

Milan Mandap

Modern Matchmaking Rooted in Tradition

Introduction

Milan-Mandap is a modern matchmaking backend application designed to cater to Indian users. It seamlessly blends traditional values with advanced matchmaking algorithms to create personalized connections based on interests, location, and preferences.

Features

1. User Management:

- Create, update, retrieve, and delete user profiles.
- Store user information such as name, age, gender, city, email, and interests.

2. Smart Matchmaking:

Matches users based on:

- Common Interests (highest priority).
- Same City or State (prioritized for Indian users).
- Age Compatibility (± 2 years).
- Uses a priority-based algorithm to find potential matches.

3. Organized Backend Architecture: Ensures clean and modularized code with components like models, schemas, and database management.

4. JSON APIs: Provides RESTful APIs for all features, including user management and matchmaking.

Matchmaking Algorithm

The matchmaking algorithm uses priority-based sorting to identify the best potential matches for a user. These priorities can be changed based on requirements, will be helpful to implement filter wise sorting. Currently showing all opposite gender users in priority order, but can change it to show maximum n (selected by user) matches.

1. Common Interests (Priority 1): Matches users with overlapping interests.
2. Same City or State (Priority 2): Matches users in the same city or state for proximity.
3. Age Compatibility (Priority 3): Matches users with an age difference of ± 2 years.
4. Fallback: Shows matches in the same country if no criteria are met.

API Endpoints

Retrieve all users: `GET /users`

Create a new user: `POST /users`

Retrieve a user by ID: `GET /users/<user_id>`

Update a user: `PUT /users/<user_id>`

Delete a user: `DELETE /users/<user_id>`

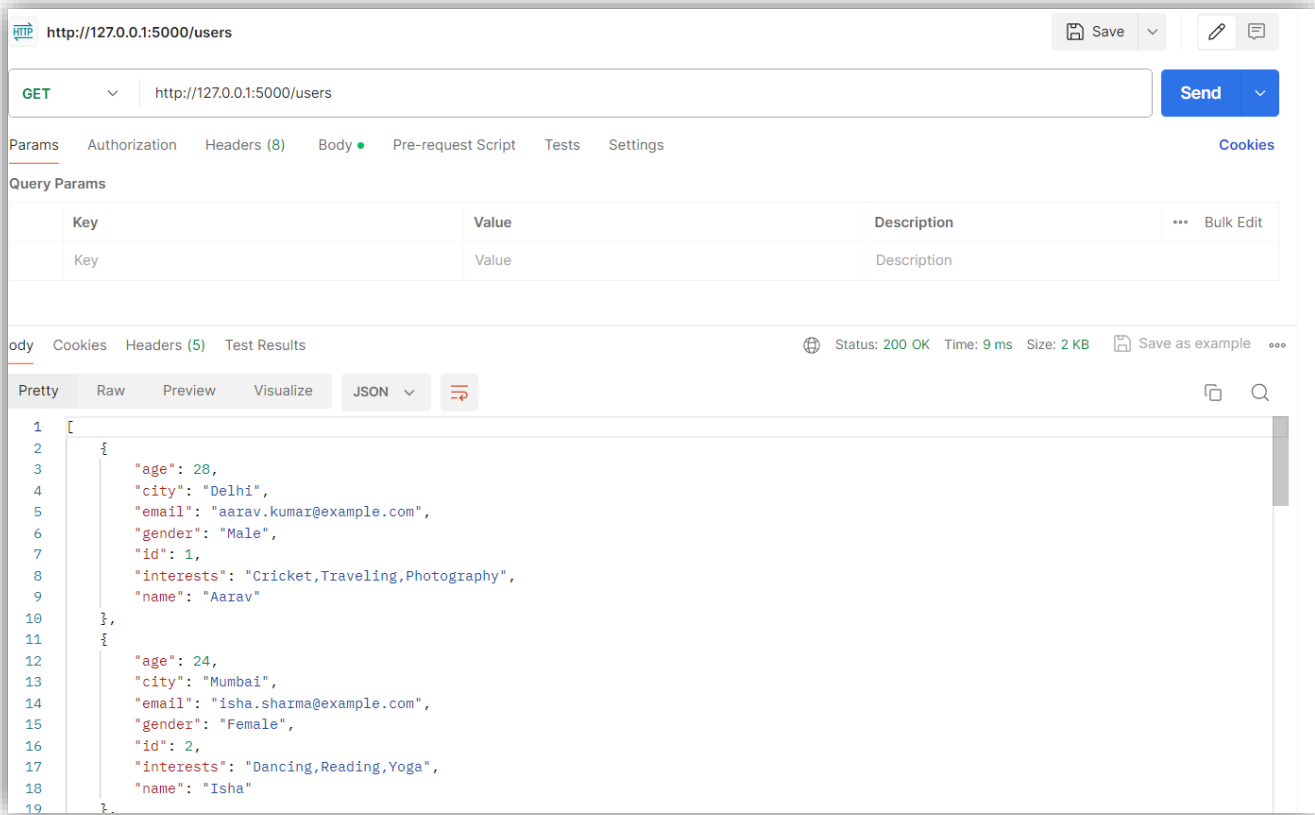
Find potential matches: `GET /users/<user_id>/matches`

Potential Improvements

- Add authentication and authorization to secure user data.
- Develop a frontend UI for easier user interaction.
- Expand geolocation features for global cities and states.
- Implement a messaging feature for matched users.
- Use machine learning algorithms to optimize matchmaking.
- Enhance user profiles with more data for improved matching.

Testing Screenshots

1. Fetch all users



HTTP <http://127.0.0.1:5000/users> Save

GET <http://127.0.0.1:5000/users> Send

Params Authorization Headers (8) Body • Pre-request Script Tests Settings Cookies

Query Params

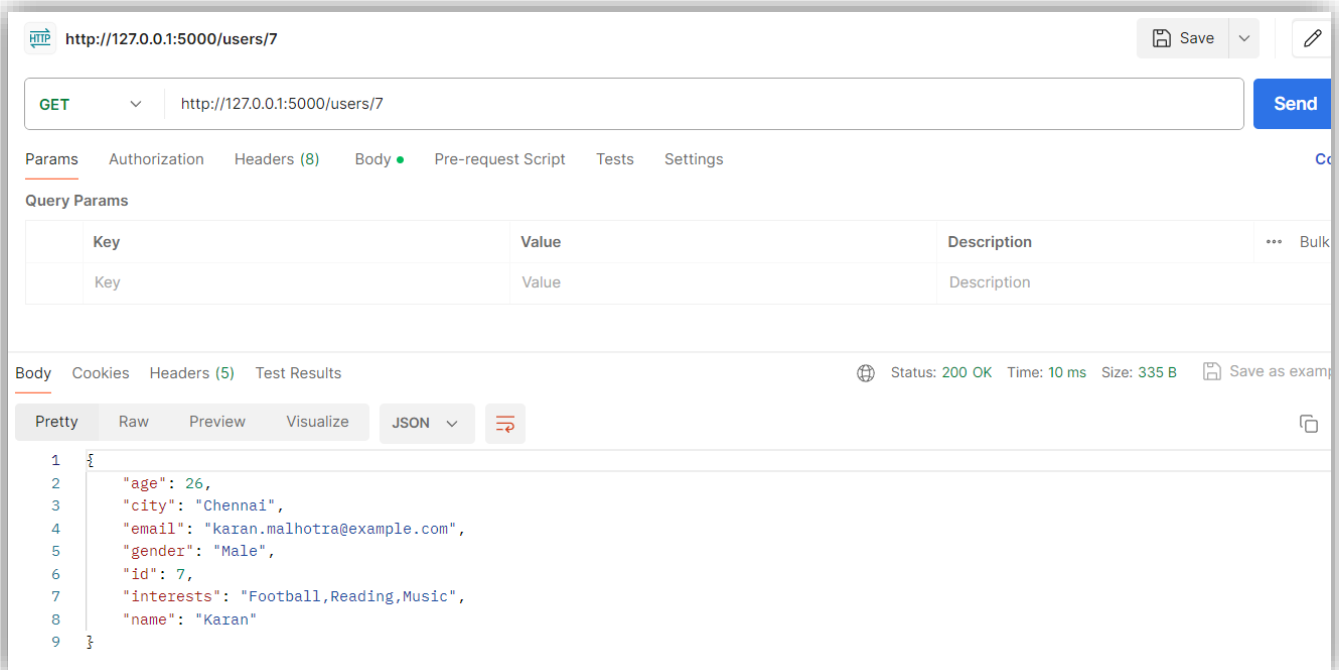
Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (5) Test Results Status: 200 OK Time: 9 ms Size: 2 KB Save as example

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "age": 28,
4     "city": "Delhi",
5     "email": "aarav.kumar@example.com",
6     "gender": "Male",
7     "id": 1,
8     "interests": "Cricket,Traveling,Photography",
9     "name": "Aarav"
10  },
11  {
12    "age": 24,
13    "city": "Mumbai",
14    "email": "isha.sharma@example.com",
15    "gender": "Female",
16    "id": 2,
17    "interests": "Dancing,Reading,Yoga",
18    "name": "Isha"
19  }
20 ]
```

2. Fetch user by id



HTTP <http://127.0.0.1:5000/users/7> Save

GET <http://127.0.0.1:5000/users/7> Send

Params Authorization Headers (8) Body • Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk
Key	Value	Description		

Body Cookies Headers (5) Test Results Status: 200 OK Time: 10 ms Size: 335 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "age": 26,
3   "city": "Chennai",
4   "email": "karan.malhotra@example.com",
5   "gender": "Male",
6   "id": 7,
7   "interests": "Football,Reading,Music",
8   "name": "Karan"
9 }
```

3. Create user

The screenshot shows a REST client interface with the following details:

- URL:** `http://127.0.0.1:5000/users`
- Method:** `POST`
- Body (Request):**

```
1 {
2   "age": 29,
3   "city": "Bhopal",
4   "email": "Awdesh.sharma@gmail.com",
5   "gender": "Male",
6   "id": 11,
7   "interests": "Swimming, Photography, Cricket",
8   "name": "Awdesh"
9 }
```
- Status:** `201 CREATED`
- Time:** `15 ms`
- Size:** `215 B`
- Body (Response):**

```
1 {
2   "message": "User created successfully"
3 }
```

4. Update user

The screenshot shows a REST client interface with the following details:

- URL:** `http://127.0.0.1:5000/users/7`
- Method:** `PUT`
- Body (Request):**

```
1 {
2   "age": 23,
3   "city": "Indore",
4   "email": "sagar.jain@example.com",
5   "gender": "Male",
6   "id": 7,
7   "interests": "Swimming, Photography, Cricket",
8   "name": "Sagar"
9 }
```
- Status:** `200 OK`
- Time:** `16 ms`
- Size:** `210 B`
- Body (Response):**

```
1 {
2   "message": "User updated successfully"
3 }
```

5. Find optimal matches

The screenshot shows a REST client interface with the following details:

- URL:** `http://127.0.0.1:5000/users/6/matches`
- Method:** `GET`
- Status:** `200 OK`, **Time:** `8 ms`, **Size:** `1.18 KB`
- Body:** A JSON array of three user match objects.

Query Params:

Key	Value	Description
-----	-------	-------------

Body (JSON):

```
[
  {
    "age": 23,
    "city": "Indore",
    "email": "sagar.jain@example.com",
    "gender": "Male",
    "id": 7,
    "interests": "Swimming,Photography,Cricket",
    "name": "Sagar",
    "priority": 70
  },
  {
    "age": 28,
    "city": "Delhi",
    "email": "aarav.kumar@example.com",
    "gender": "Male",
    "id": 1,
    "interests": "Cricket,Traveling,Photography",
    "name": "Aarav",
    "priority": 40
  },
  {
    "age": 30,
    "city": "Bangalore"
  }
]
```