Niyati Gupta | gniyati20@gmail.com | https://github.com/nickoo20

UrbanMatch- Python Task

The Marriage Matchmaking App is a web application built using FastAPI. It provides an API for managing user profiles, including creating, reading, updating, deleting users, and finding potential matches based on shared interests or location.

Marriage Matchmaking App Report

Overview

The Marriage Matchmaking App is a web application built using FastAPI. It provides an API for managing user profiles, including creating, reading, updating, deleting users, and finding potential matches based on shared interests or location.

Features:

1. **Create User**: Add a new user to the system with details such as name, age, gender, email, city, and interests.
2. **Read User**: Retrieve details of individual users or list all users in the system.
3. **Update User**: Modify existing user details.
4. **Delete User**: Remove users from the system.
5. **Find Matches**: Identify users who share similar interests or are located in the same city as a given user.

Technologies Used

- **FastAPI**: A modern, fast (high-performance) web framework for building APIs with Python 3.7+.
- **SQLAlchemy**: SQL toolkit and Object-Relational Mapping (ORM) library for Python.
- **SQLite**: Lightweight, disk-based database.
- **Pydantic**: Data validation and settings management using Python type annotations.

API Endpoints

- **GET** `/users/`: Retrieve all users.
- **POST** `/users/`: Create a new user.
- **GET** `/users/{user_id}`: Retrieve a specific user by ID.
- **PUT** `/users/{user_id}`: Update a specific user by ID.
- **DELETE** `/users/{user_id}`: Delete a specific user by ID.
- **GET** `/users/{user_id}/matches`: Find matches for a specific user based on interests or location.

Niyati Gupta | gniyati20@gmail.com | https://github.com/nickoo20

Assumptions Made

1. **Interest Storage**: Interests are stored as a single string in the database rather than a list of strings. This assumption simplifies the storage and retrieval of user interests, treating interests as a single text field.
2. **Email Validation**: The system uses `EmailStr` from Pydantic to ensure that email addresses are valid and properly formatted.
3. **Unique Email Constraint**: The email field in the `User` model is unique, preventing duplicate email entries.
4. **Interest Matching Logic**: The system assumes that users with identical interest strings or located in the same city are potential matches. This logic is implemented using simple string comparison.