

WASLA - Number Prediction - Notes

This explanation guides you to step by step procedure of WASLA Number Prediction Webapp, it's simple, concise and also has no fancy terms being thrown in it.

Flask Part

Imports and Initial Setup

1. Imports

```
python
Copy
from flask import Flask, render_template, request, redirect, url_for, session,
flash
from werkzeug.security import generate_password_hash, check_password_hash
import mysql.connector
from datetime import datetime
import os
from werkzeug.utils import secure_filename
import tensorflow as tf
import numpy as np
from PIL import Image
import io
import base64
```

- **Flask-related imports:** These bring in tools from Flask, a Python library for building web apps.
 - Flask: The main class to create the web app.
 - render_template: Loads HTML templates (web pages).
 - request: Handles data sent by users (like form inputs).
 - redirect: Sends users to another page.
 - url_for: Creates URLs for routes (like /login).
 - session: Stores user data (like who's logged in) during their visit.
 - flash: Shows temporary messages (like "Login successful!").
- **werkzeug.security:** Tools for secure password handling.
 - generate_password_hash: Turns a password into a secure, unreadable string.
 - check_password_hash: Checks if a password matches the stored hash.
- **mysql.connector:** Connects to a MySQL database to store user data, feedback, and predictions.
- **datetime:** Handles dates and times (e.g., when a user submits feedback).
- **os:** Not used in this code but typically for file/folder operations.
- **secure_filename:** Not used here but ensures safe file names when uploading files.

WASLA - Number Prediction - Notes

- **tensorflow (tf)**: A library for machine learning to load and use a pre-trained model.
- **numpy (np)**: Helps with numerical operations, like processing images for the model.
- **PIL (Image)**: Opens and processes images (e.g., resizing them for the model).
- **io**: Handles in-memory data streams (like converting images).
- **base64**: Converts images to/from a text format for web use.

2. App Setup

python

Copy

```
app = Flask(__name__)  
app.secret_key = 'your_secret_key'
```

- `app = Flask(__name__)`: Creates a Flask app. `__name__` tells Flask where to look for templates and files.
- `app.secret_key`: A secret code used to secure user sessions (like keeping someone logged in). Replace 'your_secret_key' with a unique, secret string in a real app.

3. Database Connection

python

Copy

```
conn = mysql.connector.connect(host='localhost', user='root', password='',  
database='wasla_project')  
cursor = conn.cursor()
```

- Connects to a MySQL database named `wasla_project` on the local computer (localhost).
- `user='root', password=""`: Uses the default MySQL username (root) with no password (not secure for real apps!).
- `cursor`: A tool to run SQL queries (like fetching or saving data).

4. Machine Learning Model

python

Copy

```
try:  
    model = tf.keras.models.load_model('models/sign_language_model.h5')  
except Exception as e:  
    print(f"Error loading model: {e}")  
    model = None
```

- Loads a pre-trained TensorFlow model (`sign_language_model.h5`) that recognizes sign language digits (0–9).
- `try/except`: If the model fails to load (e.g., file missing), it prints an error and sets `model = None` to avoid crashes.
- The model is used later to predict digits from images.

WASLA - Number Prediction - Notes

5. Digit Translations

python

Copy

```
DIGIT_TRANSLATIONS = {
    0: {'english': 'Zero', 'arabic': 'صفر'},
    1: {'english': 'One', 'arabic': 'واحد'},
    2: {'english': 'Two', 'arabic': 'اثنان'},
    3: {'english': 'Three', 'arabic': 'ثلاثة'},
    4: {'english': 'Four', 'arabic': 'أربعة'},
    5: {'english': 'Five', 'arabic': 'خمسة'},
    6: {'english': 'Six', 'arabic': 'سنة'},
    7: {'english': 'Seven', 'arabic': 'سبعة'},
    8: {'english': 'Eight', 'arabic': 'ثمانية'},
    9: {'english': 'Nine', 'arabic': 'تسعة'}
}
```

- A dictionary that maps digits (0–9) to their English and Arabic names.
- Example: For digit 0, it stores {'english': 'Zero', 'arabic': 'صفر'}.
- Used to show predictions in both languages.

6. Image Preprocessing Function

python

Copy

```
def preprocess_image(image):
    image = image.resize((64, 64))
    image_array = np.array(image)
    if image_array.shape[-1] != 3:
        image_array = np.stack((image_array,) * 3, axis=-1)
    image_array = image_array / 255.0
    image_array = np.expand_dims(image_array, axis=0)
    return image_array
```

- Prepares an image for the machine learning model.
- Steps:
 1. `image.resize((64, 64))`: Resizes the image to 64x64 pixels (the size the model expects).
 2. `np.array(image)`: Converts the image to a numerical array.
 3. Checks if the image has 3 color channels (RGB). If not (e.g., grayscale), it duplicates the single channel to make 3.
 4. `image_array / 255.0`: Scales pixel values from 0–255 to 0–1 (models work better this way).
 5. `np.expand_dims`: Adds a "batch" dimension (the model expects a batch of images, even if it's just one).

WASLA - Number Prediction - Notes

- Returns the processed image ready for prediction.

Routes (Web Pages and Actions)

Each route is a function that handles a specific URL (like /login) and defines what happens when a user visits it or submits data. I'll number and explain each one.

Route 1: Home Page

python

Copy

```
@app.route('/')
```

```
def home():
```

```
    return render_template('home.html')
```

- **URL:** / (the main page, e.g., http://localhost:5000/).
- **Purpose:** Shows the home page.
- **What it does:** Loads the home.html template (a web page file in the templates folder).
- **For beginners:** Think of this as the "welcome" page users see first.

Route 2: Login Page

python

Copy

```
@app.route('/login')
```

```
def login():
```

```
    return render_template('login.html')
```

- **URL:** /login.
- **Purpose:** Displays the login page where users enter their username and password.
- **What it does:** Loads login.html from the templates folder.
- **Note:** This only shows the form; the actual login check happens in the next route.

Route 3: Login Validation

python

Copy

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

```
def login_validation():
```

```
    username = request.form.get('username')
```

```
    password = request.form.get('password')
```

```
    cursor.execute("SELECT * FROM users WHERE username = %s", (username,))
```

```
    user = cursor.fetchone()
```

```
    if user:
```

```
        if check_password_hash(user[5], password):
```

```
            session['user_id'] = user[0]
```

WASLA - Number Prediction - Notes

```
        return redirect(url_for('camera'))
    else:
        flash('Incorrect password. Please try again.', 'danger')
        return redirect(url_for('login'))
else:
    flash('Username not found. Please register.', 'warning')
    return redirect(url_for('register'))
```

- **URL:** /login_validation.
- **Purpose:** Checks if the user's login details are correct.
- **Methods:** ['POST'] means it handles form submissions (when users click "Login").
- **What it does:**
 1. Gets username and password from the login form (request.form.get).
 2. Queries the database for a user with the given username.
 3. If a user is found (user):
 - Checks if the entered password matches the stored hashed password (user[5] is the password column).
 - If correct, stores the user's ID in session['user_id'] (keeps them logged in) and redirects to the /camera page.
 - If incorrect, shows an error message (flash) and reloads the login page.
 4. If no user is found, shows a message asking them to register and redirects to /register.
- **For beginners:** This is like a security guard checking your ID and password before letting you in.

Route 4: Register Page

python

Copy

```
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        first_name = request.form.get('first_name')
        last_name = request.form.get('last_name')
        username = request.form.get('username')
        phone_number = request.form.get('phone_number')
        password = request.form.get('password')
        confirm_password = request.form.get('confirm_password')
        if password != confirm_password:
            flash("Passwords do not match. Please try again.", "danger")
            return redirect(url_for('register'))
        cursor.execute("SELECT * FROM users WHERE username = %s", (username,))
        user = cursor.fetchone()
        if user:
```

WASLA - Number Prediction - Notes

```
flash("Username already exists. Please use a different username.",
"warning")
return redirect(url_for('register'))
hashed_password = generate_password_hash(password,
method='pbkdf2:sha256')
try:
    cursor.execute("""
        INSERT INTO users (first_name, last_name, username,
phone_number, password)
        VALUES (%s, %s, %s, %s, %s)
        """, (first_name, last_name, username, phone_number,
hashed_password))
    conn.commit()
    flash('Registration successful! You can now login.', 'success')
    return redirect(url_for('login'))
except Exception as e:
    flash('An error occurred during registration. Please try again.',
'danger')
    print(f"Error: {e}")
    return redirect(url_for('register'))
return render_template('register.html')
```

- **URL:** /register.
- **Purpose:** Lets users create a new account.
- **Methods:** Handles GET (shows the form) and POST (processes form submission).
- **What it does:**
 - For GET: Shows the register.html page with a form.
 - For POST (when the form is submitted):
 1. Gets form data: first_name, last_name, username, phone_number, password, confirm_password.
 2. Checks if password matches confirm_password. If not, shows an error and reloads the page.
 3. Checks if the username already exists in the database. If it does, shows an error.
 4. Hashes the password for security (generate_password_hash).
 5. Inserts the user's data into the users table.
 6. Saves changes (conn.commit()).
 7. If successful, shows a success message and redirects to /login.
 8. If an error occurs (e.g., database issue), shows an error and reloads the page.
- **For beginners:** This is like filling out a form to join a club and saving your details.

Route 5: Logout

WASLA - Number Prediction - Notes

python

Copy

```
@app.route('/logout')
def logout():
    session.pop('user_id', None)
    flash('You have been logged out.', 'success')
    return redirect(url_for('login'))
```

- **URL:** /logout.
- **Purpose:** Logs the user out.
- **What it does:**
 1. Removes user_id from the session (forgets who's logged in).
 2. Shows a success message.
 3. Redirects to the login page.
- **For beginners:** This is like signing out of an app or website.

Route 6: Forgot Password

python

Copy

```
@app.route('/forgot_password', methods=['GET', 'POST'])
def forgot_password():
    if request.method == 'POST':
        username = request.form.get('username')
        cursor.execute("SELECT * FROM users WHERE username = %s", (username,))
        user = cursor.fetchone()
        if user:
            return render_template('reset_password.html', username=username)
        else:
            flash("Username not found. Please register.", "warning")
            return redirect(url_for('register'))
    return render_template('forgot_password.html')
```

- **URL:** /forgot_password.
- **Purpose:** Lets users start the process to reset their password.
- **Methods:** Handles GET (shows form) and POST (checks username).
- **What it does:**
 - For GET: Shows forgot_password.html (a form to enter username).
 - For POST:
 1. Gets the username from the form.
 2. Checks if the username exists in the database.
 3. If found, shows reset_password.html with the username.
 4. If not found, shows an error and redirects to /register.
- **For beginners:** This is like the "I forgot my password" link on a login page.

WASLA - Number Prediction - Notes

Route 7: Reset Password

python

Copy

```
@app.route('/reset_password', methods=['POST'])
def reset_password():
    username = request.form.get('username')
    new_password = request.form.get('new_password')
    confirm_password = request.form.get('confirm_password')
    if new_password != confirm_password:
        flash("Passwords do not match. Try again.", "danger")
        return render_template('reset_password.html', username=username)
    hashed_password = generate_password_hash(new_password,
method='pbkdf2:sha256')
    try:
        cursor.execute("UPDATE users SET password = %s WHERE username = %s",
(hashed_password, username))
        conn.commit()
        flash("Password reset successfully! You can now login.", "success")
        return redirect(url_for('login'))
    except Exception as e:
        print("Error updating password:", e)
        flash("An error occurred. Please try again.", "danger")
        return render_template('reset_password.html', username=username)
```

- **URL:** /reset_password.
- **Purpose:** Updates the user's password.
- **Methods:** Handles POST (form submission).
- **What it does:**
 1. Gets username, new_password, and confirm_password from the form.
 2. Checks if the passwords match. If not, reloads the reset page with an error.
 3. Hashes the new password.
 4. Updates the password in the database for the given username.
 5. Saves changes (conn.commit()).
 6. If successful, shows a success message and redirects to /login.
 7. If an error occurs, reloads the reset page with an error.
- **For beginners:** This is like changing your password after verifying your account.

Route 8: Profile Page

python

Copy

```
@app.route('/profile', methods=['GET', 'POST'])
def profile():
    if 'user_id' not in session:
```


WASLA - Number Prediction - Notes

```
flash('Please log in to access your profile.', 'warning')
return redirect(url_for('login'))
user_id = session['user_id']
cursor.execute("SELECT first_name, last_name, username, phone_number FROM
users WHERE id = %s", (user_id,))
user = cursor.fetchone()
cursor.execute("SELECT id, comment, timestamp FROM feedback WHERE user_id
= %s ORDER BY timestamp DESC", (user_id,))
feedback = [type('Feedback', (), {'id': row[0], 'comment': row[1],
'timestamp': row[2]}) for row in cursor.fetchall()]
if request.method == 'POST':
    first_name = request.form.get('first_name')
    last_name = request.form.get('last_name')
    username = request.form.get('username')
    phone_number = request.form.get('phone_number')
    password = request.form.get('password')
    try:
        if password:
            hashed_password = generate_password_hash(password,
method='pbkdf2:sha256')
            cursor.execute("""
                UPDATE users SET first_name = %s, last_name = %s, username
= %s, phone_number = %s, password = %s
                WHERE id = %s
            """, (first_name, last_name, username, phone_number,
hashed_password, user_id))
        else:
            cursor.execute("""
                UPDATE users SET first_name = %s, last_name = %s, username
= %s, phone_number = %s
                WHERE id = %s
            """, (first_name, last_name, username, phone_number, user_id))
            conn.commit()
            flash('Profile updated successfully!', 'success')
    except Exception as e:
        flash('Error updating profile.', 'danger')
        print(f"Error: {e}")
    return redirect(url_for('profile'))
return render_template('profile.html', user=user, feedback=feedback)
```

- **URL:** /profile.
- **Purpose:** Shows and lets users edit their profile and view feedback.

WASLA - Number Prediction - Notes

- **Methods:** Handles GET (shows profile) and POST (updates profile).
- **What it does:**
 - Checks if the user is logged in. If not, redirects to /login.
 - Gets the user's ID from the session.
 - Fetches user details (first_name, last_name, etc.) from the database.
 - Fetches the user's feedback history, sorted by newest first.
 - For GET: Shows profile.html with user details and feedback.
 - For POST (form submission):
 1. Gets updated details from the form.
 2. If a new password is provided, hashes it and updates all fields, including the password.
 3. If no password, updates only other fields.
 4. Saves changes to the database.
 5. Shows a success or error message and reloads the profile page.
- **For beginners:** This is like your account settings page where you can update your name or password.

Route 9: History Page

python

Copy

```
@app.route('/history')
def history():
    if 'user_id' not in session:
        flash('Please log in to view your history.', 'warning')
        return redirect(url_for('login'))
    cursor.execute("SELECT id, message, timestamp FROM translations WHERE
user_id = %s ORDER BY timestamp DESC", (session['user_id'],))
    translations = [type('Translation', (), {'id': row[0], 'message': row[1],
'timestamp': row[2]}) for row in cursor.fetchall()]
    return render_template('history.html', translations=translations)
```

- **URL:** /history.
- **Purpose:** Shows the user's prediction history (e.g., digits recognized from images).
- **What it does:**
 - Checks if the user is logged in. If not, redirects to /login.
 - Fetches all translations (predictions) for the user from the translations table, sorted by newest first.
 - Creates a list of translation objects with id, message, and timestamp.
 - Shows history.html with the translations.
- **For beginners:** This is like a log of all the signs you've translated.

Route 10: Submit Feedback

python

Copy

WASLA - Number Prediction - Notes

```
@app.route('/submit_feedback', methods=['POST'])
def submit_feedback():
    if 'user_id' not in session:
        flash('Please log in to submit feedback.', 'warning')
        return redirect(url_for('login'))
    user_id = session['user_id']
    comment = request.form.get('comment', '').strip()
    try:
        cursor.execute("""
            INSERT INTO feedback (user_id, comment, timestamp)
            VALUES (%s, %s, %s)
            """, (user_id, comment, datetime.now()))
        conn.commit()
        flash('Feedback submitted successfully!', 'success')
    except Exception as e:
        flash('Error submitting feedback.', 'danger')
        print(f"Error: {e}")
    return redirect(url_for('profile'))
```

- **URL:** /submit_feedback.
- **Purpose:** Saves user feedback (comments about the app).
- **Methods:** Handles POST (form submission).
- **What it does:**
 - Checks if the user is logged in. If not, redirects to /login.
 - Gets the comment from the form and the user's ID.
 - Inserts the feedback into the feedback table with the current time (datetime.now()).
 - Saves changes to the database.
 - Shows a success or error message and redirects to /profile.
- **For beginners:** This is like leaving a review or comment about the app.

Route 11: Camera Page

python

Copy

```
@app.route('/camera', methods=['GET', 'POST'])
def camera():
    if 'user_id' not in session:
        flash('Please log in to access the camera.', 'warning')
        return redirect(url_for('login'))
    return render_template('camera.html')
```

- **URL:** /camera.

WASLA - Number Prediction - Notes

- **Purpose:** Shows a page where users can capture images (e.g., via webcam) for sign language prediction.
- **What it does:**
 - Checks if the user is logged in. If not, redirects to /login.
 - Loads camera.html (likely has JavaScript to access the webcam).
- **For beginners:** This is the page where you take a picture of a hand sign.

Route 12: Predict from Camera

python

Copy

```
@app.route('/predict_camera', methods=['POST'])
def predict_camera():
    if 'user_id' not in session:
        flash('Please log in to access the camera.', 'warning')
        return redirect(url_for('login'))

    if not model:
        flash('Model not loaded. Please contact the administrator.', 'danger')
        return redirect(url_for('camera'))

    image_data = request.form.get('image_data')
    if not image_data:
        flash('No image captured.', 'danger')
        return redirect(url_for('camera'))

    try:
        image_data = image_data.split(',')[1]
        image_bytes = base64.b64decode(image_data)
        image = Image.open(io.BytesIO(image_bytes))
    except Exception as e:
        flash('Error processing image.', 'danger')
        print(f"Error: {e}")
        return redirect(url_for('camera'))

    image_array = preprocess_image(image)
    prediction = model.predict(image_array)
    predicted_digit = np.argmax(prediction, axis=1)[0]

    prediction_text = DIGIT_TRANSLATIONS[predicted_digit]

    try:
        cursor.execute("""
            INSERT INTO translations (user_id, message, timestamp)
```

WASLA - Number Prediction - Notes

```
VALUES (%s, %s, %s)
""", (session['user_id'], f"{prediction_text['english']}
({prediction_text['arabic']}))", datetime.now()))
conn.commit()
except Exception as e:
    flash('Error saving prediction.', 'danger')
    print(f"Error: {e}")

return render_template('camera.html', prediction=prediction_text)
```

- **URL:** /predict_camera.
- **Purpose:** Processes an image from the webcam and predicts the sign language digit.
- **Methods:** Handles POST (image submission).
- **What it does:**
 - Checks if the user is logged in. If not, redirects to /login.
 - Checks if the model is loaded. If not, shows an error.
 - Gets the image_data (a base64-encoded image from the webcam).
 - If no image, shows an error.
 - Decodes the base64 image:
 - Removes the prefix (data:image/jpeg;base64,) and decodes the rest.
 - Converts it to an image using PIL.
 - If decoding fails, shows an error.
 - Preprocesses the image using preprocess_image.
 - Uses the model to predict the digit (0–9).
 - Gets the English and Arabic names from DIGIT_TRANSLATIONS.
 - Saves the prediction to the translations table.
 - Reloads camera.html with the prediction result.
- **For beginners:** This takes a photo, figures out what number it shows (like "5"), and saves it.

Route 13: Submit Prediction

python

Copy

```
@app.route('/submit_prediction', methods=['POST'])
def submit_prediction():
    if 'user_id' not in session:
        flash('Please log in to submit predictions.', 'warning')
        return redirect(url_for('login'))

    prediction_english = request.form.get('prediction_english')
    prediction_arabic = request.form.get('prediction_arabic')
    if not prediction_english or not prediction_arabic:
        flash('Invalid prediction data.', 'danger')
```

WASLA - Number Prediction - Notes

```
return redirect(url_for('camera'))

try:
    cursor.execute("""
        INSERT INTO translations (user_id, message, timestamp)
        VALUES (%s, %s, %s)
        """, (session['user_id'], f"{prediction_english}
({prediction_arabic})", datetime.now()))
    conn.commit()
    flash('Prediction saved successfully!', 'success')
except Exception as e:
    flash('Error saving prediction.', 'danger')
    print(f"Error: {e}")

return redirect(url_for('camera'))
```

- **URL:** /submit_prediction.
- **Purpose:** Saves a prediction manually (e.g., if submitted via a form instead of webcam).
- **Methods:** Handles POST.
- **What it does:**
 - Checks if the user is logged in.
 - Gets English and Arabic prediction text from the form.
 - If either is missing, shows an error.
 - Saves the prediction to the translations table.
 - Shows a success or error message and redirects to /camera.
- **For beginners:** This is like manually saving a translation result.

Route 14: Delete History

python

Copy

```
@app.route('/delete_history', methods=['POST'])
def delete_history():
    if 'user_id' not in session:
        flash('Please log in to delete history.', 'warning')
        return redirect(url_for('login'))

    try:
        cursor.execute("DELETE FROM translations WHERE user_id = %s",
(session['user_id'],))
        conn.commit()
        flash('Translation history deleted successfully!', 'success')
    except Exception as e:
        flash('Error deleting history.', 'danger')
```

WASLA - Number Prediction - Notes

```
print(f"Error: {e}")
```

```
return redirect(url_for('history'))
```

- **URL:** /delete_history.
- **Purpose:** Deletes all of the user's translation history.
- **Methods:** Handles POST.
- **What it does:**
 - Checks if the user is logged in.
 - Deletes all rows in the translations table for the user.
 - Saves changes.
 - Shows a success or error message and redirects to /history.
- **For beginners:** This clears your entire list of saved translations.

Route 15: Delete Single Translation

python

Copy

```
@app.route('/delete_translation', methods=['POST'])
def delete_translation():
    if 'user_id' not in session:
        flash('Please log in to delete translations.', 'warning')
        return redirect(url_for('login'))

    translation_id = request.form.get('translation_id')
    if not translation_id:
        flash('Invalid translation ID.', 'danger')
        return redirect(url_for('history'))

    try:
        cursor.execute("DELETE FROM translations WHERE id = %s AND user_id = %s", (translation_id, session['user_id']))
        conn.commit()
        if cursor.rowcount == 0:
            flash('Translation not found or not authorized.', 'danger')
        else:
            flash('Translation deleted successfully!', 'success')
    except Exception as e:
        flash('Error deleting translation.', 'danger')
        print(f"Error: {e}")

    return redirect(url_for('history'))
```

- **URL:** /delete_translation.

WASLA - Number Prediction - Notes

- **Purpose:** Deletes a specific translation.
- **Methods:** Handles POST.
- **What it does:**
 - Checks if the user is logged in.
 - Gets the translation_id from the form.
 - If no ID, shows an error.
 - Deletes the translation with the given ID, but only if it belongs to the user.
 - If no rows are deleted (e.g., wrong ID or not the user's), shows an error.
 - Shows a success or error message and redirects to /history.
- **For beginners:** This deletes one specific translation from your history.

Route 16: Delete Single Feedback

python

Copy

```
@app.route('/delete_feedback', methods=['POST'])
def delete_feedback():
    if 'user_id' not in session:
        flash('Please log in to delete feedback.', 'warning')
        return redirect(url_for('login'))

    feedback_id = request.form.get('feedback_id')
    if not feedback_id:
        flash('Invalid feedback ID.', 'danger')
        return redirect(url_for('profile'))

    try:
        cursor.execute("DELETE FROM feedback WHERE id = %s AND user_id = %s",
            (feedback_id, session['user_id']))
        conn.commit()
        if cursor.rowcount == 0:
            flash('Feedback not found or not authorized.', 'danger')
        else:
            flash('Feedback deleted successfully!', 'success')
    except Exception as e:
        flash('Error deleting feedback.', 'danger')
        print(f"Error: {e}")

    return redirect(url_for('profile'))
```

- **URL:** /delete_feedback.
- **Purpose:** Deletes a specific feedback entry.
- **Methods:** Handles POST.
- **What it does:**

WASLA - Number Prediction - Notes

- Checks if the user is logged in.
- Gets the feedback_id from the form.
- If no ID, shows an error.
- Deletes the feedback with the given ID, but only if it belongs to the user.
- If no rows are deleted, shows an error.
- Shows a success or error message and redirects to /profile.
- **For beginners:** This deletes one comment you left about the app.

Route 17: Delete All Feedback

python

Copy

```
@app.route('/delete_all_feedback', methods=['POST'])
def delete_all_feedback():
    if 'user_id' not in session:
        flash('Please log in to delete feedback.', 'warning')
        return redirect(url_for('login'))

    try:
        cursor.execute("DELETE FROM feedback WHERE user_id = %s",
            (session['user_id'],))
        conn.commit()
        flash('All feedback deleted successfully!', 'success')
    except Exception as e:
        flash('Error deleting feedback.', 'danger')
        print(f"Error: {e}")

    return redirect(url_for('profile'))
```

- **URL:** /delete_all_feedback.
- **Purpose:** Deletes all of the user's feedback.
- **Methods:** Handles POST.
- **What it does:**
 - Checks if the user is logged in.
 - Deletes all rows in the feedback table for the user.
 - Saves changes.
 - Shows a success or error message and redirects to /profile.
- **For beginners:** This clears all your comments about the app.

Final Block: Run the App

python

Copy

```
if __name__ == "__main__":
    app.run(debug=True)
```

WASLA - Number Prediction - Notes

- **Purpose:** Starts the Flask web server.
- **What it does:**
 - `if __name__ == "__main__":` Ensures the server runs only if this file is executed directly (not imported).
 - `app.run(debug=True)`: Starts the app in debug mode (shows detailed errors and reloads automatically on code changes).
- **For beginners:** This is like pressing "start" to make the website live on your computer.

What the App Does Overall

This is a web app for recognizing sign language digits (0–9) from webcam images. Users can:

- Register and log in securely.
- Reset their password if forgotten.
- Update their profile (name, username, etc.).
- Use a webcam to capture hand signs, which a machine learning model translates to digits (in English and Arabic).
- Save and view their prediction history.
- Submit and manage feedback about the app.
- Log out when done.

The app uses:

- **Flask:** To create web pages and handle user actions.
- **MySQL:** To store user data, predictions, and feedback.
- **TensorFlow:** To predict digits from images.
- **HTML templates:** For the user interface (not shown in the code but referenced, like `home.html`).
- **Base64 and PIL:** To handle webcam images.

These are some simple definitions of the terms used above:

- **Routes** are like different pages or actions in the app (e.g., `/login` for logging in).
- **Templates** (like `login.html`) are the HTML files that create the web pages users see.
- **Database** (MySQL) stores information like usernames and predictions.
- **Session** keeps track of who's logged in.
- **Flash messages** show quick notifications (e.g., "Wrong password!").
- The **machine learning model** is the "brain" that recognizes hand signs.
- **Error handling** (`try/except`) prevents the app from crashing if something goes wrong.

WASLA - Number Prediction - Notes

Explanation of HTML Files

File 1: camera.html

Purpose: This page allows users to capture or upload images for sign language digit recognition, view predictions, and navigate to other parts of the app.

Structure and Features:

1. Head Section:

- Sets the page title to "Wasla - Camera".
- Includes Tailwind CSS via a CDN for styling (e.g., bg-[#8dd97d] for a green background).
- Uses meta tags for UTF-8 encoding and responsive viewport (width=device-width).

2. Body Section:

- **Flash Messages:** Displays success (green) or error (red) messages using Jinja2's get_flashed_messages (e.g., "Camera not active").
- **Header:**
 - Shows a "Back" link to /login, the Wasla logo (100x100px), and a "Logout" link with an emoji (👤).
 - Centered layout with flex justify-between.
- **Camera View:**
 - A black box (w-full max-w-xs h-48) contains a <video> element (id="video") for live webcam feed, set to autoplay.
 - A hidden <canvas> (id="canvas") captures the video frame for processing.
- **Camera Controls:**
 - A "Flip Camera" button (🔄) toggles between front and back cameras.
 - A toggle switch (<input type="checkbox" id="cameraToggle">) turns the camera on/off, styled as a sliding button.
- **Capture and Upload Form:**
 - A <form> submits to /predict_camera with a hidden input (image_data) for base64-encoded images.
 - A "Capture & Predict" button triggers webcam capture.
 - An <input type="file"> allows uploading JPEG/PNG images, paired with a "Predict" button.
- **Prediction Display:**
 - If a prediction exists ({% if prediction %}), shows a white card with the English and Arabic translation (e.g., "Five (خمسة)").
- **Bottom Navigation:**
 - Fixed at the bottom, links to /history (📜) and /profile (👤) with icons and labels.

3. JavaScript:

WASLA - Number Prediction - Notes

- **Variables:** References DOM elements (e.g., video, canvas, captureButton) for interaction.
- **Camera Functions:**
 - startCamera(): Requests webcam access using navigator.mediaDevices.getUserMedia, with facingMode (user or environment) for front/back camera.
 - stopCamera(): Stops the webcam stream.
- **Event Listeners:**
 - cameraToggle: Starts/stops the camera when toggled.
 - flipCamera: Switches facingMode and restarts the camera if active.
 - captureButton: Captures the video frame to the canvas, converts it to a base64 JPEG (canvas.toDataURL), and submits the form.
 - uploadButton: Reads an uploaded image as base64 using FileReader, validates it (JPEG/PNG), and submits the form.
- **Initialization:** Starts the camera if the toggle is checked on page load.

Short Guide: This is the main interaction page where you use your webcam or upload a photo to recognize a hand sign, see the result, and navigate to your history or profile.

File 2: forgot_password.html

Purpose: Allows users to enter their username to initiate a password reset.

Structure and Features:

1. **Head Section:**
 - Title: "Wasla - Forgot Password".
 - Includes Tailwind CSS and standard meta tags.
2. **Body Section:**
 - **Flash Messages:** Shows messages like "Username not found".
 - **Header:**
 - Displays the Wasla logo (150x150px) and the app name ("wasla") in large, bold text.
 - **Forgot Password Form:**
 - A white card (max-w-xs) contains a form submitting to /forgot_password.
 - Single input for username, marked as required.
 - A "Proceed" button submits the form.
 - No navigation links, keeping the focus on password recovery.




Short Guide: This is a simple page where you type your username to start resetting your password, like the first step of a "forgot password" flow.

File 3: history.html

Purpose: Displays the user's translation history and allows deleting individual or all translations.

WASLA - Number Prediction - Notes

Structure and Features:

1. **Head Section:**
 - Title: "Wasla - History".
 - Includes Tailwind CSS and meta tags.
2. **Body Section:**
 - **Flash Messages:** Shows messages like "Translation deleted successfully".
 - **Header:**
 - Includes a "Back" link to /camera, the Wasla logo (150x150px), and a "Logout" link ().
 - **Translation History:**
 - A heading ("Translation History") with a "Delete All" button (red, submits to /delete_history).
 - If translations exist ({% if translations %}):
 - Loops through translations, showing each in a white card with:
 - The translation message (e.g., "Five (خمسة)").
 - Timestamp (formatted as YYYY-MM-DD HH:MM:SS).
 - A "Delete" button (red, submits to /delete_translation with a hidden translation_id).
 - If no translations, shows "No translations yet".
 - **Bottom Navigation:**
 - Links to /history () and /profile (), same as camera.html.

Short Guide: This page lists all your past sign language predictions, letting you delete one or all of them, with easy navigation to other pages.

File 4: home.html

Purpose: The landing page introducing the app and directing users to log in.

Structure and Features:

1. **Head Section:**
 - Title: "Wasla - Home".
 - Includes Tailwind CSS and meta tags.
2. **Body Section:**
 - **Header:**
 - Shows a larger Wasla logo (wasla_full.png, 300x300px).
 - A tagline: "DEAF AND MUTE COMMUNITY" in white, bold text.
 - **Sign-In Button:**
 - A single button links to /login, styled as a white, shadowed button with "Sign In".
 - No flash messages or navigation bar, keeping it minimal.


WASLA - Number Prediction - Notes

Short Guide: This is the first page you see, like a welcome screen that invites you to log in, with a focus on the app's purpose for the deaf and mute community.

File 5: login.html

Purpose: Allows users to sign in with their username and password, with options for password recovery or registration.

Structure and Features:


1. **Head Section:**
 - Title: "Wasla - Sign In".
 - Includes Tailwind CSS and meta tags.
2. **Body Section:**
 - **Flash Messages:** Shows messages like "Incorrect password".
 - **Header:**
 - Displays the Wasla logo (wasla_full.png, 150x150px).
 - A "Welcome!" heading.
 - **Sign-In Form:**
 - A white card with:
 - A gray avatar icon () in a circle.
 - Inputs for username and password (both required).
 - Links to /forgot_password ("Forgot Password?") and /register ("Don't have an account?").
 - An "Enter" button submits to /login_validation.
 - **Bottom Navigation:**
 - Empty <div>, likely a placeholder (incomplete compared to other pages).

Short Guide: This is where you log in with your username and password, with links to recover your password or sign up if you're new.

File 6: profile.html

Purpose: Allows users to update their profile, submit feedback, and view or delete past feedback.

Structure and Features:

1. **Head Section:**
 - Title: "Wasla - Profile".
 - Includes Tailwind CSS and meta tags.
2. **Body Section:**
 - **Flash Messages:** Shows messages like "Profile updated successfully".
 - **Header:**
 - Includes a "Back" link to /camera, the Wasla logo (100x100px), and a "Logout" link ().

WASLA - Number Prediction - Notes

- **Profile Form:**
 - A white card with:
 - A gray avatar icon (👤).
 - Inputs pre-filled with user data (user[0] for first_name, etc.) for first_name, last_name, username, phone_number (all required), and an optional password.
 - An "Update Profile" button submits to /profile.
- **Feedback Submission:**
 - A white card with a <textarea> for feedback (comment) and a green "Submit" button (submits to /submit_feedback).
- **Feedback History:**
 - A white card with:
 - A "Delete All" button (red, submits to /delete_all_feedback).
 - If feedback exists ({% if feedback %}), loops through feedback, showing:
 - The comment text and timestamp (YYYY-MM-DD HH:MM:SS).
 - A "Delete" button (red, submits to /delete_feedback with a hidden feedback_id).
 - If no feedback, shows "No feedback submitted yet".
- **Bottom Navigation:**
 - Links to /history (📄) and /profile (👤), same as camera.html.

Short Guide: This page lets you edit your account details, leave feedback about the app, and see or delete your past comments, with navigation to other sections.

File 7: register.html

Purpose: Allows new users to create an account by entering their details.

Structure and Features:

1. **Head Section:**
 - Title: "Wasla - Register".
 - Includes Tailwind CSS and meta tags.
2. **Body Section:**
 - **Flash Messages:** Shows messages like "Username already exists".
 - **Header:**
 - Displays the Wasla logo (100x100px) and the app name ("wasla").
 - **Register Form:**
 - A white card with:
 - A gray avatar icon (👤).
 - Inputs for first_name, last_name, username, phone_number, password, and confirm_password (all required).
 - A link to /login ("Already a member? Login Here").

WASLA - Number Prediction - Notes

- A "Register" button submits to /register.
- No navigation bar, focusing on registration.

Short Guide: This is where you sign up for a new account by filling in your details, with an option to go back to the login page if you already have an account.

File 8: reset_password.html

Purpose: Allows users to set a new password after verifying their username.

Structure and Features:

1. **Head Section:**
 - Title: "Wasla - Reset Password".
 - Includes Tailwind CSS and meta tags.
2. **Body Section:**
 - **Flash Messages:** Shows messages like "Passwords do not match".
 - **Header:**
 - Displays the Wasla logo (100x100px) and the app name ("wasla").
 - **Reset Password Form:**
 - A white card with:
 - A hidden input for username (pre-filled from the previous step).
 - Inputs for new_password and confirm_password (both required).
 - A "Submit" button submits to /reset_password.
 - No navigation links, keeping the focus on password reset.

Short Guide: This is the final step of resetting your password, where you enter and confirm a new password after proving your username.

Summary of Pages

There are 8 HTML files, each serving a distinct purpose in the Wasla app:

1. **camera.html:** Captures or uploads images for sign language prediction.
2. **forgot_password.html:** Initiates password reset by entering a username.
3. **history.html:** Shows and manages translation history.
4. **home.html:** The landing page with a sign-in button.
5. **login.html:** Handles user login with recovery and registration links.
6. **profile.html:** Manages user profile and feedback.
7. **register.html:** Creates new user accounts.
8. **reset_password.html:** Sets a new password.

Common Features Across Files:

- All use Tailwind CSS for responsive, styled layouts (e.g., green background #8dd97d, white cards with shadows).

WASLA - Number Prediction - Notes

- Most include flash messages for user feedback.
- Pages like camera.html, history.html, and profile.html have a bottom navigation bar for /history and /profile.
- Logos (wasla_logo.png or wasla_full.png) and headers provide consistent branding.
- Forms use POST methods to securely send data to Flask routes.

Short Summary: These HTML files are the visual parts of the app you see in your browser. Each page has a specific job (like logging in or showing predictions), and they work with the Python code to make the app interactive. If you're new, try opening these files in a browser (after running the Flask app) to see how they look, and check the browser's developer tools to explore the JavaScript or styling. Let me know if you need help with setup or specific parts!

The End