app.py

```
from flask import Flask, request, jsonify, render_template, redirect, url_for, session
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
app = Flask(__name__)
app.secret_key = 'supersecretkey'
# Sample user credentials
users = {
    'Davish': 'davish',
    '2': 'k2'
    }
# Sample Data Preparation
np.random.seed(42)
num_customers = 1000
num_products = 500
# Updated Sample Data Preparation
fixed_customers = ['1', '2', '3', '4', '5']
fixed_products = ['avocado', 'broccoli', 'carrot', 'dragonfruit', 'elderberry', 'fig', 'grape', 'honeydew',
'iceberg lettuce', 'jalapeno']
fixed_purchases = {
    'customer_id': ['1', '1', '1', '2', '2', '2', '3', '3', '3', '4', '4', '4', '5', '5', '5'],
    'product_id': ['avocado', 'broccoli', 'carrot', 'broccoli', 'dragonfruit', 'elderberry', 'carrot', 'fig',
'grape', 'fig', 'grape', 'honeydew', 'iceberg lettuce', 'jalapeno', 'avocado']
}

# Additional fixed data and random data generation remain unchanged
fixed_df = pd.DataFrame(fixed_purchases)
additional_fixed_data = {
    'customer_id': np.random.choice(fixed_customers, size=35, replace=True),
    'product_id': np.random.choice(fixed_products, size=35, replace=True)
}
additional_fixed_df = pd.DataFrame(additional_fixed_data)
fixed_df = pd.concat([fixed_df, additional_fixed_df], ignore_index=True)

random_data = {
    'customer_id': np.random.randint(6, num_customers + 1, size=4950).astype(str),
    'product_id': np.random.randint(11, num_products + 1, size=4950).astype(str)
}
random_df = pd.DataFrame(random_data)
df = pd.concat([fixed_df, random_df], ignore_index=True)
df['customer_id'] = df['customer_id'].astype(str)
df['product_id'] = df['product_id'].astype(str)

user_product_matrix = df.pivot_table(index='customer_id', columns='product_id', aggfunc='size',
fill_value=0)
# Calculate item-item similarity matrix
item_similarity = cosine_similarity(user_product_matrix.T)
item_similarity_df = pd.DataFrame(item_similarity, index=user_product_matrix.columns,
columns=user_product_matrix.columns)
def recommend_products(customer_id, n=5):
```

```
    if customer_id not in user_product_matrix.index:
        return [], []
    user_purchases = user_product_matrix.loc[customer_id]
    purchased_products = user_purchases[user_purchases > 0].index.tolist()
    similar_products = []
    for product in purchased_products:
        similar = item_similarity_df[product].sort_values(ascending=False)[1:n+1].index.tolist()
        similar_products.extend(similar)
    recommendations = list(set(similar_products) - set(purchased_products))
    recommendations = sorted(recommendations,
                             key=lambda x: item_similarity_df.loc[purchased_products, x].mean(),
                             reverse=True)

    return purchased_products, recommendations[:n]
@app.route('/')
def index():
    if 'username' not in session:
        return redirect(url_for('login'))
    return render_template('index.html')
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        if username in users and users[username] == password:
            session['username'] = username
            return jsonify({'success': True})
        return jsonify({'error': 'Invalid username or password'})
    return render_template('login.html')
@app.route('/recommend', methods=['POST'])
def recommend():
    if 'username' not in session:
        return jsonify({"error": "Unauthorized"}), 401
    customer_id = request.form.get('customer_id')
    purchased_products, recommendations = recommend_products(customer_id
    if not purchased_products:
        return jsonify({"error": "Customer not found"}), 404

    return jsonify({
       'purchased': purchased_products,
        'recommendations': recommendations
    })

@app.route('/logout')
def logout():
    session.pop('username', None)
    return redirect(url_for('login'))

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