Detailed API Documentation

Overview

This API provides endpoints for user registration, authentication, profile management, and dashboard functionality for a matrimonial/dating application.

Base URL

```
https://your-domain.com/api/v1
```

Authentication

Most endpoints require authentication using Bearer token. Include the token in the Authorization header:

```
Authorization: Bearer {your_token}
```

Response Format

All API responses follow this standard format:

Success Response

```
{
    "status": true,
    "message": "Success message",
    "data": {
        // Response data
    }
}
```

Error Response

```
{
    "status": false,
    "message": "Error message",
    "errors": {
        // Validation errors (if any)
    }
}
```

---

1. User Registration

Register New User

POST `/register`

Creates a new user account with basic information.

Request Body

```
{
    "firstname": "John",
    "lastname": "Doe",
    "email": "john.doe@example.com",
    "password": "password123",
    "password_confirmation": "password123",
    "gender": "m",
    "agree": true,
}
```

Validation Rules

- `firstname`: required, string
- `lastname`: required, string
- `email`: required, email format, unique in users table
- `password`: required, minimum 6 characters, confirmed
- `password_confirmation`: required, must match password
- `gender`: required, must be "m" or "f"
- `agree`: required if gs('agree') is true

Password Complexity (if enabled)

If `gs('secure_password')` is enabled, password must contain:

- Mixed case letters
- Numbers
- Symbols
- Not compromised (checked against breach databases)

Success Response (200)

```
{
    "status": true,
    "message": "Registration successful",
    "data": {
        "user": {
            "id": 1,
            "profile_id": "12345678",
            "firstname": "John",
            "lastname": "Doe",
            "email": "john.doe@example.com",
            "gender": "m",
            "profile_complete": 0,
            "completed_step": [],
            "skipped_step": [],
            "created_at": "2024-01-01T00:00:00.000000Z",
            "updated_at": "2024-01-01T00:00:00.000000Z"
        },
        "token": "1|abc123def456..."
    }
}
```

Error Responses

- **422 Unprocessable Entity**: Validation errors
- **400 Bad Request**: Invalid captcha or registration disabled
- **403 Forbidden**: Registration disabled

---

2. User Authentication

Login

POST `/v1/user/login`

Authenticates a user and returns an access token. Supports login with email or username.

Request Body

```
{
    "username": "john.doe@example.com",
    "password": "password123"
}
```

Validation Rules

- `username`: required, string (email or username)
- `password`: required, string

Authentication Process

1. Determines if username is email or username
2. Attempts authentication with provided credentials
3. Logs user login information (IP, browser, OS, location)
4. Creates and returns access token

Success Response (200)

```
{
    "status": true,
    "message": "Login successful",
    "data": {
        "token": "1|abc123def456...",
        "user": {
            "id": 1,
            "profile_id": "12345678",
            "firstname": "John",
            "lastname": "Doe",
            "email": "john.doe@example.com",
            "username": "john_doe",
            "gender": "m",
            "profile_complete": 0,
            "completed_step": [],
            "skipped_step": [],
            "created_at": "2024-01-01T00:00:00.000000Z",
            "updated_at": "2024-01-01T00:00:00.000000Z"
        }
    }
}
```

Error Responses

- **422 Unprocessable Entity**: Validation errors

- **401 Unauthorized**: Invalid credentials

Logout

POST `/v1/user/logout`

Logs out the authenticated user by revoking the current token.

Headers

```
Authorization: Bearer {your_token}
```

Success Response (200)

```
{
    "status": true,
    "message": "Logout successful",
    "data": null
}
```

Error Responses

- **401 Unauthorized**: Invalid or missing token

---

3. User Profile Management

Get User Data

GET `/v1/user/user-data`

Retrieves user data for profile completion steps. Returns different data based on current step progress.

Headers

```
Authorization: Bearer {your_token}
```

Step Progression Logic

The API determines the current step based on completed and skipped steps:

- Step 0: basicInfo (if no steps completed)
- Step 1: familyInfo (if 1 step completed)
- Step 2: educationInfo (if 2 steps completed)

- Step 3: careerInfo (if 3 steps completed)
- Step 4: physicalAttributeInfo (if 4 steps completed)
- Step 5: partnerExpectation (if 5 steps completed)

Success Response (200) - Basic Info Step

```
{
    "status": true,
    "message": "User data step retrieved",
    "data": {
        "current_step": "basicInfo",
        "data": {
            "religions": [
                {
                    "id": 1,
                    "name": "Islam"
                },
                {
                    "id": 2,
                    "name": "Christianity"
                }
            ],
            "maritalStatuses": [
                {
                    "id": 1,
                    "title": "Single",
                    "children": []
                },
                {
                    "id": 2,
                    "title": "Divorced",
                    "children": []
                }
            ],
            "countries": [
                {
                    "country": "United States",
                    "dial_code": "+1",
                    "code": "US"
                },
                {
                    "country": "United Kingdom",
                    "dial_code": "+44",
                    "code": "GB"
                }
            ],
            "user": {
                "id": 1,
                "profile_id": "12345678",
                "firstname": "John",
                "lastname": "Doe",
                "email": "john.doe@example.com"
            },
            "mobileCode": "+1"
        }
    }
}
```

Submit User Data Step

POST `/v1/user/user-data-submit/{step}`

Submits data for a specific profile completion step. Supports step navigation and validation.

Headers

```
Authorization: Bearer {your_token}
```

URL Parameters

- `step` (required): Step name - one of: `basicInfo`, `familyInfo`, `educationInfo`, `careerInfo`, `physicalAttributeInfo`, `partnerExpectation`

Request Body Examples with Validation Rules

Basic Info Step

```json
{
    "birth_date": "1990-01-01",
    "religion": "Islam",
    "gender": "m",
    "profession": "Software Engineer",
    "financial_condition": "Good",
    "smoking_status": "0",
    "drinking_status": "0",
    "marital_status": "Single",
    "languages": ["English", "Spanish"],
    "country_code": "US",
    "country": "United States",
    "mobile_code": "+1",
    "username": "john_doe",
    "mobile": "1234567890",
    "pre_city": "Los Angeles",
    "per_city": "Los Angeles"
}
```

Validation Rules:

- `birth_date`: required, date format Y-m-d, must be before today
- `religion`: required, must exist in religion_infos table
- `gender`: required, must be "m" or "f"
- `profession`: required, string
- `financial_condition`: required, string
- `smoking_status`: required, must be "0" or "1"
- `drinking_status`: required, must be "0" or "1"
- `marital_status`: required, must exist in marital_statuses table
- `languages`: required, array of strings
- `country_code`: required, must be valid country code
- `country`: required, must be valid country name
- `mobile_code`: required, must be valid dial code
- `username`: required, minimum 6 characters, unique, alphanumeric and underscore only
- `mobile`: required, numeric only, unique with mobile_code
- `pre_city`: required, string
- `per_city`: required, string

Family Info Step

```
{
    "father_name": "John Doe Sr.",
    "mother_name": "Jane Doe",
    "family_type": "Nuclear",
    "family_status": "Middle Class",
    "family_values": "Traditional",
    "family_income": "50000-75000",
    "siblings": "2",
    "family_location": "Los Angeles"
}
```

Validation Rules:

- `father_name`: required, string
- `mother_name`: required, string
- `family_type`: required, string
- `family_status`: required, string
- `family_values`: required, string
- `family_income`: required, string
- `siblings`: required, string
- `family_location`: required, string

Education Info Step

```
{
    "education_level": "Bachelor's Degree",
    "institution": "University of California",
    "degree": "Computer Science",
    "passing_year": "2012",
    "result": "3.8 GPA"
}
```

Validation Rules:

- `education_level`: required, string
- `institution`: required, string
- `degree`: required, string
- `passing_year`: required, string
- `result`: required, string

Career Info Step

```
{
    "occupation": "Software Engineer",
    "company": "Tech Corp",
    "income": "75000-100000",
    "work_location": "San Francisco"
}
```

Validation Rules:

- `occupation`: required, string
- `company`: required, string
- `income`: required, string
- `work_location`: required, string

Physical Attribute Info Step

```
{
    "height": "5'10\"",
    "weight": "70 kg",
    "body_type": "Athletic",
    "complexion": "Fair",
    "blood_group": "O+",
    "disability": "None"
}
```

Validation Rules:

- `height`: required, string
- `weight`: required, string
- `body_type`: required, string
- `complexion`: required, string
- `blood_group`: required, must exist in blood_groups table
- `disability`: required, string

Partner Expectation Step

```
{
    "preferred_age_min": "25",
    "preferred_age_max": "35",
    "preferred_height_min": "5'5\"",
    "preferred_height_max": "6'0\"",
    "preferred_marital_status": "Single",
    "preferred_religion": "Islam",
    "preferred_education": "Bachelor's Degree",
    "preferred_profession": "Professional",
    "preferred_country": "United States"
}
```

Validation Rules:

- `preferred_age_min`: required, string
- `preferred_age_max`: required, string
- `preferred_height_min`: required, string
- `preferred_height_max`: required, string
- `preferred_marital_status`: required, string
- `preferred_religion`: required, string
- `preferred_education`: required, string
- `preferred_profession`: required, string
- `preferred_country`: required, string

Step Navigation

You can navigate back to previous steps by including a `back_to` parameter:

```
{
    "back_to": "basicInfo"
}
```

This will:

1. Remove the current step from completed/skipped arrays
2. Allow you to re-submit data for the specified step

Success Response (200)

```
{
    "status": true,
    "message": "Step completed successfully",
    "data": null
}
```

Error Responses

- **422 Unprocessable Entity**: Validation errors
- **404 Not Found**: Invalid step name
- **400 Bad Request**: Invalid back_to field

---

4. Dashboard

Get Dashboard Data

GET `/v1/user/dashboard`

Retrieves comprehensive user dashboard information including profile data, interests, and statistics. Requires profile completion.

Headers

```
Authorization: Bearer {your_token}
```

Middleware Requirements

- `auth:sanctum`: User must be authenticated
- `check.status`: User account must be active
- `lastActivity`: Updates user's last activity
- `registration.complete`: User profile must be complete

Success Response (200)

```
{
    "status": true,
    "message": "User dashboard data retrieved successfully",
    "data": {
        "id": 1,
        "profile_id": "12345678",
        "firstname": "John",
        "lastname": "Doe",
        "email": "john.doe@example.com",
        "username": "john_doe",
        "gender": "m",
        "profile_complete": 1,
        "limitation": {
            "id": 1,
            "package_id": 1,
            "interest_express_limit": 10,
            "contact_view_limit": 5,
            "image_upload_limit": 10,
            "validity_period": 30,
            "expire_date": "2024-02-01T00:00:00.000000Z",
            "package": {
                "id": 1,
                "name": "Basic Package",
                "price": 0,
                "validity_period": 30
            }
        },
        "interests": [
            {
                "id": 1,
                "user_id": 1,
                "profile_id": 2,
                "created_at": "2024-01-01T00:00:00.000000Z",
                "profile": {
                    "id": 2,
                    "basicInfo": {
                        "id": 2,
                        "user_id": 2,
                        "gender": "f",
                        "profession": "Doctor",
                        "birth_date": "1992-05-15"
                    }
                }
            }
        ],
        "interestRequests": [
            {
                "id": 1,
                "user_id": 2,
                "profile_id": 1,
                "created_at": "2024-01-01T00:00:00.000000Z",
                "user": {
                    "id": 2,
                    "basicInfo": {
                        "id": 2,
                        "user_id": 2,
                        "gender": "f",
                        "profession": "Doctor",
                        "birth_date": "1992-05-15"
                    }
                }
            }
        ],
        "total_images": 5,
        "totalShortlisted": 3,
        "interestSent": 8,
        "totalInterestRequests": 12
    }
}
```

Dashboard Statistics

- `total_images`: Number of images in user's gallery
- `totalShortlisted`: Number of profiles shortlisted by user
- `interestSent`: Number of interests sent by user
- `totalInterestRequests`: Number of interest requests received

Error Responses

- **401 Unauthorized**: User not authenticated
- **403 Forbidden**: Profile incomplete or account inactive

5. Check User Existence

Check User

POST `/check-user`

Checks if a user with the provided email, mobile, or username already exists. Useful for form validation.

Request Body

```
{
    "email": "john.doe@example.com"
}
```

OR

```
{
    "mobile": "1234567890",
    "mobile_code": "+1"
}
```

OR

```
{
    "username": "john_doe"
}
```

Validation Rules

- Only one field should be provided at a time
- `email`: valid email format
- `mobile`: numeric only
- `mobile_code`: valid dial code (required if mobile provided)
- `username`: alphanumeric and underscore only

Success Response (200)

```
{
    "status": true,
    "message": "User check completed",
    "data": {
        "data": true,
        "type": "email",
        "field": "Email"
    }
}
```

Response Fields

- `data`: Boolean indicating if user exists
- `type`: Type of field checked ("email", "mobile", or "username")
- `field`: Human-readable field name

Error Codes and Messages

Status Code	Description	Common Causes
200	Success	Request completed successfully
400	Bad Request	Invalid captcha, registration disabled, invalid back_to field
401	Unauthorized	Invalid credentials, missing or invalid token
403	Forbidden	Registration disabled, unauthorized access, profile incomplete
404	Not Found	Invalid step name, resource not found
422	Unprocessable Entity	Validation errors, missing required fields

Common Validation Errors

- **Email already exists**: Email address is already registered
- **Username can contain only small letters, numbers and underscore**: Invalid username format
- **The mobile number already exists**: Mobile number is already registered
- **Invalid step**: Step name not in allowed list
- **Profile already complete**: User has already completed profile

---

Technical Notes

Profile Completion Process

1. **Registration**: User creates account with basic info
2. **Step-by-step completion**: User completes 6 profile steps
3. **Dashboard access**: Only available after profile completion

Step Order

1. `basicInfo`: Personal and contact information
2. `familyInfo`: Family background
3. `educationInfo`: Educational background
4. `careerInfo`: Professional information

5. `physicalAttributeInfo`: Physical characteristics
6. `partnerExpectation`: Partner preferences

Token Management

- Tokens are created using Laravel Sanctum
- Tokens are automatically generated on login/registration
- Tokens are revoked on logout
- Token format: `{id}|{random_string}`

Data Validation

- All endpoints include comprehensive server-side validation
- Validation rules are enforced at the controller level
- Detailed error messages are returned for validation failures

Security Features

- CSRF protection (where applicable)
- Rate limiting (configurable)
- Captcha verification (optional)
- Password complexity requirements (configurable)
- IP-based login tracking

Rate Limiting

API endpoints may be subject to rate limiting based on server configuration. Check with your system administrator for specific limits.

Captcha Integration

Some endpoints may require captcha verification depending on system settings. The captcha value should be included in the request body when required.

Profile Completion Requirements

Users must complete all 6 profile steps before accessing the dashboard and other protected features. The system tracks completion progress using `completed_step` and `skipped_step` arrays in the user model.