except pyodbc.Error as e:

        connection.rollback()

        return jsonify({"error": f"Database error: {str(e)}"}), 500

    finally:

        cursor.close()

### Mobile Comparison Module Code

**programs/compare.py**

import google.generativeai as genai

import requests

import json

from decimal import Decimal

API\_KEY = "AIzaSyBjc32TnXnaMHOkBTOLFMkCCAP-EPfKuDk"  *# Replace with your real key*

url = f"https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key={API\_KEY}"

def convert\_decimal\_to\_float(data):

*""" Recursively convert all Decimal values to float in a dictionary or list """*

    if isinstance(data, list):

        return [convert\_decimal\_to\_float(item) for item in data]

    elif isinstance(data, dict):

        return {key: convert\_decimal\_to\_float(value) for key, value in data.items()}

    elif isinstance(data, Decimal):  *# Convert Decimal to float*

        return float(data)

    return data

def compare\_and\_recommend(mobiles\_list, preferences):

*"""*

*Uses OpenAI GPT-4 to compare mobile specifications and recommend the best mobile.*

*"""*

    mobiles\_list = convert\_decimal\_to\_float(mobiles\_list)

    prompt = f"""

    You are an AI mobile comparison expert. The user has selected the following mobiles for comparison:

    {json.dumps(mobiles\_list, indent=4)}

    The user preferences are: {preferences}

    Based on real-time mobile market trends, analyze and compare the mobiles in these categories:

    - Price (Lower is better)

    - RAM (Higher is better)

    - Storage (Higher is better)

    - Battery Capacity (Higher is better)

    - Display Quality (AMOLED > OLED > LCD > TFT)

    - Processor Performance (Ranked by real-world benchmarks)

    - Front Camera (Higher MP is better)

    - Rear Camera (Count lenses first, then compare MP)

    - User Rating (Higher rating is better)

    \*\*Tasks to Perform:\*\*

    1️⃣ Provide a detailed \*\*comparison table\*\* with the best model for each category.

    2️⃣ Recommend the \*\*best mobile overall\*\* based on user preferences.

    Provide a JSON response in this format:

    {{

        "comparisons": [

            {{

                "feature": "Feature Name",

                "best\_model": "Model Name",

                "best\_value": "Best Value"

            }},

            ...

        ],

        "best\_suggestion": {{

            "model": "Best Mobile Model Name",

            "brand": "Brand Name",

            "price": "₹ Best Price",

            "reason": "Explain why this mobile is best based on user preferences."

        }}

    }}

    """

    headers = {

        "Content-Type": "application/json"

    }

    data = {

        "contents": [

            {

                "parts": [

                    {"text": prompt}

                ]

            }

        ]

    }

    try:

        response = requests.post(url, headers=headers, data=json.dumps(data))

        if response.status\_code == 200:

            result = response.json()

            return result['candidates'][0]['content']['parts'][0]['text'].strip()

        else:

            print("Gemini API Error:", response.status\_code, response.text)

            return "Could not get AI recommendation."

    except Exception as e:

        print(f"Error in AI Comparison & Recommendation: {e}")

        return {

            "comparisons": [],

            "best\_suggestion": {

                "model": "N/A",

                "brand": "N/A",

                "price": "N/A",

                "reason": "AI analysis failed. Try again later."

            }

        }

### Cart Management Code

**app.py**

@app.route('/add-to-cart', methods=['POST'])

def add\_to\_cart():

    try:

        data = request.get\_json()

        mobile\_id = data.get("mobile\_id")

        if not user:

            return jsonify({"success": False, "message": "Login/Sign Up required"})

*# Get user ID from username*

        cursor = conn.cursor()

        cursor.execute("SELECT UserID FROM Users WHERE Username = ?", (user["username"],))

        result = cursor.fetchone()

        if not result:

            return jsonify({"success": False, "message": "User not found"})

        user\_id = int(result[0])

*# Insert into Cart table using sequence*

        cursor.execute("INSERT INTO Cart (CartID, UserID, MobileID) VALUES (cart\_seq.NEXTVAL, ?, ?)", (user\_id, mobile\_id))

        conn.commit()

        cursor.close()

        return jsonify({"success": True})

    except Exception as e:

        print("Error in add\_to\_cart:", e)

        return jsonify({"success": False, "message": "Error adding to cart"})

@app.route('/remove-from-cart', methods=['POST'])

def remove\_from\_cart():

    try:

        data = request.get\_json()

        mobile\_id = data.get("mobile\_id")

*# Get user ID from the logged-in user*

        cursor = conn.cursor()

        cursor.execute("SELECT UserID FROM Users WHERE Username = ?", (user["username"],))

        result = cursor.fetchone()

        user\_id = result[0]

*# Remove item from the cart*

        cursor.execute("DELETE FROM Cart WHERE UserID = ? AND MobileID = ?", (user\_id, mobile\_id))

        conn.commit()

        return jsonify({"success": True, "message": "Item removed from cart!"})

    except pyodbc.Error as e:

        return jsonify({"success": False, "message": "Error removing from cart"}), 500

    finally:

        cursor.close()

### Checkout and Order Placement Code

**app.py**

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

def checkout():

    try:

        data = request.get\_json()

        address = data.get("address", {})

        payment\_method = data.get("payment\_method", "Cash on Delivery")

        order\_status = data.get("order\_status", "Active")

        delivery\_stage = data.get("delivery\_stage", "Dispatched")

        mobile\_ids = data.get("mobile\_ids", [])

        if not user:

            return jsonify({"success": False, "message": "Login required"}), 401

        cursor = conn.cursor()

*# Get user ID*

        cursor.execute("SELECT UserID FROM Users WHERE Username = ?", (user["username"],))

        user\_id = cursor.fetchone()[0]

*# Insert new address*

        cursor.execute("""

            INSERT INTO UserAddresses (AddressID, UserID, DoorStreet, Locality, City, Country, Pincode)

            VALUES (address\_seq.NEXTVAL, ?, ?, ?, ?, ?, ?)

        """, (user\_id, address["door\_street"], address["locality"], address["city"], address["country"], address["pincode"]))

*# Get AddressID from CURRVAL*

        cursor.execute("SELECT address\_seq.CURRVAL FROM dual")

        address\_id = cursor.fetchone()[0]

*# Expected delivery date (within 7 days)*

        expected\_delivery = datetime.now().date() + timedelta(days=random.randint(0, 6))

*# Insert order for each mobile*

        for mobile\_id in mobile\_ids:

            cursor.execute("""

                INSERT INTO Orders (OrderID, UserID, AddressID, MobileID, PaymentMethod, ExpectedDelivery, OrderDate)

                VALUES (order\_seq.NEXTVAL, ?, ?, ?, ?, TO\_DATE(?, 'YYYY-MM-DD'), SYSDATE)

            """, (user\_id, address\_id, mobile\_id, payment\_method, expected\_delivery.strftime('%Y-%m-%d')))

        conn.commit()

*# Fetch phone number and username*

        cursor.execute("SELECT Phone, Username FROM Users WHERE UserID = ?", (user\_id,))

        result = cursor.fetchone()

        phone\_number = result[0]

        username = result[1]

*# Calculate grand total*

        cursor.execute("SELECT Price FROM Mobiles WHERE MobileID IN ({})".format(','.join(['?']\*len(mobile\_ids))), mobile\_ids)

        prices = cursor.fetchall()

        grand\_total = sum([float(price[0]) for price in prices])

        grand\_total = indian\_format(grand\_total)

*# Send email confirmation*

        subject = "Order Confirmation - MobiWise Insight"

        body = f"""

        Hello {username},

        ✅ Your order has been placed successfully on MobiWise Insight.

        📱 Number of Mobiles: {len(mobile\_ids)}

        💰 Grand Total: ₹{grand\_total}

        📦 Expected Delivery Date: {expected\_delivery.strftime('%d-%m-%Y')}

        Thank you for shopping with us!

        """

*# Fetch user email*

        cursor.execute("SELECT Email FROM Users WHERE UserID = ?", (user\_id,))

        email = cursor.fetchone()[0]

        if send\_email(email, subject, body):

            print(f"✅ Order confirmation email sent to {email}")

        else:

            print(f"❌ Failed to send email to {email}")

*# Send WhatsApp order confirmation*

        send\_order\_confirmation\_whatsapp(phone\_number, username, expected\_delivery, len(mobile\_ids), grand\_total)

        cursor.close()

        return jsonify({"success": True, "message": "Order placed successfully!"})

    except Exception as e:

        print("Checkout error:", e)

        return jsonify({"success": False, "message": "An error occurred during checkout"}), 500

### Order Tracking and Order History Code

**app.py**

@app.route('/order-details')

def order\_details():

    token = request.cookies.get("jwt")

    user\_data = decode\_token(token) if token else None

    if not user\_data:

        return render\_template("track\_order.html", user=None, orders=[])

    cursor = conn.cursor()

    try:

        cursor.execute("SELECT UserID FROM Users WHERE Username = ?", (user\_data["username"],))

        user\_id = cursor.fetchone()[0]

        cursor.execute("""

            SELECT o.ExpectedDelivery, o.DeliveryStage, m.Model, m.Brand, m.Images, o.OrderID, m.Price

            FROM Orders o

            JOIN Mobiles m ON o.MobileID = m.MobileID

            WHERE o.UserID = ? AND o.OrderStatus = 'Active' AND o.DeliveryStage != 'Delivered'

        """, (user\_id,))

        rows = cursor.fetchall()

        orders = [{

            "ExpectedDelivery": row[0].strftime("%d-%m-%Y"),

            "DeliveryStage": row[1],

            "Model": row[2],

            "Brand": row[3],

            "Image": row[4],

            "OrderID": row[5],

            "Price": indian\_format(row[6])

        } for row in rows]

        return render\_template("track\_order.html", user=user\_data, orders=orders)

    except Exception as e:

        print("Track Order Error:", e)

        return render\_template("track\_order.html", user=user\_data, orders=[])

    finally:

        cursor.close()

@app.route('/order-history')

def order\_history():

    token = request.cookies.get("jwt")

    user\_data = decode\_token(token) if token else None

    if not user\_data:

        return redirect("/")

    cursor = conn.cursor()

    try:

        cursor.execute("SELECT UserID FROM Users WHERE Username = ?", (user\_data["username"],))

        user\_id = cursor.fetchone()[0]

        cursor.execute("""

            SELECT m.Model, m.Brand, TO\_CHAR(m.Price, '99,99,999'), TO\_CHAR(o.OrderDate, 'DD-MM-YYYY'), a.DoorStreet, a.Locality, a.City, a.Country, a.Pincode, m.Images, TO\_CHAR(o.ExpectedDelivery, 'DD-MM-YYYY'), o.OrderID

            FROM Orders o

            JOIN Mobiles m ON o.MobileID = m.MobileID

            JOIN UserAddresses a ON o.AddressID = a.AddressID

            WHERE o.UserID = ? AND o.DeliveryStage = 'Delivered'

            ORDER BY o.OrderDate DESC

        """, (user\_id,))

        rows = cursor.fetchall()

        history = [{

            "Model": row[0],

            "Brand": row[1],

            "Price": row[2],

            "OrderDate": row[3],

            "DoorStreet": row[4],

            "Locality": row[5],

            "City": row[6],

            "Country": row[7],

            "Pincode": row[8],

            "Image": row[9],

            "DeliveryDate": row[10],

            "OrderID": row[11],

        } for row in rows]

        return render\_template("order\_history.html", user=user\_data, history=history)

    except Exception as e:

        print("Order History Error:", e)

        return render\_template("order\_history.html", user=user\_data, history=[])

    finally:

        cursor.close()

### Chatbot Handling Code

**programs/chatbot.py**

import requests

import json

*# Gemini API Key & Endpoint*

API\_KEY = "AIzaSyBjc32TnXnaMHOkBTOLFMkCCAP-EPfKuDk"  *# Replace with a real secure key*

url = f"https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key={API\_KEY}"

*# Main chatbot interaction function*

def get\_chatbot\_reply(user\_input):

*"""*

*Send a message to Gemini and receive an AI-generated reply.*

*Args:*

*user\_input (str): User's message input.*

*Returns:*

*str: Gemini's cleaned response or error fallback.*

*"""*

    prompt = f"""

    You are a friendly, helpful chatbot assistant for a mobile insight platform.

    Answer user queries in a clear and conversational tone.

    Example queries include:

    - Recommend a good mobile under ₹30,000 with good battery life.

    - What's the difference between AMOLED and OLED displays?

    - Suggest phones with best camera in mid-range.

    User's message:

    {user\_input}

    Please answer in a helpful, informative manner. Keep your response short and user-friendly.

    """

    headers = {

        "Content-Type": "application/json"

    }

    payload = {

        "contents": [

            {

                "parts": [

                    {"text": prompt}

                ]

            }

        ]

    }

    try:

        response = requests.post(url, headers=headers, data=json.dumps(payload))

        if response.status\_code == 200:

            result = response.json()

            result = result['candidates'][0]['content']['parts'][0]['text'].strip()

            return result

        else:

            print("❌ Gemini API Error:", response.status\_code, response.text)

            return "Sorry, I couldn't process your message at the moment."

    except Exception as e:

        print(f"❌ Error during Gemini chat request: {e}")

        return "Oops! Something went wrong. Please try again later."

### Interactive Game Code

**app.py**

from flask import send\_from\_directory

@app.route('/dino-game/<path:filename>')

def dino\_game\_files(filename):

    return send\_from\_directory('Dino\_Game', filename)

### Cloud Integration Code

**app.py**

def fetch\_mobile\_news():

    sources = [

        "https://newsapi.org/v2/everything?q=smartphone%20OR%20%22mobile%20phone%22%20OR%20%22mobile%20technology%22&language=en&sortBy=publishedAt&apiKey=92aea20e041045ad8ed5fcdd757ba098",

        "https://gnews.io/api/v4/search?q=smartphone%20OR%20%22mobile%20device%22&lang=en&sortby=publishedAt&token=c3ee7fb0fa4474800c9a3f5c60175cf1",

        "https://api.currentsapi.services/v1/latest-news?keywords=smartphone%20technology&language=en&apiKey=qgXQBOO\_NW3Ogf41LkC9Qop5UWpoLEa2IBKWh2kEOu\_ZhZcw",

        "https://real-time-amazon-data.p.rapidapi.com/product-details",

        "https://api.thenewsapi.com/v1/news/all?api\_token=RYDrBJ0CiYQKIpAK5l3FWlh55Q78C8w7cyQk3csH&search=smartphones&language=en&limit=10",

*#"https://contextualwebsearch-websearch-v1.p.rapidapi.com/api/search/NewsSearchAPI?q=smartphone&pageNumber=1&pageSize=10&autoCorrect=true",*

        "http://api.mediastack.com/v1/news?access\_key=2bd1f1e962b206e80c2038b1f88c5d10&keywords=smartphone&languages=en",

*#"https://content.guardianapis.com/search?q=smartphone&api-key=b728e35e-0f2c-4e26-b32a-851fe9f04604"*

    ]

    headers = {

        "X-RapidAPI-Host": "real-time-amazon-data.p.rapidapi.com",

        "X-RapidAPI-Key": "13c3a25697msh13c75961abd1277p15bf55jsnf054c694b1c7"

    }

    querystring = {"asin":"B07ZPKBL9V","country":"US"}

    news\_articles = []

    for url in sources:

        try:

            response = requests.get(url, headers=headers if "rapidapi" in url else None, params=querystring if "rapidapi" in url else None)

            if response.status\_code == 200:

                    data = response.json()

                    articles = data.get("articles", data.get("news", []))

                    for article in articles:

                        if article.get("urlToImage") or article.get("image") or article.get("image\_url"):  *# Ensure only articles with images*

                            news\_articles.append({

                                "title": article.get("title", "No Title"),

                                "description": article.get("description", "No description available")[:150] + "...",

                                "image": article.get("urlToImage") if article.get("urlToImage") else (article.get("image") if article.get("image") else article.get("image\_url")),

                                "url": article.get("url", "#"),

                                "publishedAt": article.get("publishedAt", "Unknown Date")[:10]

                            })

        except Exception as e:

            print(f"Error fetching news from {url}: {e}")  *# Debugging if an API fails*

    if not news\_articles:

        return [{"title": "No news available", "description": "Try again later.", "image": "/static/images/no-news.png", "url": "#", "publishedAt": "N/A"}]

    random.shuffle(news\_articles)  *# Shuffle news to prevent repetition*

    return news\_articles[:21]  *# Return latest 20 articles*

### Error Handling Code Samples

**app.py**

try:

# DB or API operation

except Exception as e:

print("Error:", e)

return "An error occurred", 500

### Security and Validation Snippets

**app.py**

from functools import wraps

# Decode token

def decode\_token(token):

try:

return jwt.decode(token, SECRET\_KEY, algorithms=["HS256"])

except jwt.ExpiredSignatureError:

return None

except jwt.InvalidTokenError:

return None

# Auth check decorator

def login\_required(f):

@wraps(f)

def decorated\_function(\*args, \*\*kwargs):

token = request.cookies.get("jwt")

if not token:

return redirect("/") # Or your login page

user = decode\_token(token)

if not user:

return redirect("/")

return f(\*args, \*\*kwargs, user=user)

return decorated\_function

### AI Discount Advertisement

**programs/ai\_ad.py**

import requests

import time

import os

*# Hugging Face API config*

API\_URL = "https://api-inference.huggingface.co/models/stabilityai/stable-diffusion-3.5-large"

HEADERS = {

    "Authorization": "Bearer hf\_YXFhEbEixpdjsdVgUJTJbROXctLhgpjmtY",  *# Replace with your actual token*

    "Content-Type": "application/json"

}

*# Retry-enabled query function*

def query\_huggingface(payload):

    response = requests.post(API\_URL, headers=HEADERS, json=payload)

    if response.status\_code == 503:

        print("⚠️ Model is loading, retrying in 10 seconds...")

        time.sleep(10)

        return query\_huggingface(payload)

    if not response.ok:

        print(f"❌ Error {response.status\_code}: {response.text}")

        return None

    return response.content

*# Main function to process all discount entries*

def generate\_all\_discount\_ads(discount\_data):

*"""*

*discount\_data: List of tuples (MobileID, Model, ImageURL, OfferName, Discount%, Price)*

*"""*

    discount\_img\_path = 'mobiwiseinsight/static/discount\_images/'

    os.makedirs(discount\_img\_path, exist\_ok=True)

    rendered\_data = []

    for row in discount\_data:

        mobile\_id, model, img\_url, offer\_name, discount\_percent, price = row

        filename = f"{mobile\_id}.jpg"

        save\_path = os.path.join(discount\_img\_path, filename)

*# Prompt: generate a visually compelling ad*

        prompt = generate\_fancy\_prompt(model, offer\_name, discount\_percent, price)

*# Generate image*

        image\_bytes = query\_huggingface({"inputs": prompt})

        if image\_bytes:

            with open(save\_path, "wb") as f:

                f.write(image\_bytes)

            print(f"✅ Ad image saved for {model} at {save\_path}")

        else:

            print(f"❌ Failed to generate image for {model}")

            continue

*# Append data for rendering*

        rendered\_data.append({

            "id": mobile\_id,

            "model": model,

            "discount": discount\_percent,

            "price": price,

            "offer": offer\_name,

            "image\_path": f"/static/discount\_images/{filename}"

        })

    return rendered\_data

*# Prompt generator for Hugging Face*

def generate\_fancy\_prompt(model, offer\_name, discount\_percent, price):

    return (

        f"""A luxurious, futuristic mobile discount advertisement featuring the {model} smartphone.

The scene is set in an ultra-modern tech showroom with sleek, reflective surfaces and dynamic lighting. Glowing, animated digital banners hover above the phone display, showcasing the latest technology.

The smartphone is showcased on a rotating platform, emphasizing its sleek design, smooth edges, and vibrant display.

Prominently display the text: '{offer\_name} and {discount\_percent}% OFF' in a bold, 3D-glowing neon font, with the discount percentage pulsing for attention.

The final price, '₹{price}', appears in a large, metallic gradient font with a shiny gold finish, positioned below the phone in a clean, professional layout.

Use a high-tech digital theme with bright accents like electric blue, neon purple, and glowing orange.

Incorporate motion blur effects for a futuristic, fast-paced feel.

Add dynamic reflections on the showroom floor and subtle animated sparks to suggest premium quality.

Ensure the design is ultra-sharp, highly detailed, and suitable for social media promotion, including platforms like Instagram and Facebook.

Maintain a professional and luxurious style while ensuring the smartphone remains the focal point of the image.

Use fancy lighting, vibrant contrast, and cinematic framing to create an eye-catching visual that highlights the discount and premium nature of the offer.""")

### WhatsApp, SMS, and Email Message Send

**programs/whatsapp\_msg.py**

import pywhatkit as kit

import time

from datetime import datetime, timedelta

def send\_whatsapp\_message(user\_phone, model, discount\_percentage, new\_price, mobile\_id):

    url = f"http://127.0.0.1:5001/mobile-details/{mobile\_id}"

*# Format the message*

    message = f"             📦 \*MobiWise Insight\* 📦             \n\n" \

              f"🚨 \*Exclusive Offer!\* 🚨\n\n" \

              f"Get your hands on the \*{model}\* 📱 at an unbeatable price!\n\n" \

              f"🎉 \*{discount\_percentage}% OFF!\* 🎉\n\n" \

              f"🔥 \*New Price\*: ₹{new\_price}\n\n" \

              f"🛒 \*Hurry, Limited Offer!\* Grab it now!\n\n" \

              f"Check the product below 👇\n" \

              f"{url}\n"  *# Include URL in the message text*

*# Send message immediately using pywhatkit without entering manually*

    kit.sendwhatmsg\_instantly(

        phone\_no=user\_phone,              *# User's phone number*

        message=message,                  *# Formatted message*

        wait\_time=15                       *# Wait 15 seconds before sending the message*

    )

    print(f"Message sent to {user\_phone} via WhatsApp Web!")

def send\_order\_confirmation\_whatsapp(phone\_number, username, expected\_delivery, total\_items, grand\_total):

    try:

        message = f"""Hello {username},

✅ Your order has been placed successfully at \*MobiWise Insight\*!

🛒 Total Items: {total\_items}

💰 Grand Total: ₹{grand\_total}

📦 Expected Delivery: {expected\_delivery.strftime('%d-%b-%Y')}

We’ll keep you updated about your delivery status.

Thank you for shopping with us!

- MobiWise Insight Team"""

*# Format phone number (India example: +91XXXXXXXXXX)*

        formatted\_phone = f"{phone\_number.strip()}"

*# Send message immediately using pywhatkit without entering manually*

        kit.sendwhatmsg\_instantly(

            phone\_no=formatted\_phone,              *# User's phone number*

            message=message,                  *# Formatted message*

            wait\_time=15                       *# Wait 15 seconds before sending the message*

        )

        print(f"✅ WhatsApp confirmation sent to {formatted\_phone}")

    except Exception as e:

        print(f"❌ WhatsApp message error: {e}")

**programs/email\_msg.py**

import smtplib

from email.mime.text import MIMEText

*# SMTP configuration*

SMTP\_SERVER = 'smtp.gmail.com'

SMTP\_PORT = 587

SENDER\_EMAIL = 'arjun.christopher877@ptuniv.edu.in'  *# Your email*

SENDER\_PASSWORD = 'xmnp ujhy lyfj zvxj'  *# App password*

def send\_email(recipient, subject, body):

    try:

        msg = MIMEText(body)

        msg['Subject'] = subject

        msg['From'] = SENDER\_EMAIL

        msg['To'] = recipient

        with smtplib.SMTP(SMTP\_SERVER, SMTP\_PORT) as server:

            server.starttls()

            server.login(SENDER\_EMAIL, SENDER\_PASSWORD)

            server.send\_message(msg)

    except Exception as e:

        print(f"SMTP Error: {str(e)}")

        return False

    return True

**User\_Authenticity\_Form/programs/sms.py**

from twilio.rest import Client

*# Twilio Credentials (Replace with your actual credentials)*

ACCOUNT\_SID = "ACf7dcc05cc77b2d55e5bca24f0a4d8401"

AUTH\_TOKEN = "57d217a69b1b28649bc230a61ebbf000"

TWILIO\_PHONE\_NUMBER = "+12677191942"

*# Initialize Twilio Client*

client = Client(ACCOUNT\_SID, AUTH\_TOKEN)

def send\_sms(otp, phone):

*# Message to be sent*

    message\_body = f"OTP: {otp}"

*# Send SMS*

    message = client.messages.create(

        body=message\_body,

        from\_=TWILIO\_PHONE\_NUMBER,

        to=phone

    )

*# Print message SID for tracking*

    print(f"Message sent successfully! Message SID: {message.sid}")

    return True

### Chat Forum

**templates/mobile\_details.html**

<div class="disqus-wrapper">

            <section class="discussion-section">

                <h2 class="discussion-title">User Discussions</h2>

                {% if user %}

                    <p class="comment-tip">

                        You are logged in as <strong>{{ user['username'] }}</strong> on MobiWise.

                        <br>

                        Please use the same name when commenting on Disqus to maintain identity.

                    </p>

                    <div id="disqus\_thread" class="discussion-box"></div>

                    <script>

*var* disqus\_config = function () {

                            this.page.url = "{{ request.url }}";

                            this.page.identifier = "mobile-{{ mobile.MobileID }}";

                        };

                        (function () {

*var* d = document, s = d.createElement('script');

                            s.src = 'https://mobiwise-insight.disqus.com/embed.js';

                            s.setAttribute('data-timestamp', +new Date());

                            (d.head || d.body).appendChild(s);

                        })();

                    </script>

                    <noscript>Please enable JavaScript to view the comments powered by Disqus.</noscript>

                {% else %}

                    <div class="comment-overlay">

                        <p>Login to participate in the discussion 🔒</p>

                        <a href="http://127.0.0.1:5002/">Click here to Login</a>

                    </div>

                {% endif %}

            </section>

            <script>

*var* disqus\_config = function () {

                    this.page.url = "{{ request.url }}";  *// Dynamic page URL*

                    this.page.identifier = "mobile-{{ mobile.MobileID }}";  *// Unique for each mobile*

                };

                (function () {

*var* d = document, s = d.createElement('script');

                    s.src = 'https://mobiwise-insight.disqus.com/embed.js';  *// Replace with your Disqus shortname*

                    s.setAttribute('data-timestamp', +new Date());

                    (d.head || d.body).appendChild(s);

                })();

            </script>

            <noscript>

                Please enable JavaScript to view the comments powered by Disqus.

            </noscript>

        </div>

### JWT (JSON Web Token) Usage

**app.py**

import jwt

def encode\_token(data):

return jwt.encode(data, SECRET\_KEY, algorithm="HS256")

def decode\_token(token):

try:

return jwt.decode(token, SECRET\_KEY, algorithms=["HS256"])

except:

return None

### Invoice Generation Code

**app.py**

@app.route("/invoice/<int:order\_id>")

def generate\_invoice(order\_id):

    token = request.cookies.get("jwt")

    user\_data = decode\_token(token) if token else None

    if not user\_data:

        return redirect("/")

    cursor = conn.cursor()

    try:

*# Get order + address + mobile info*

        cursor.execute("""

            SELECT m.Model, m.Brand, TO\_CHAR(m.Price, '99,99,999'), TO\_CHAR(o.OrderDate, 'DD-MM-YYYY'),

                   TO\_CHAR(o.ExpectedDelivery, 'DD-MM-YYYY'), a.DoorStreet, a.Locality, a.City,

                   a.Country, a.Pincode, m.Images, u.Username, u.Email, u.Phone

            FROM Orders o

            JOIN Mobiles m ON o.MobileID = m.MobileID

            JOIN UserAddresses a ON o.AddressID = a.AddressID

            JOIN Users u ON o.UserID = u.UserID

            WHERE o.OrderID = ?

        """, (order\_id,))

        row = cursor.fetchone()

        if not row:

            return "Invalid Order ID", 404

        invoice\_data = {

            "Model": row[0],

            "Brand": row[1],

            "Price": row[2],

            "OrderDate": row[3],

            "DeliveryDate": row[4],

            "Address": f"{row[5]}, {row[6]}, {row[7]} - {row[9]}, {row[8]}",

            "Image": row[10],

            "Username": row[11],

            "Email": row[12],

            "Phone": row[13],

            "OrderID": order\_id

        }

        return render\_template("invoice.html", invoice=invoice\_data)

    except Exception as e:

        return f"Error: {e}", 500

    finally:

        cursor.close()