Rate-It User API - Technical Documentation

For Developers & QA Teams

Version: 1.0
Last Updated: 2026-01-19
Base URL: `{base_url}/api/v1/user`

Table of Contents

---

1. System Overview

Architecture

Rate-It is a Laravel 12 backend API following a **modular architecture** under `app/Modules/User/`. Each module contains:

- **Services**: Business logic layer
- **Controllers**: HTTP request/response handling
- **Resources**: API response transformation
- **FormRequests**: Input validation
- **Routes**: API endpoint definitions
- **Models**: Database entity representation

Module Breakdown

01. Home Module

- **Purpose**: Landing screen content (banners, featured categories, search)
- **Dependencies**: Categories, Brands
- **Status**: Implemented
- **Endpoints**: Home banners, categories, search

02. Onboarding Module

- **Purpose**: New user onboarding flow
- **Dependencies**: None
- **Status**: Implemented

- **Endpoints**: Onboarding screens/steps

03. Auth Module

- **Purpose**: Authentication & user management
- **Dependencies**: OTP codes, Genders, Nationalities
- **Status**: Fully implemented
- **Key Features**:
    - Phone-based registration with OTP verification
    - Email/phone + password login
    - Forgot password flow (send OTP → verify → reset)
    - Phone verification flow
    - Sanctum Bearer token authentication

04. Lookups Module

- **Purpose**: Static data lookups (genders, nationalities)
- **Dependencies**: None
- **Status**: Implemented
- **Endpoints**: Genders list, Nationalities list

05. Categories Module

- **Purpose**: Browse categories and subcategories
- **Dependencies**: None
- **Status**: Implemented
- **Key Features**:
    - Category listing with localization (en/ar)
    - Subcategories per category
    - Search across categories/subcategories/brands

06. Brands & Places Module

- **Purpose**: Brand and place discovery
- **Dependencies**: Categories (subcategories), Branches
- **Status**: Implemented
- **Key Features**:
    - Brand details with logo
    - Place details (associated with brand and subcategory)
    - Place listing per brand
    - Place reviews listing (paginated)

07. Reviews Module

- **Purpose**: QR-based review submission system
- **Dependencies**: Branches, Rating Criteria, Points, Notifications
- **Status**: Fully implemented
- **Key Features**:

    - QR code scanning → session token generation
    - Branch-specific questions (rating criteria)
    - Multi-criteria answer validation (RATING/YES_NO/MULTIPLE_CHOICE)
    - Photo uploads (max 3)
    - Review cooldown enforcement (per branch)
    - Points awarding integration
    - My reviews listing
    - Review details

08. Notifications Module

- **Purpose**: User notification management
- **Dependencies**: None
- **Status**: Implemented
- **Key Features**:
    - Notification listing (paginated, with unread filter)
    - Unread count
    - Mark single notification as read
    - Mark all as read
    - Delete single notification
    - Clear all notifications

09. Points (Loyalty) Module

- **Purpose**: Loyalty points system
- **Dependencies**: Points Settings, Reviews, Invites
- **Status**: Fully implemented
- **Key Features**:
    - Points balance calculation (unexpired transactions only)
    - Points earning (review submission, invites)
    - Points expiration (brand-specific or global)
    - Points redemption (vouchers)
    - Transaction history (paginated)
    - Idempotent point awarding (prevents duplicates)

10. Invites (Referral) Module

- **Purpose**: Referral program
- **Dependencies**: Points, Auth
- **Status**: Fully implemented
- **Key Features**:
    - Check phone installation status
    - Create invites for non-registered phones
    - Automatic reward on invite completion (new user registration)
    - Invites listing with status tracking

11. Profile Module

- **Purpose**: User profile management
- **Dependencies**: Phone Change Requests
- **Status**: Implemented
- **Key Features**:
  - Get user profile
  - Update profile (name, avatar upload)
  - Phone change flow (send OTP → verify OTP)

12. Subscriptions Module

- **Purpose**: Manage subscription plans, user subscriptions, trials, and auto-renew settings
- **Dependencies**: `subscriptions` table (legacy fields retained), new `subscription_plans` and `subscription_transactions` tables; NotificationService (optional for events), Scheduler for renewals
- **Status**: Implemented (DB migrations, models, service, controller, resources, routes, seeder). Payment provider integration (Stripe/Apple/Google) is a planned next step.
- **Key Features**:
  - Bilingual plan metadata (`name_en`, `name_ar`, `description_en`, `description_ar`)
  - Trial support (seeded with `trial_days = 180` → ~6 months free)
  - Auto-renew toggle (default ON) with cancel/resume endpoints; cancellation sets `auto_renew=false` and `canceled_at` but does not immediately revoke access
  - Manual `provider` checkout placeholder (records a pending transaction and subscription) with structured extension points for payment provider integration
  - Transaction history tracking in `subscription_transactions`

Subscriptions API Reference (Detailed)

This section documents database schema changes, models, service behaviour, and the Subscriptions endpoints.

Database (new/updated tables)

- `subscription_plans` (new)
  - id, name_en, name_ar, description_en, description_ar
  - price_cents (integer), currency (string), interval (enum: monthly, annual)
  - trial_days (integer), is_best_value (boolean), is_active (boolean)
  - metadata JSON, timestamps
- `subscriptions` (updated)
  - existing legacy fields kept (status, started_at, free_until, paid_until)
  - added: subscription_plan_id (fk), subscription_status (enum: trialing, active, past_due, canceled, expired)
  - auto_renew (boolean), canceled_at (nullable), provider (string), provider_subscription_id (string), provider_transaction_id (string), meta JSON
- `subscription_transactions` (new)

    - id, subscription_id (fk), user_id, amount_cents, currency, provider, provider_transaction_id, status (pending, success, failed), meta JSON, created_at

Models

- `SubscriptionPlan` — Eloquent model for `subscription_plans` with localized accessors
- `Subscription` — updated relations: `plan()` and `transactions()`
- `SubscriptionTransaction` — transaction records

Business Rules

- Trial: `trial_days` seeded as 180 (6 months). When checking out, if a plan has `trial_days>0`, subscription is created with `subscription_status = 'trialing'` and `free_until = now + trial_days`
- Auto-renew: default `true`. Cancelling auto-renew sets `auto_renew=false` and `canceled_at` but access remains until `paid_until` or `free_until`.
- Checkout flow: currently supports `provider = 'manual'` which creates a pending transaction and records the subscription. Payment provider integration must update transactions via webhooks and set `subscription_status`/`paid_until` accordingly.
- Renewals & expirations: a scheduled job should process upcoming renewals and handle provider interactions. Not implemented here.

Endpoints

Base: `/api/v1/user/subscriptions`

1. GET `/plans` — Public
    - Description: List available subscription plans (localized by `X-Lang`)
    - Response: array of plan objects (id, name, description, price_cents, currency, interval, trial_days, is_best_value)
2. GET `/me` — Auth required
    - Description: Get the authenticated user's current subscription (if any)
    - Response: subscription object with plan, status, free_until, paid_until, auto_renew
3. POST `/checkout` — Auth required
    - Description: Start a subscription checkout. Request body: `{ "plan_id": <id>, "provider": "manual" }`
    - Behaviour: Creates `subscription` and a pending `subscription_transaction`. If the plan has trial days, sets `subscription_status` to `trialing` and `free_until` accordingly.
    - Response: subscription object and transaction meta
4. POST `/cancel-auto-renew` — Auth required
    - Description: Disable auto-renew for the user's current subscription. Sets `auto_renew=false` and `canceled_at`.
    - Response: success true

5. POST `/resume-auto-renew` — Auth required
    - Description: Re-enable auto-renew for the subscription (`auto_renew=true`, clears `canceled_at`)
    - Response: success true
6. GET `/history` — Auth required
    - Description: Paginated transaction/subscription history for the user
    - Response: paginated list of subscription transactions and past subscriptions

Request/Response Examples

Example: GET Plans

Response (200):

```
{
  "success": true,
  "message": "Plans retrieved",
  "data": [
    {
      "id": 1,
      "name": "Monthly",
      "description": "Monthly plan",
      "price_cents": 9900,
      "currency": "EGP",
      "interval": "monthly",
      "trial_days": 180,
      "is_best_value": false
    }
  ],
  "meta": null
}
```

Example: POST Checkout (manual)

Request:

```
{ "plan_id": 1, "provider": "manual" }
```

Response (200):

```
{
  "success": true,
  "message": "Subscription created",
  "data": {
    "subscription": { /* subscription fields */ },
```

```
    "transaction": { /* pending transaction */ }
  },
  "meta": null
}
```

Postman

The updated Postman collection includes a `12 - Subscriptions` folder with these requests:

- `GET Plans`
- `GET My Subscription`
- `POST Checkout` (manual)
- `POST Cancel Auto-Renew`
- `POST Resume Auto-Renew`
- `GET History`

Use the collection variables to store `subscription_plan_id` and `user_token` during tests.

Developer Notes / Next Steps

- Integrate a payment provider (Stripe recommended) and implement webhook handlers to mark transactions `success` and set `paid_until`.
- Implement a scheduler to process renewals and expire subscriptions.
- Add notifications on subscription activation, renewal, cancellation, and expiry via existing `NotificationService`.
- Add unit/integration tests for checkout, trial, cancel/resume flows.

Local Setup (migrate & seed)

```
php artisan migrate
php artisan db:seed --class=SubscriptionPlansSeeder
```

---

2. API Conventions

Request Headers

All API requests **MUST** include:

```
Accept: application/json
X-Lang: en
```

Protected endpoints additionally require:

```
Authorization: Bearer {user_token}
```

Supported Languages: `en` (English), `ar` (Arabic)

Response Format

All API responses follow this **unified wrapper structure**:

```
{
  "success": true,
  "message": "Success message or error message",
  "data": { /* Response payload or null */ },
  "meta": { /* Optional metadata (pagination, counts, etc.) */ }
}
```

Success Response Example

```
{
  "success": true,
  "message": "Review created successfully",
  "data": {
    "id": 123,
    "user_id": 1,
    "branch_id": 5,
    "overall_rating": 4.5,
    "comment": "Great service",
    "review_score": 4.2,
    "created_at": "2026-01-19T10:30:00Z"
  },
  "meta": {
    "points_awarded": 50,
    "points_balance": 250
  }
}
```

Error Response Example (Validation)

```
{
  "success": false,
  "message": "Validation failed",
  "data": {
    "phone": ["The phone field is required."],
    "password": ["The password must be at least 8 characters."]
  },
  "meta": null
}
```

Error Response Example (Business Logic)

```
{
  "success": false,
  "message": "QR session has expired. Please scan the QR code again.",
  "data": null,
  "meta": null
}
```

HTTP Status Codes

Code	Meaning	Usage
200	OK	Successful request
201	Created	Resource created successfully
400	Bad Request	Malformed request or business rule violation
401	Unauthorized	Missing or invalid authentication token
403	Forbidden	User doesn't have permission
404	Not Found	Resource doesn't exist
422	Unprocessable Entity	Validation errors
500	Internal Server Error	Server error (logged, no stack trace returned)

Error Handling

- **NO STACK TRACES** are exposed in production responses
- All errors are logged server-side with full context
- Error messages are localized based on `X-Lang` header
- Validation errors return field-level error arrays in `data`

Pagination

Endpoints returning lists support pagination via query parameters:

```
GET /api/v1/user/notifications?page=1&per_page=20
```

Paginated response meta:

```
{
  "data": [ /* items */ ],
  "meta": {
    "current_page": 1,
    "last_page": 5,
```

```
    "per_page": 20,
    "total": 100,
    "from": 1,
    "to": 20
  }
}
```

File Uploads

- Use `multipart/form-data` encoding
- **DO NOT** set `Content-Type` header manually (let browser/client set boundaries)
- Maximum file size: 5MB (configurable)
- Supported image formats: jpg, jpeg, png, gif, webp

---

3. Full Endpoint Reference

01. Auth Module

Register

```
POST /api/v1/user/auth/register
Content-Type: application/json
Accept: application/json
X-Lang: en
```

Request Body:

```
{
  "full_name": "Ahmed Ali",
  "phone": "+201000000000",
  "email": "ahmed@example.com",
  "birth_date": "1995-05-15",
  "gender_id": 1,
  "nationality_id": 1,
  "password": "SecurePass123!",
  "password_confirmation": "SecurePass123!",
  "invited_by_phone": "+201111111111"
}
```

Response (200):

```
{
  "success": true,
  "message": "Registration successful",
  "data": {
```

```
    "user": {
      "id": 1,
      "full_name": "Ahmed Ali",
      "phone": "+201000000000",
      "email": "ahmed@example.com",
      "is_phone_verified": false,
      "avatar_url": null
    },
    "token": "1|abc123xyz..."
  },
  "meta": null
}
```

Validation Rules:

- `full_name`: required, string, max 255
- `phone`: required, unique, E164 format
- `email`: required, email, unique
- `birth_date`: required, date, before today
- `gender_id`: required, exists in genders table
- `nationality_id`: required, exists in nationalities table
- `password`: required, min 8, confirmed
- `invited_by_phone`: optional, E164 format

Business Logic:

- If `invited_by_phone` is provided and matches a pending invite, inviter receives reward points

---

Login

```
POST /api/v1/user/auth/login
Content-Type: application/json
```

Request Body:

```
{
  "phone": "+201000000000",
  "password": "SecurePass123!"
}
```

Response (200):

```
{
  "success": true,
  "message": "Login successful",
```

```
  "data": {
    "user": { /* user object */ },
    "token": "2|xyz789abc..."
  },
  "meta": null
}
```

Error (401):

```
{
  "success": false,
  "message": "Invalid credentials",
  "data": null,
  "meta": null
}
```

Forgot Password - Send OTP

```
POST /api/v1/user/auth/forgot-password/send-otp
Content-Type: application/json
```

Request Body:

```
{
  "phone": "+201000000000"
}
```

Response (200):

```
{
  "success": true,
  "message": "OTP sent successfully",
  "data": null,
  "meta": {
    "otp_code": "1234"
  }
}
```

Note: `otp_code` in `meta` is only for **dev/staging** environments. Not present in production.

Forgot Password - Verify OTP

```
POST /api/v1/user/auth/forgot-password/verify-otp
Content-Type: application/json
```

Request Body:

```
{
  "phone": "+201000000000",
  "otp": "1234"
}
```

Response (200):

```
{
  "success": true,
  "message": "OTP verified",
  "data": {
    "reset_token": "r-abc123xyz"
  },
  "meta": null
}
```

Forgot Password - Reset

```
POST /api/v1/user/auth/forgot-password/reset
Content-Type: application/json
```

Request Body:

```
{
  "phone": "+201000000000",
  "reset_token": "r-abc123xyz",
  "new_password": "NewPass123!",
  "new_password_confirmation": "NewPass123!"
}
```

Response (200):

```
{
  "success": true,
  "message": "Password reset successfully",
```

```
  "data": null,
  "meta": null
}
```

Phone - Send OTP

```
POST /api/v1/user/auth/phone/send-otp
Content-Type: application/json
```

Request Body:

```
{
  "phone": "+201000000000"
}
```

Response (200):

```
{
  "success": true,
  "message": "OTP sent",
  "data": null,
  "meta": {
    "otp_code": "5678"
  }
}
```

Phone - Verify OTP

```
POST /api/v1/user/auth/phone/verify-otp
Content-Type: application/json
```

Request Body:

```
{
  "phone": "+201000000000",
  "otp": "5678"
}
```

Response (200):

```
{
  "success": true,
  "message": "Phone verified",
  "data": {
    "user": {
      "id": 1,
      "phone": "+201000000000",
      "is_phone_verified": true
    },
    "token": "3|newtoken..."
  },
  "meta": null
}
```

Me (Get Current User)

```
GET /api/v1/user/auth/me
Authorization: Bearer {user_token}
```

Response (200):


```
{
  "success": true,
  "message": "User retrieved successfully",
  "data": {
    "id": 1,
    "full_name": "Ahmed Ali",
    "phone": "+201000000000",
    "email": "ahmed@example.com",
    "is_phone_verified": true,
    "avatar_url": "https://example.com/storage/avatars/123.jpg",
    "gender": { "id": 1, "code": "MALE", "name": "Male" },
    "nationality": { "id": 1, "country_code": "EG", "name": "Egypt", "flag_url":
"..." }
  },
  "meta": null
}
```

Logout

```
POST /api/v1/user/auth/logout
Authorization: Bearer {user_token}
```

Response (200):

```
{
  "success": true,
  "message": "Logged out successfully",
  "data": null,
  "meta": null
}
```

---

02. Home Module

Get Home Data

```
GET /api/v1/user/home
Accept: application/json
X-Lang: en
```

Response (200):

```
{
  "success": true,
  "message": "Home data retrieved",
  "data": {
    "banners": [
      {
        "id": 1,
        "image_url": "https://example.com/banners/1.jpg",
        "target_type": "brand",
        "target_id": 5
      }
    ],
    "featured_categories": [
      { "id": 1, "name": "Restaurants", "logo_url": "..." }
    ]
  },
  "meta": null
}
```

Search

```
GET /api/v1/user/home/search?q=coffee&limit=20
Accept: application/json
X-Lang: en
```

Response (200):

```
{
  "success": true,
  "message": "Search results",
  "data": {
    "query": "coffee",
    "results": {
      "categories": [ /* matching categories */ ],
      "subcategories": [ /* matching subcategories */ ],
      "brands": [ /* matching brands */ ],
      "places": [ /* matching places */ ]
    }
  },
  "meta": null
}
```

---

03. Lookups Module

Get Genders

```
GET /api/v1/user/lookups/genders
```

Response (200):

```
{
  "success": true,
  "message": "Genders retrieved",
  "data": [
    { "id": 1, "code": "MALE", "name": "Male" },
    { "id": 2, "code": "FEMALE", "name": "Female" }
  ],
  "meta": null
}
```

Get Nationalities

```
GET /api/v1/user/lookups/nationalities
```

Response (200):

```
{
  "success": true,
  "message": "Nationalities retrieved",
  "data": [
    {
      "id": 1,
      "country_code": "EG",
      "name": "Egypt",
      "flag_url": "https://flagsapi.com/EG/flat/64.png"
    },
    {
      "id": 2,
      "country_code": "SA",
      "name": "Saudi Arabia",
      "flag_url": "https://flagsapi.com/SA/flat/64.png"
    }
  ],
  "meta": null
}
```


---

04. Categories Module

Get Categories

```
GET /api/v1/user/categories
X-Lang: en
```

Response (200):


```
{
  "success": true,
  "message": "Categories retrieved",
  "data": [
    {
      "id": 1,
      "name": "Food & Beverages",
      "logo_url": "https://example.com/categories/1.jpg",
      "is_active": true
    },
    {
      "id": 2,
      "name": "Healthcare",
      "logo_url": "https://example.com/categories/2.jpg",
      "is_active": true
    }
  ],
```

```
  "meta": null
}
```

Get Subcategories

```
GET /api/v1/user/categories/{category_id}/subcategories
X-Lang: en
```

Response (200):

```
{
  "success": true,
  "message": "Subcategories retrieved",
  "data": [
    { "id": 1, "name": "Restaurants", "image_url": "..." },
    { "id": 2, "name": "Cafes", "image_url": "..." }
  ],
  "meta": null
}
```

Search Categories

```
GET /api/v1/user/categories/search?
q=clinic&category_id=2&types=subcategories&limit=10
```

Query Parameters:

- `q`: Search query (required)
- `category_id`: Filter by category (optional)
- `types`: Comma-separated list (categories, subcategories, brands, places) - optional
- `limit`: Max results (default 20)

Response (200):

```
{
  "success": true,
  "message": "Search results",
  "data": {
    "query": "clinic",
    "results": [
      { "type": "subcategory", "id": 5, "name": "Medical Clinics",
"category_name": "Healthcare" }
```

```
    ]
  },
  "meta": null
}
```

---

05. Brands & Places Module

Get Brand Details

```
GET /api/v1/user/brands/{brand_id}
X-Lang: en
```

Response (200):

```
{
  "success": true,
  "message": "Brand retrieved",
  "data": {
    "id": 1,
    "name": "Starbucks",
    "logo_url": "https://example.com/brands/starbucks.jpg",
    "points_expiry_days": 365,
    "places_count": 15
  },
  "meta": null
}
```

Get Brand Places

```
GET /api/v1/user/brands/{brand_id}/places
X-Lang: en
```

Response (200):

```
{
  "success": true,
  "message": "Places retrieved",
  "data": [
    {
      "id": 1,
      "name": "Starbucks City Center",
      "city": "Cairo",
```

```
      "area": "Downtown",
      "description": "Coffee shop in the heart of Cairo",
      "is_featured": true
    }
  ],
  "meta": null
}
```

Get Place Details

```
GET /api/v1/user/places/{place_id}
X-Lang: en
```

Response (200):

```
{
  "success": true,
  "message": "Place retrieved",
  "data": {
    "id": 1,
    "name": "Starbucks City Center",
    "brand": { "id": 1, "name": "Starbucks", "logo_url": "..." },
    "subcategory": { "id": 2, "name": "Cafes" },
    "description": "Premium coffee experience",
    "city": "Cairo",
    "area": "Downtown",
    "branches": [
      {
        "id": 1,
        "name": "Main Branch",
        "address": "123 Main St, Cairo",
        "lat": 30.0444,
        "lng": 31.2357,
        "working_hours": { "saturday": "08:00-22:00" }
      }
    ],
    "reviews_count": 125,
    "average_rating": 4.3
  },
  "meta": null
}
```

Get Place Reviews

```
GET /api/v1/user/places/{place_id}/reviews?page=1
X-Lang: en
```

Response (200):


```
{
  "success": true,
  "message": "Reviews retrieved",
  "data": [
    {
      "id": 1,
      "user": { "full_name": "Ahmed Ali", "avatar_url": "..." },
      "overall_rating": 4.5,
      "review_score": 4.2,
      "comment": "Excellent service and quality!",
      "photos": [
        { "url": "https://example.com/reviews/1_photo1.jpg" }
      ],
      "created_at": "2026-01-15T10:30:00Z"
    }
  ],
  "meta": {
    "current_page": 1,
    "last_page": 5,
    "per_page": 20,
    "total": 100
  }
}
```


---

06. Reviews Module

Scan QR Code

```
POST /api/v1/user/reviews/scan-qr
Authorization: Bearer {user_token}
Content-Type: application/json
```

Request Body:


```
{
  "qr_code_value": "BRANCH_123_QR_XYZ789"
}
```


Response (200):

```
{
  "success": true,
  "message": "QR scanned successfully",
  "data": {
    "session_token": "sess_abc123xyz",
    "branch": {
      "id": 1,
      "name": "Main Branch",
      "place_name": "Starbucks City Center",
      "brand_name": "Starbucks"
    },
    "expires_at": "2026-01-19T11:00:00Z"
  },
  "meta": null
}
```


Error (400):


```
{
  "success": false,
  "message": "Invalid QR code",
  "data": null,
  "meta": null
}
```


Business Logic:

- Session expires after 30 minutes (configurable)
- Session token is single-use (consumed after review submission)

Get Branch Questions

```
GET /api/v1/user/reviews/branch/{branch_id}/questions
Authorization: Bearer {user_token}
X-Lang: en
```

Response (200):


```
{
  "success": true,
  "message": "Questions retrieved",
  "data": [
    {
      "id": 1,
      "type": "RATING",
```

```
      "question_en": "How would you rate the food quality?",
      "question_ar": "كيف تقيم جودة الطعام؟",
      "is_required": true,
      "choices": null
    },
    {
      "id": 2,
      "type": "YES_NO",
      "question_en": "Would you recommend this place?",
      "question_ar": "هل توصي بهذا المكان؟",
      "is_required": true,
      "choices": null
    },
    {
      "id": 3,
      "type": "MULTIPLE_CHOICE",
      "question_en": "What did you order?",
      "question_ar": "ماذا طلبت؟",
      "is_required": false,
      "choices": [
        { "id": 1, "text_en": "Coffee", "text_ar": "قهوة" },
        { "id": 2, "text_en": "Tea", "text_ar": "شاي" },
        { "id": 3, "text_en": "Food", "text_ar": "طعام" }
      ]
    }
  ],
  "meta": null
}
```


Rating Criteria Types:

- **RATING**: 1-5 scale (answer field: `rating_value`)
- **YES_NO**: Boolean question (answer field: `yes_no_value` - 1 for Yes, 0 for No)
- **MULTIPLE_CHOICE**: Select from predefined choices (answer field: `choice_id`)

Create Review

```
POST /api/v1/user/reviews
Authorization: Bearer {user_token}
Content-Type: multipart/form-data
```

Request Body (multipart):

```
session_token: sess_abc123xyz
overall_rating: 5
comment: Excellent experience!
answers[0][criteria_id]: 1
answers[0][rating_value]: 5
```

```
answers[1][criteria_id]: 2
answers[1][yes_no_value]: 1
answers[2][criteria_id]: 3
answers[2][choice_id]: 1
photos[]: (file)
photos[]: (file)
```

Response (200):

```
{
  "success": true,
  "message": "Review submitted successfully",
  "data": {
    "id": 123,
    "user_id": 1,
    "branch_id": 5,
    "place_id": 2,
    "overall_rating": 5,
    "comment": "Excellent experience!",
    "review_score": 4.67,
    "photos": [
      { "url": "https://example.com/storage/reviews/123/photo1.jpg" }
    ],
    "created_at": "2026-01-19T10:45:00Z"
  },
  "meta": {
    "points_awarded": 50,
    "points_balance": 350
  }
}
```

Validation Rules:

- `session_token`: required, valid, not expired, not consumed
- `overall_rating`: required, decimal between 1.0 and 5.0
- `comment`: optional, string, max 1000 characters
- `answers`: array of answer objects, each must match criteria type validation
- `photos`: optional, max 3 files, max 5MB each, image formats only

Answer Validation by Type:

- **RATING**: `rating_value` must be integer 1-5
- **YES_NO**: `yes_no_value` must be boolean (1/0, true/false, "1"/"0" auto-normalized)
- **MULTIPLE_CHOICE**: `choice_id` must exist and belong to the criteria

Error (400) - Cooldown:

```
{
  "success": false,
```

```
  "message": "You must wait 7 days before submitting another review for this
branch.",
  "data": {
    "last_review_date": "2026-01-12T10:30:00Z",
    "cooldown_days": 7,
    "next_allowed_date": "2026-01-19T10:30:00Z"
  },
  "meta": null
}
```

Error (400) - Session Expired:

```
{
  "success": false,
  "message": "QR session has expired. Please scan the QR code again.",
  "data": null,
  "meta": null
}
```

Error (422) - Validation:

```
{
  "success": false,
  "message": "Validation failed",
  "data": {
    "answers.0.rating_value": ["Rating value must be between 1 and 5"],
    "photos": ["You can upload a maximum of 3 photos"]
  },
  "meta": null
}
```

Business Logic:

1. Validate session token (exists, not expired, not consumed, belongs to user)
2. Check cooldown period (based on `branches.review_cooldown_days`)
3. Resolve allowed criteria for branch (via subcategory)
4. Validate each answer against its criteria type
5. Normalize multipart form data (string "1"/"0" → int/bool)
6. Upload photos (max 3, stored in `storage/reviews/{review_id}/`)
7. Mark session as consumed (`consumed_at` timestamp)
8. Calculate `review_score` (average of rating_value answers)
9. Award points via PointsService (idempotent, checks for duplicate)
10. Return review + points meta

Get My Reviews

```
GET /api/v1/user/reviews/me/list?page=1
Authorization: Bearer {user_token}
```

Response (200):


```
{
  "success": true,
  "message": "Reviews retrieved",
  "data": [
    {
      "id": 123,
      "branch": { "name": "Main Branch" },
      "place": { "name": "Starbucks City Center" },
      "brand": { "name": "Starbucks" },
      "overall_rating": 5,
      "review_score": 4.67,
      "comment": "Excellent experience!",
      "photos_count": 2,
      "created_at": "2026-01-19T10:45:00Z"
    }
  ],
  "meta": {
    "current_page": 1,
    "per_page": 20,
    "total": 15
  }
}
```

Get Review Details

```
GET /api/v1/user/reviews/{review_id}
Authorization: Bearer {user_token}
```

Response (200):


```
{
  "success": true,
  "message": "Review retrieved",
  "data": {
    "id": 123,
    "user": { "full_name": "Ahmed Ali", "avatar_url": "..." },
    "branch": { "id": 5, "name": "Main Branch" },
    "place": { "id": 2, "name": "Starbucks City Center" },
    "brand": { "id": 1, "name": "Starbucks" },
    "overall_rating": 5,
```

```
    "review_score": 4.67,
    "comment": "Excellent experience!",
    "photos": [
      { "url": "https://example.com/storage/reviews/123/photo1.jpg" },
      { "url": "https://example.com/storage/reviews/123/photo2.jpg" }
    ],
    "answers": [
      {
        "criteria": { "question": "How would you rate the food quality?" },
        "rating_value": 5
      },
      {
        "criteria": { "question": "Would you recommend this place?" },
        "yes_no_value": true
      }
    ],
    "created_at": "2026-01-19T10:45:00Z"
  },
  "meta": null
}
```

---

07. Notifications Module

Get Notifications List

```
GET /api/v1/user/notifications?page=1&per_page=20
Authorization: Bearer {user_token}
```

Response (200):

```
{
  "success": true,
  "message": "Notifications retrieved",
  "data": [
    {
      "id": 1,
      "type": "REVIEW_SUBMITTED",
      "title": "Review Submitted",
      "body": "Your review for Starbucks has been submitted successfully.",
      "data": { "review_id": 123 },
      "is_read": false,
      "created_at": "2026-01-19T10:46:00Z"
    },
    {
      "id": 2,
      "type": "POINTS_AWARDED",
      "title": "Points Earned",
      "body": "You earned 50 points for your review!",
```

```
      "data": { "points": 50 },
      "is_read": true,
      "created_at": "2026-01-19T10:46:10Z"
    }
  ],
  "meta": {
    "unread_count": 5,
    "current_page": 1,
    "per_page": 20,
    "total": 25
  }
}
```

Get Unread Count

```
GET /api/v1/user/notifications/unread-count
Authorization: Bearer {user_token}
```

Response (200):

```
{
  "success": true,
  "message": "Unread count retrieved",
  "data": {
    "unread_count": 5
  },
  "meta": null
}
```

Mark Notification as Read

```
POST /api/v1/user/notifications/{notification_id}/read
Authorization: Bearer {user_token}
```

Response (200):

```
{
  "success": true,
  "message": "Notification marked as read",
  "data": null,
  "meta": null
}
```

Mark All as Read

```
POST /api/v1/user/notifications/read-all
Authorization: Bearer {user_token}
```

Response (200):

```
{
  "success": true,
  "message": "All notifications marked as read",
  "data": null,
  "meta": null
}
```

Delete Notification

```
DELETE /api/v1/user/notifications/{notification_id}
Authorization: Bearer {user_token}
```

Response (200):

```
{
  "success": true,
  "message": "Notification deleted",
  "data": null,
  "meta": null
}
```

Clear All Notifications

```
DELETE /api/v1/user/notifications
Authorization: Bearer {user_token}
```

Response (200):

```
{
  "success": true,
  "message": "All notifications cleared",
  "data": null,
  "meta": null
}
```

---

08. Points (Loyalty) Module

Get Points Summary

```
GET /api/v1/user/points/summary
Authorization: Bearer {user_token}
```

Response (200):

```
{
  "success": true,
  "message": "Points summary retrieved",
  "data": {
    "points_balance": 350,
    "total_earned": 500,
    "total_redeemed": 150,
    "expiring_soon": 50,
    "expiring_soon_date": "2026-02-15T00:00:00Z"
  },
  "meta": null
}
```

Balance Calculation:

- Sum of all `points_transactions` where `user_id = current_user`
- Only unexpired transactions (`expires_at IS NULL OR expires_at > NOW()`)
- Positive transactions (EARN_*) add points
- Negative transactions (REDEEM_*, EXPIRE) subtract points

Get Points History

```
GET /api/v1/user/points/history?page=1
Authorization: Bearer {user_token}
```

Response (200):

```
{
  "success": true,
  "message": "Points history retrieved",
  "data": [
    {
      "id": 1,
      "type": "EARN_REVIEW",
      "points": 50,
      "reference_type": "review",
      "reference_id": 123,
      "expires_at": "2027-01-19T10:46:00Z",
      "created_at": "2026-01-19T10:46:00Z"
    },
    {
      "id": 2,
      "type": "REDEEM_VOUCHER",
      "points": -100,
      "reference_type": null,
      "reference_id": null,
      "expires_at": null,
      "created_at": "2026-01-18T14:20:00Z"
    }
  ],
  "meta": {
    "current_page": 1,
    "per_page": 20,
    "total": 45
  }
}
```


Transaction Types:

- `EARN_REVIEW`: Points earned for review submission
- `EARN_INVITE`: Points earned when invited friend joins (stored in invites logic)
- `REDEEM_VOUCHER`: Points redeemed for voucher
- `ADJUST_ADMIN`: Manual adjustment by admin
- `EXPIRE`: Points expired (auto-processed by cron job)

Redeem Points

```
POST /api/v1/user/points/redeem
Authorization: Bearer {user_token}
Content-Type: application/json
```

Request Body:

```
{
  "points": 100
}
```

Response (200):

```
{
  "success": true,
  "message": "Points redeemed successfully",
  "data": {
    "points_balance": 250,
    "redeemed_points": 100
  },
  "meta": null
}
```

Error (422) - Insufficient Balance:

```
{
  "success": false,
  "message": "Insufficient points balance",
  "data": {
    "current_balance": 50,
    "requested_points": 100
  },
  "meta": null
}
```

Validation Rules:

- `points`: required, integer, min 1
- Current balance must be >= requested points

---

09. Invites (Referral) Module

Check Phones

```
POST /api/v1/user/invites/check-phones
Authorization: Bearer {user_token}
Content-Type: application/json
```

Request Body:

```
{
  "phones": ["+201000000001", "+201000000002", "+201000000003"]
}
```

Response (200):

```
{
  "success": true,
  "message": "Phone status checked",
  "data": {
    "installed": [
      { "phone": "+201000000001", "user_id": 5 }
    ],
    "not_installed": [
      { "phone": "+201000000002" },
      { "phone": "+201000000003" }
    ]
  },
  "meta": null
}
```

Business Logic:

- "installed" = phone is registered in users table
- "not_installed" = phone is not registered (can be invited)

Create Invites

```
POST /api/v1/user/invites
Authorization: Bearer {user_token}
Content-Type: application/json
```

Request Body:

```
{
  "phones": ["+201000000002", "+201000000003"]
}
```

Response (200):

```
{
  "success": true,
  "message": "Invites created successfully",
```

```
  "data": {
    "invited_count": 2,
    "reward_points_per_friend": 50
  },
  "meta": null
}
```

Validation Rules:

- `phones`: required, array, min 1 phone
- Each phone: valid E164 format
- Skips phones already registered
- Creates invite record with status 'pending'
- `reward_points` from `points_settings.invite_points_per_friend`

Error (422) - Duplicate Invite:

```
{
  "success": false,
  "message": "You have already invited this phone number",
  "data": {
    "phone": "+201000000002"
  },
  "meta": null
}
```

Get My Invites

```
GET /api/v1/user/invites
Authorization: Bearer {user_token}
```

Response (200):

```
{
  "success": true,
  "message": "Invites retrieved",
  "data": [
    {
      "id": 1,
      "invited_phone": "+201000000002",
      "status": "pending",
      "reward_points": 50,
      "created_at": "2026-01-18T09:00:00Z"
    },
    {
```

```
      "id": 2,
      "invited_phone": "+201000000003",
      "status": "joined",
      "reward_points": 50,
      "rewarded_at": "2026-01-19T11:00:00Z",
      "invited_user": { "id": 10, "full_name": "Sara Ahmed" },
      "created_at": "2026-01-18T09:00:00Z"
    }
  ],
  "meta": null
}
```

Invite Statuses:

- `pending`: Invite sent, friend hasn't registered yet
- `joined`: Friend registered using invite
- `rejected`: (future) Friend declined invite
- `expired`: (future) Invite expired

Business Logic - Invite Completion:

- When new user registers with `invited_by_phone` matching inviter's phone:
  1. Find matching invite (`inviter_user.phone` matches `invited_by_phone`, status='pending')
  2. Create PointsTransaction for inviter (type: EARN_INVITE, source: invite record)
  3. Update invite: `status='joined'`, `invited_user_id=new_user.id`, `rewarded_at=NOW()`
- Idempotent: if invite already completed, skip reward

---

10. Profile Module

Get Profile

```
GET /api/v1/user/profile
Authorization: Bearer {user_token}
```

Response (200):

```
{
  "success": true,
  "message": "Profile retrieved",
  "data": {
    "id": 1,
    "full_name": "Ahmed Ali",
    "phone": "+201000000000",
    "email": "ahmed@example.com",
    "birth_date": "1995-05-15",
    "avatar_url": "https://example.com/storage/avatars/123.jpg",
    "gender": { "id": 1, "name": "Male" },
```

```
    "nationality": { "id": 1, "name": "Egypt", "flag_url": "..." },
    "is_phone_verified": true,
    "created_at": "2026-01-01T00:00:00Z"
  },
  "meta": null
}
```

Update Profile

```
POST /api/v1/user/profile
Authorization: Bearer {user_token}
Content-Type: multipart/form-data
```

Request Body (multipart):

```
full_name: Ahmed Ali Updated
avatar: (file)
```

Response (200):

```
{
  "success": true,
  "message": "Profile updated successfully",
  "data": {
    "id": 1,
    "full_name": "Ahmed Ali Updated",
    "avatar_url": "https://example.com/storage/avatars/123_updated.jpg"
  },
  "meta": null
}
```

Validation Rules:

- `full_name`: optional, string, max 255
- `avatar`: optional, image file, max 5MB

Profile Phone Change - Send OTP

```
POST /api/v1/user/profile/phone/send-otp
Authorization: Bearer {user_token}
Content-Type: application/json
```

Request Body:

```
{
  "phone": "+201000000099"
}
```

Response (200):

```
{
  "success": true,
  "message": "OTP sent to new phone number",
  "data": null,
  "meta": {
    "otp_code": "5678"
  }
}
```

Validation Rules:

- phone: required, valid E164 format, not already registered

Profile Phone Change - Verify OTP

```
POST /api/v1/user/profile/phone/verify-otp
Authorization: Bearer {user_token}
Content-Type: application/json
```

Request Body:

```
{
  "phone": "+201000000099",
  "otp": "5678"
}
```

Response (200):

```
{
  "success": true,
  "message": "Phone number updated successfully",
  "data": {
    "phone": "+201000000099",
    "is_phone_verified": true
  },
```

```
  "meta": null
}
```

Business Logic:

1. Validate OTP against `phone_change_requests` table
2. Check OTP not expired (`expires_at > NOW()`)
3. Update user's phone number
4. Set `is_phone_verified=true`
5. Delete phone_change_request record

Error (422) - Invalid OTP:

```
{
  "success": false,
  "message": "Invalid or expired OTP",
  "data": null,
  "meta": null
}
```

---

4. Critical Business Logic

Reviews Flow (Complete Lifecycle)

Step 1: QR Code Scan

1. User scans QR code at branch
2. System validates QR code (`branches.qr_code_value`)
3. System creates `branch_qr_sessions` record:
   - Generates unique `session_token`
   - Sets `expires_at` = NOW() + 30 minutes
   - Links to `user_id` and `branch_id`
4. Returns session token + branch details

Step 2: Fetch Questions

1. User requests questions for branch
2. System resolves branch → place → subcategory → rating criteria
3. Returns criteria filtered by subcategory (via `subcategory_rating_criteria` pivot)
4. Criteria types: RATING (1-5 scale), YES_NO (boolean), MULTIPLE_CHOICE (predefined choices)

Step 3: Review Submission

1. **Session Validation:**
   - Token exists and belongs to user
   - Not expired (`expires_at > NOW()`)
   - Not consumed (`consumed_at IS NULL`)

2. **Cooldown Check:**
    - Query last review by user for same branch
    - If exists and `created_at` + `branch.review_cooldown_days` > NOW(), reject
    - Cooldown is per-branch (configurable: 0, 7, 30, 90 days)
3. **Answer Validation:**
    - Normalize multipart data (string "1"/"0" → int/bool)
    - For each answer:
        - **RATING**: `rating_value` must be integer 1-5
        - **YES_NO**: `yes_no_value` must be boolean (auto-normalized from string)
        - **MULTIPLE_CHOICE**: `choice_id` must exist in `rating_criteria_choices` and belong to criteria
4. **Photo Upload:**
    - Max 3 photos
    - Validate: image format, max 5MB per file
    - Store in `storage/reviews/{review_id}/photo{n}.jpg`
    - Create `review_photos` records
5. **Session Consumption:**
    - Update `branch_qr_sessions.consumed_at = NOW()`
    - Token cannot be reused
6. **Review Score Calculation:**
    - Average of all `rating_value` answers
    - Formula: `SUM(rating_value) / COUNT(rating_value)`
    - Stored in `reviews.review_score`
7. **Points Awarding:**
    - Call `PointsService::awardPointsForReview(user, review)`
    - Idempotent check: prevent duplicate via `source_type='review'` + `source_id=review.id`
    - Award points from `points_settings.points_per_review`
    - Set expiration: `expires_at = NOW() + brand.points_expiry_days` (if brand configured)
    - Create `points_transactions` record (type: EARN_REVIEW)
8. **Response:**
    - Return review object
    - Include meta: `points_awarded`, `points_balance`

Edge Cases:

- **Expired Session:** User must rescan QR
- **Cooldown Active:** User cannot review until period passes
- **Invalid Criteria Answer:** Validation error with field-level details
- **Photo Upload Failure:** Review created without photos (photos are optional)

- **Duplicate Point Award:** Idempotent check prevents double-award

---

Points (Loyalty) System

Earning Points:

1. **Review Submission:**
   - Source: `PointsService::awardPointsForReview()`
   - Points: `points_settings.points_per_review` (default 50)
   - Expiration: Brand-specific (`brands.points_expiry_days`) or global setting
   - Transaction type: `EARN_REVIEW`
   - Idempotent: checks for existing transaction with same `source_type='review'` + `source_id`
2. **Invite Completion:**
   - Source: `InviteService::completeInviteForNewUser()`
   - Points: `points_settings.invite_points_per_friend` (default 50)
   - Expiration: Same as review points
   - Transaction type: `EARN_INVITE` (stored in reference_type/reference_id)
   - Triggered when invited user registers

Balance Calculation:

```
SELECT SUM(points) as balance
FROM points_transactions
WHERE user_id = ?
  AND (expires_at IS NULL OR expires_at > NOW())
```

Points Expiration:

- Each transaction has `expires_at` timestamp (nullable)
- Expired points are **excluded** from balance calculation
- Cron job (recommended daily) creates `EXPIRE` transactions for cleanup

Redemption:

1. Validate sufficient balance
2. Create negative transaction (type: `REDEEM_VOUCHER`, points: `-{amount}`)
3. No expiration on redemption transactions
4. Return new balance

Notifications System

Triggers:

- Review submitted successfully
- Points awarded

- Invite friend joined
- Profile updated
- (Future) Voucher redemption, promotions

Storage:

- `user_notifications` table
- Fields: `type`, `title`, `body`, `data` (JSON payload), `is_read`, `sent_at`

Read Status:

- `is_read` boolean flag
- `read_at` timestamp (nullable)
- Both fields supported for backward compatibility

Operations:

- List (paginated, filterable by unread)
- Unread count (badge)
- Mark single as read
- Mark all as read
- Delete single
- Clear all

Invites (Referral) System

Lifecycle:

Phase 1: Invite Creation

1. User provides phone numbers
2. System checks install status (registered vs not registered)
3. For non-registered phones:
    - Create `invites` record
    - Status: `pending`
    - `reward_points`: from `points_settings.invite_points_per_friend`
    - Unique constraint: `(inviter_user_id, invited_phone)`

Phase 2: Registration Completion

1. New user registers with `invited_by_phone`
2. System finds matching invite:
    - `inviter_user.phone == invited_by_phone`
    - Status: `pending`
3. Award points to inviter:
    - Create `points_transactions` (type: EARN_INVITE)
    - Points: `invite.reward_points`
4. Update invite:
    - `status = 'joined'`
    - `invited_user_id = new_user.id`

- `rewarded_at = NOW()`

Edge Cases:

- Duplicate invite: unique constraint prevents re-invite of same phone
- Inviter invites themselves: validation prevents
- Invited user already registered: skipped at invite creation
- Invite not found at registration: registration proceeds without reward

---

Profile Phone Change Flow

Step 1: Send OTP

1. User requests phone change with new phone number
2. Validate: phone not already registered
3. Generate OTP (4-6 digits)
4. Create `phone_change_requests` record:
   - `user_id`, `new_phone`, `otp_hash`
   - `expires_at = NOW() + 10 minutes`
5. Send OTP via SMS (integration required)

Step 2: Verify OTP

1. User submits phone + OTP
2. Validate:
   - Record exists for `user_id` + `new_phone`
   - OTP hash matches
   - Not expired (`expires_at > NOW()`)
   - Attempts < 3 (configurable)
3. Update user:
   - `phone = new_phone`
   - `is_phone_verified = true`
4. Delete `phone_change_requests` record

Security:

- OTP stored as hash (bcrypt)
- Max 3 attempts before expiration
- 10-minute expiration window
- Phone uniqueness validated before OTP send

---

5. Database Mapping

Entity Relationship Overview

```
users
  └─ one-to-many → reviews
  └─ one-to-many → points_transactions
```

```
  └─ one-to-many → invites (as inviter)
  └─ one-to-many → user_notifications
  └─ one-to-many → phone_change_requests
  └─ many-to-one ← genders
  └─ many-to-one ← nationalities

categories
  └─ one-to-many → subcategories
  └─ many-to-many ← rating_criteria (via subcategory_rating_criteria)

subcategories
  └─ one-to-many → places
  └─ many-to-many ← rating_criteria

brands
  └─ one-to-many → places
  └─ one-to-many → points_transactions

places
  └─ one-to-many → branches
  └─ one-to-many → reviews
  └─ many-to-one ← brands
  └─ many-to-one ← subcategories

branches
  └─ one-to-many → reviews
  └─ one-to-many → branch_qr_sessions
  └─ many-to-one ← places

rating_criteria
  └─ one-to-many → rating_criteria_choices
  └─ one-to-many → review_answers
  └─ many-to-many ← subcategories

reviews
  └─ one-to-many → review_answers
  └─ one-to-many → review_photos
  └─ many-to-one ← users
  └─ many-to-one ← branches
  └─ many-to-one ← places
```

Table Schemas

users

```
id (PK)
full_name (string, 255)
phone (string, unique, E164)
email (string, unique, nullable)
password (hashed)
birth_date (date)
```

```
gender_id (FK → genders)
nationality_id (FK → nationalities)
avatar_url (string, nullable)
is_phone_verified (boolean, default false)
timestamps, soft_deletes

Indexes:
- phone (unique)
- email (unique)
- gender_id
- nationality_id
```

categories

```
id (PK)
name_en (string, 255)
name_ar (string, 255, nullable)
logo_url (string, nullable)
is_active (boolean, default true)
timestamps

Indexes:
- is_active
```

subcategories

```
id (PK)
category_id (FK → categories, cascade delete)
name_en (string, 255)
name_ar (string, 255, nullable)
image_url (string, nullable)
is_active (boolean, default true)
timestamps

Indexes:
- category_id
- is_active
```

brands

```
id (PK)
name (string, 255)
logo_url (string, nullable)
points_expiry_days (integer, nullable) -- null = global default
timestamps, soft_deletes
```

```
Indexes:
- name (for search)
```

places

```
id (PK)
brand_id (FK → brands, nullable, set null on delete)
subcategory_id (FK → subcategories, nullable, set null on delete)
name (string, 255)
description (text, nullable)
is_featured (boolean, default false)
city (string, nullable)
area (string, nullable)
meta (json, nullable)
timestamps, soft_deletes

Indexes:
- brand_id
- subcategory_id
- (city, area)
- is_featured
- name (trigram for full-text search)
```

branches

```
id (PK)
place_id (FK → places, cascade delete)
name (string, nullable)
address (text)
lat (decimal 10,7, nullable)
lng (decimal 10,7, nullable)
working_hours (json, nullable) -- { "monday": "08:00-22:00", ... }
qr_code_value (string, unique)
qr_generated_at (timestamp, nullable)
review_cooldown_days (integer, default 0) -- 0 = no cooldown
timestamps, soft_deletes

Indexes:
- place_id
- qr_code_value (unique)
- (lat, lng) -- for geospatial queries
```

rating_criteria

```
id (PK)
type (enum: RATING, YES_NO, MULTIPLE_CHOICE)
```

```
question_en (text)
question_ar (text, nullable)
is_required (boolean, default true)
display_order (integer, default 0)
timestamps

Indexes:
- type
- display_order
```

rating_criteria_choices

```
id (PK)
criteria_id (FK → rating_criteria, cascade delete)
text_en (string, 255)
text_ar (string, 255, nullable)
display_order (integer, default 0)
timestamps

Indexes:
- criteria_id
```

subcategory_rating_criteria (pivot)

```
subcategory_id (FK → subcategories, cascade delete)
criteria_id (FK → rating_criteria, cascade delete)

Indexes:
- (subcategory_id, criteria_id) -- composite unique
```

branch_qr_sessions

```
id (PK)
user_id (FK → users, cascade delete)
branch_id (FK → branches, cascade delete)
qr_code_value (string)
session_token (string, unique)
scanned_at (timestamp)
expires_at (timestamp) -- scanned_at + 30 minutes
consumed_at (timestamp, nullable) -- set when review submitted
timestamps

Indexes:
- session_token (unique)
- (user_id, branch_id)
- expires_at
```

reviews

```
id (PK)
user_id (FK → users, cascade delete)
place_id (FK → places, nullable, set null on delete)
branch_id (FK → branches, cascade delete)
overall_rating (decimal 2,1) -- 1.0 to 5.0
comment (text, nullable)
status (enum: ACTIVE, DELETED_BY_ADMIN)
review_score (decimal 5,2, nullable) -- calculated average
timestamps, soft_deletes

Indexes:
- user_id
- branch_id
- place_id
- created_at
- status
```

review_answers

```
id (PK)
review_id (FK → reviews, cascade delete)
criteria_id (FK → rating_criteria, cascade delete)
rating_value (integer, nullable) -- 1-5 for RATING type
yes_no_value (boolean, nullable) -- for YES_NO type
choice_id (FK → rating_criteria_choices, nullable, set null) -- for
MULTIPLE_CHOICE
timestamps

Indexes:
- review_id
- criteria_id
```

review_photos

```
id (PK)
review_id (FK → reviews, cascade delete)
url (text) -- full URL to storage
display_order (integer, default 0)
timestamps

Indexes:
- review_id
```

points_transactions

```
id (PK)
user_id (FK → users, cascade delete)
brand_id (FK → brands, nullable, set null)
type (enum: EARN_REVIEW, REDEEM_VOUCHER, ADJUST_ADMIN, EXPIRE, EARN_INVITE)
points (integer) -- positive for earn, negative for redeem/expire
reference_type (string, nullable) -- e.g. "review", "invite"
reference_id (bigint, nullable) -- polymorphic relation
expires_at (timestamp, nullable)
meta (json, nullable)
timestamps

Indexes:
- user_id
- (user_id, created_at)
- (user_id, expires_at)
- type
- (reference_type, reference_id) -- for idempotency check

Constraints:
- Unique: (user_id, reference_type, reference_id) WHERE reference_type IS NOT NULL
```

points_settings

```
id (PK)
points_per_review (integer, default 0)
invite_points_per_friend (integer, default 50)
invitee_bonus_points (integer, default 0) -- future: bonus for invited user
point_value_money (decimal 10,2, default 0) -- e.g. 1 point = 0.10 EGP
currency (string, 3, default "EGP")
is_active (boolean, default true)
timestamps

Indexes:
- (is_active, created_at)

Note: Only one active setting at a time. Get latest by `is_active=1 ORDER BY
created_at DESC LIMIT 1`
```

invites

```
id (PK)
inviter_user_id (FK → users, cascade delete)
invited_phone (string, E164)
invited_user_id (FK → users, nullable, set null) -- set when friend joins
status (enum: pending, joined, rejected, expired)
```

```
reward_points (integer, default 50) -- snapshot from settings
rewarded_at (timestamp, nullable) -- when inviter received points
timestamps

Indexes:
- inviter_user_id
- invited_phone
- invited_user_id
- status
- (inviter_user_id, invited_phone) -- unique constraint
```

user_notifications

```
id (PK)
user_id (FK → users, cascade delete)
type (string) -- e.g. REVIEW_SUBMITTED, POINTS_AWARDED
title (string, 255)
body (text, nullable)
data (json, nullable) -- additional payload
is_read (boolean, default false)
sent_at (timestamp, nullable)
timestamps

Indexes:
- (user_id, is_read)
- (user_id, created_at)
```

phone_change_requests

```
id (PK)
user_id (FK → users, cascade delete)
new_phone (string)
otp_hash (string) -- bcrypt hash
attempts (integer, default 0)
expires_at (timestamp, nullable) -- 10 minutes from creation
verified_at (timestamp, nullable)
timestamps

Indexes:
- user_id
- new_phone
- expires_at
```

otp_codes

```
id (PK)
phone (string)
otp_hash (string)
type (string) -- e.g. PHONE_VERIFY, FORGOT_PASSWORD
attempts (integer, default 0)
expires_at (timestamp)
verified_at (timestamp, nullable)
timestamps

Indexes:
- phone
- type
- expires_at
```

genders

```
id (PK)
code (string, unique) -- MALE, FEMALE, OTHER
name_en (string, 100)
name_ar (string, 100, nullable)
timestamps

Data seeded at installation
```

nationalities

```
id (PK)
country_code (string, 2, unique) -- ISO 3166-1 alpha-2 (EG, SA, AE, etc.)
name_en (string, 255)
name_ar (string, 255, nullable)
flag_url (computed) -- https://flagsapi.com/{country_code}/flat/64.png
timestamps

Data seeded at installation (200+ countries)
```

Recommended Indexes (Performance Optimization)

Critical for Performance:

1. `reviews.created_at` + `reviews.branch_id` composite (cooldown check)
2. `points_transactions (user_id, expires_at)` composite (balance calculation)
3. `points_transactions (reference_type, reference_id)` composite (idempotency)
4. `invites (inviter_user_id, invited_phone)` unique composite
5. `branch_qr_sessions.session_token` unique (session lookup)
6. `branches.qr_code_value` unique (QR scan)

Nice to Have: 7. `places.name` trigram/full-text index (search) 8. `branches (lat, lng)` geospatial index (location-based queries) 9. `user_notifications (user_id, created_at DESC)` composite (timeline queries)

---

6. Postman Collection Structure

Recommended Folder Organization

```
RateIt-User API
├── Variables (Collection-level)
│   ├── base_url: http://localhost/rate-it-backend/public
│   ├── api_version: v1
│   ├── user_token: (auto-set on login/register)
│   ├── reset_token: (auto-set on forgot-password verify)
│   ├── phone: +201000000000
│   ├── otp: (auto-set on OTP send)
│   ├── category_id: (auto-set from categories response)
│   ├── brand_id: (auto-set)
│   ├── place_id: (auto-set)
│   ├── branch_id: (auto-set)
│   ├── branch_qr: BRANCH_123_QR_XYZ
│   ├── review_session_token: (auto-set on QR scan)
│   ├── review_id: (auto-set on review create)
│   └── first_notification_id: (auto-set)
│
├── Pre-request Script (Collection-level)
│   └── Set Accept, X-Lang headers
│   └── Conditionally set Content-Type for raw JSON
│
├── 01. Auth
│   ├── Public
│   │   ├── POST Register (sets user_token in test)
│   │   ├── POST Login (sets user_token in test)
│   │   ├── POST Forgot Password - Send OTP (sets otp in test)
│   │   ├── POST Forgot Password - Verify OTP (sets reset_token in test)
│   │   ├── POST Forgot Password - Reset
│   │   ├── POST Phone - Send OTP (sets otp in test)
│   │   └── POST Phone - Verify OTP (sets user_token in test)
│   └── Protected
│       ├── GET Me
│       └── POST Logout
│
├── 02. Home
│   ├── GET Home Data
│   └── GET Search
│
├── 03. Onboarding
│   └── GET Onboarding Steps
│
├── 04. Lookups
│   ├── GET Genders
```

```
│   └── GET Nationalities
│
├── 05. Categories
│   ├── GET Categories (sets category_id in test)
│   ├── GET Subcategories
│   └── GET Categories Search
│
├── 06. Brands & Places
│   ├── GET Brand Details (sets brand_id in test)
│   ├── GET Brand Places (sets place_id in test)
│   ├── GET Place Details (sets branch_id in test)
│   └── GET Place Reviews
│
├── 07. Reviews
│   ├── POST Scan QR (sets review_session_token, branch_id in test)
│   ├── GET Branch Questions
│   ├── POST Create Review (sets review_id in test)
│   ├── GET My Reviews
│   └── GET Review Details
│
├── 08. Notifications
│   ├── GET Notifications List (sets first_notification_id in test)
│   ├── GET Unread Count
│   ├── POST Mark Read
│   ├── POST Mark All Read
│   ├── DELETE Delete One
│   └── DELETE Clear All
│
├── 09. Points
│   ├── GET Points Summary (sets points_balance in test)
│   ├── GET Points History
│   └── POST Redeem Points
│
├── 10. Invites
│   ├── POST Check Phones
│   ├── POST Create Invites
│   └── GET My Invites
│
└── 11. Profile
    ├── GET Profile
    ├── POST Update Profile
    ├── POST Phone - Send OTP
    └── POST Phone - Verify OTP
```

Pre-request Script (Collection-level)

```javascript
// Set standard headers
pm.request.headers.add({
    key: 'Accept',
    value: 'application/json'
});
```

```
pm.request.headers.add({
    key: 'X-Lang',
    value: 'en'
});

// Only set Content-Type for raw JSON bodies (NOT formdata)
if (pm.request.body && pm.request.body.mode === 'raw') {
    pm.request.headers.add({
        key: 'Content-Type',
        value: 'application/json'
    });
}
```

Test Script Template (Request-level)

```
// Status assertion
pm.test('Status is 200', function() {
    pm.response.to.have.status(200);
});

// Parse response
var json = pm.response.json();

// Success flag assertion
pm.test('Success is true', function() {
    pm.expect(json.success).to.eql(true);
});

// Auto-save tokens/IDs to environment
if (json.data && json.data.token) {
    pm.environment.set('user_token', json.data.token);
}

if (json.data && json.data.id) {
    pm.environment.set('review_id', json.data.id);
}

// Meta extraction
if (json.meta && json.meta.otp_code) {
    pm.environment.set('otp', json.meta.otp_code);
}
```

Best Practices

1. **Token Management:**
    - Auto-extract `user_token` from register/login responses
    - Use `{{user_token}}` variable in Authorization headers
    - Clear token on logout (manual or test script)

2. **ID Propagation:**
    - Save IDs in test scripts for dependent requests
    - Example: `category_id` from GET Categories → used in GET Subcategories
3. **Response Examples:**
    - Add success and error examples to each request
    - Include validation errors (422), business logic errors (400), auth errors (401)
4. **Folder Auth:**
    - Set folder-level auth for Protected endpoints
    - Type: Bearer Token, Value: `{{user_token}}`
5. **Environment Setup:**
    - Create separate environments: Local, Staging, Production
    - Override `base_url` per environment
    - Keep sensitive data (tokens) in environment, not collection

---

7. Changelog

Version 1.0 (2026-01-19)

- **Auth Module:** Implemented phone-based registration, login, forgot password, phone verification
- **Reviews Module:** Complete QR-based review flow with multi-criteria validation, photo uploads, cooldown enforcement
- **Points Module:** Loyalty system with earning, balance calculation, redemption, expiration
- **Invites Module:** Referral program with phone check, invite creation, automatic reward on completion
- **Notifications Module:** User notification management with CRUD operations
- **Profile Module:** Profile retrieval, update (avatar upload), phone change flow
- **Database:** 20+ tables with proper relationships, indexes, constraints
- **Postman Collection:** Comprehensive collection with 50+ endpoints, auto-variable setting, test automation

Known Limitations

- Phone OTP sending requires SMS gateway integration (currently logs OTP to console)
- Push notifications not implemented (user_notifications table only)
- Points expiration cron job not implemented (manual trigger required)
- Review photo moderation not implemented (auto-approved)
- Multi-language support partial (en/ar structure exists, translations incomplete)

Roadmap

- Admin panel for review moderation
- Voucher redemption catalog
- Advanced search with filters (location, rating range, category)
- Social features (follow users, like reviews)

- Analytics dashboard (user activity, review trends)

Missing Information

The following details are **not confirmed** in the codebase and may require clarification:

1. **Home Module Endpoints:** Exact implementation of banners, featured categories not fully documented
2. **Onboarding Module:** No routes/controllers found for onboarding flow
3. **SMS Gateway Integration:** OTP sending mechanism not implemented (logs to console only)
4. **Push Notifications:** Firebase/OneSignal integration details not present
5. **File Storage Configuration:** Exact storage driver (local/S3) not specified
6. **Rate Limiting:** No rate limiting observed on API endpoints
7. **Admin Panel:** No admin-specific routes or controllers found
8. **Voucher Catalog:** Points redemption targets (vouchers) not implemented
9. **Review Moderation:** Auto-approval vs manual review process not defined
10. **Multi-tenancy:** Single database vs multi-tenant setup not clarified

---

Document Version: 1.0
Generated: 2026-01-19
Maintained By: Development Team
Contact: dev@rateit.example.com