**Tasks Completed**

1. **Update User Endpoint:**
   * Added an endpoint to update user profiles based on their ID.
   * Utilized FastAPI’s dependency injection to interact with the database and update only the provided fields using the exclude\_unset feature from Pydantic.
2. **Delete User Endpoint:**
   * Implemented an endpoint to delete user profiles based on their ID.
   * Ensured proper error handling for non-existent user IDs with descriptive error messages.
3. **Find Matches for a User:**
   * Developed a matching algorithm to retrieve potential matches based on common attributes such as city and interests.
   * Ensured efficient querying using SQLAlchemy filters.
4. **Email Validation:**
   * Integrated email validation by updating the schemas.py file to use Pydantic’s EmailStr type.
   * This ensures that invalid email formats are rejected during user creation or updates.

**Assumptions Made**

* Matches are identified based on exact matches of the city and interests fields. This approach was chosen for simplicity and can be extended for partial or weighted matches.
* Profiles are uniquely identified by their id in the database.
* Users can update only the fields they provide, leaving other fields unchanged.

**Testing**

All endpoints were thoroughly tested using **Postman**:

* **Create User**: Verified successful creation and validation for required fields.
* **Update User**: Confirmed that fields were correctly updated and handled missing or invalid IDs.
* **Delete User**: Ensured users were deleted and proper error messages were returned for invalid IDs.
* **Find Matches**: Checked accuracy of matches based on criteria (city and interests).
* **Email Validation**: Tested valid and invalid email formats to confirm robust validation.