Marriage Matchmaking App

1. Modules Changes and Assumptions

1. **Database Setup (`database.py`)**
    - **Configuration**: Connected to PostgreSQL using SQLAlchemy.
    - **Changes**: No changes required here.
    - **Assumptions**: PostgreSQL is correctly installed and running on the local machine.

2. **Schemas (`schemas.py`)**
    - **Class `UserBase`**:
        - **Purpose**: Base class for user-related schemas.
        - **Changes**: No changes needed here.
    - **Class `UserCreate`**:
        - **Purpose**: Used for creating new users.
        - **Changes**: No changes needed here.
    - **Class `UserUpdate`**:
        - **Purpose**: Allows partial updates of user fields.
        - **Changes**: All fields are made optional to support partial updates.
    - **Class `User`**:
        - **Purpose**: Represents the user model including `id`.
        - **Changes**: No changes needed here.
3. **Main Application (`main.py`)**
    - **Endpoints**:
        - `/users/` **(POST)**:
            - **Purpose**: Create a new user.
            - **Changes**: Accepts user input and creates a new user entry in the database.
        - `/users/{user_id}` **(PUT)**:
            - **Purpose**: Update an existing user.
            - **Changes**: Utilizes `UserUpdate` schema to update only provided fields, leaving unspecified fields unchanged.
        - `/users/{user_id}` **(DELETE)**:
            - **Purpose**: Delete a user by ID.
            - **Changes**: Removes a user entry based on the provided ID.
        - `/users/{user_id}/matches` **(GET)**:
            - **Purpose**: Find users with matching interests but different genders.

        - **Changes**: Retrieves users with overlapping interests and different genders.

5. Assumptions

1. **Email Validation**:
    - Ensured by using `EmailStr` in `UserBase` and `UserUpdate` schemas.
2. **Database Operation**:
    - PostgreSQL is assumed to be running and accessible.
    - `ARRAY` type is used for storing lists of interests in PostgreSQL.
3. **User Update**:
    - Assumes that partial updates are correctly handled by checking and updating only the provided fields.
4. **User Match**:
    - Assumes that users of different genders with at least one same interest are a match.

6. Testing

1. **Create Users**:
    - Verified using curl commands to ensure users are created with the provided details.
2. **Update Users**:
    - Verified that users can be updated by partially providing fields, and non-specified fields remain unchanged.
3. **Delete Users**:
    - Verified that users can be deleted by ID.
4. **Find Matches**:
    - Checked that the `find_matches` endpoint retrieves users with matching interests and different genders.
5. **Email Validation**:
    - Ensured that email fields adhere to the `EmailStr` validation.

Used both curl requests and Swagger UI for manual testing.