Rate It — Admin Backend Handover

0) Document Control

Property	Value
Document Title	Admin Backend Handover
Scope	Admin Module only (app/Modules/Admin/*)
Version	1.0
Last Updated	January 25, 2026
Owner / Maintainers	Backend Team
Audience	Backend Developers, QA, DevOps

Related Documentation

- **SRS & Non-Technical Docs**: `docs/ADMIN_README.md`, `docs/ADMIN_QUICK_REFERENCE.md`, `docs/ADMIN_WORKFLOW_FLOWS.md`
- **Technical API Docs**: `docs/ADMIN_DOCUMENTATION.md`
- **Testing Guide**: `TESTING.md`
- **Code Source of Truth**: All implemented code in `app/Modules/Admin/*`

Important: If this document conflicts with the actual code, **treat the code as the source of truth**. This document is a guide; verify endpoint signatures and responses by reading the controller code.

---

1) Executive Summary

The **Admin Backend Module** provides a comprehensive management system for the Rate-It platform. It enables administrators to:

- **Authenticate & Manage Access**: Secure login and role-based permission enforcement via Sanctum tokens
- **Manage Platform Catalog**: Create/edit/manage categories, subcategories, brands, places, branches, and rating criteria
- **Moderate Reviews**: View, hide, reply to, and feature user reviews
- **Monitor Users**: List users, view their profiles, block/unblock accounts, track reviews and points
- **Track Points & Loyalty**: Monitor points transactions, configure loyalty settings and rules
- **Manage Subscriptions**: Create and manage subscription plans
- **Send Notifications**: Create message templates, broadcast announcements, send targeted notifications
- **View Analytics**: Dashboard KPIs (users count, reviews, ratings, points metrics), top places rankings, review timeseries charts
- **Manage Invitations**: Monitor user invitation campaigns and their status
- **Configure RBAC**: Create roles, manage permissions, and assign granular access to other admins

Architecture Snapshot

The Admin module follows a **layered architecture**:

- **Routes** (`Routes/api.php`) → **Controllers** (HTTP layer) → **Services** (business logic) → **Models** (Eloquent) → **Database**
- **Middleware**: `AdminAuthenticate` (Sanctum token) + `AdminPermission` (policy enforcement)
- **Response Format**: Unified JSON wrapper with `{success, message, data, meta}` structure
- **Validation**: FormRequest classes for input validation
- **Resources**: JSON:API-style response transformers

---

2) Repository & Module Structure

Admin Module Layout

```
app/Modules/Admin/
├── Auth/
│   ├── Controllers/
│   │   └── AuthController.php          # Login, logout, me endpoint
│   ├── Requests/
│   │   └── LoginRequest.php
│   ├── Resources/
│   │   └── AdminResource.php           # Admin DTO with roles/permissions
│   ├── Routes/
│   │   └── api.php                     # /api/v1/admin/auth/*
│   └── Services/
│       └── AuthService.php             # Login logic, token generation
├── Rbac/
│   └── Controllers/
│       ├── RolesController.php         # Role CRUD
│       └── PermissionsController.php   # Permission listing
├── Catalog/
│   ├── Controllers/
│   │   ├── CategoriesController.php
│   │   ├── SubcategoriesController.php
│   │   ├── BrandsController.php
│   │   ├── PlacesController.php
│   │   ├── BranchesController.php
│   │   ├── RatingCriteriaController.php
│   │   ├── RatingCriteriaChoicesController.php
│   │   └── SubcategoryRatingCriteriaController.php
│   ├── Requests/
│   │   └── Store/Update*Request.php
│   ├── Resources/
│   │   └── *Resource.php
│   ├── Routes/
│   │   └── api.php                     # /api/v1/admin/categories, brands,
places, etc.
│   └── Services/
│       ├── CategoryService.php
│       ├── SubcategoryService.php
│       ├── BrandService.php
│       ├── PlaceService.php
│       ├── BranchService.php
│       ├── RatingCriteriaService.php
│       ├── RatingCriteriaChoiceService.php
│       └── SubcategoryCriteriaService.php
├── CatalogIntegrity/
│   ├── Controllers/
│   │   └── LookupController.php        # Data validation endpoints
│   └── Requests/
├── Dashboard/
│   ├── Controllers/
│   │   └── DashboardController.php     # Analytics endpoints
│   ├── Requests/
│   │   └── DashboardChartRequest.php
│   ├── Routes/
│   │   └── api.php
│   └── Services/
│       └── DashboardService.php        # KPI, top places, chart logic
├── Reviews/
│   ├── Controllers/
│   │   └── ReviewsController.php       # List, show, hide, reply, feature
│   ├── Requests/
│   │   ├── AdminReviewsIndexRequest.php
│   │   ├── HideReviewRequest.php
│   │   ├── ReplyReviewRequest.php
│   │   └── MarkFeaturedRequest.php
│   ├── Resources/
│   │   ├── AdminReviewResource.php     # Detailed review DTO
│   │   └── AdminReviewListResource.php # List item DTO
│   ├── Routes/
│   │   └── api.php
│   └── Services/
│       └── ReviewModerationService.php
├── Users/
│   ├── Controllers/
│   │   └── UsersController.php         # List, show, block, reviews, points
│   ├── Requests/
│   │   ├── AdminUsersIndexRequest.php
│   │   └── BlockUserRequest.php
│   ├── Resources/
│   │   ├── AdminUserResource.php
│   │   └── AdminUserListResource.php
│   ├── Routes/
│   │   └── api.php
│   └── Services/
│       └── UserAdminService.php
├── Points/
│   ├── Controllers/
│   │   └── PointsController.php        # List & show transactions
│   ├── Requests/
│   │   └── AdminPointsTransactionsIndexRequest.php
│   ├── Resources/
│   │   └── AdminPointsTransactionResource.php
│   ├── Routes/
│   │   └── api.php
│   └── Services/
│       └── PointsAdminService.php
├── LoyaltySettings/
│   ├── Controllers/
│   │   └── LoyaltySettingsController.php
│   ├── Requests/
│   │   ├── AdminLoyaltySettingsIndexRequest.php
│   │   └── CreateLoyaltySettingRequest.php
│   ├── Resources/
│   │   └── AdminLoyaltySettingResource.php
│   ├── Routes/
│   │   └── api.php
│   └── Services/
│       └── LoyaltySettingsService.php
├── Notifications/
│   ├── Broadcast/
│   │   ├── Controllers/
│   │   │   └── BroadcastController.php
│   │   ├── Requests/
│   │   │   └── SendBroadcastRequest.php
│   │   └── Services/
│   │       └── BroadcastService.php
│   ├── Send/
│   │   ├── Controllers/
│   │   │   └── SendController.php
│   │   ├── Requests/
│   │   └── Services/
│   │       └── SendService.php
│   ├── Templates/
│   │   ├── Controllers/
│   │   │   └── TemplatesController.php
│   │   ├── Requests/
│   │   │   ├── StoreNotificationTemplateRequest.php
│   │   │   └── UpdateNotificationTemplateRequest.php
│   │   ├── Resources/
│   │   │   └── AdminNotificationTemplateResource.php
│   │   └── Services/
│   │       └── TemplatesService.php
│   ├── Routes/
│   │   └── api.php
│   └── Send/
│       └── Controllers/
├── Invites/
│   ├── Controllers/
│   │   └── InvitesController.php
│   ├── Requests/
│   │   └── AdminInvitesIndexRequest.php
│   ├── Resources/
│   │   └── AdminInviteResource.php
│   ├── Routes/
│   │   └── api.php
│   └── Services/
│       └── InvitesService.php
└── Subscriptions/
    ├── Plans/
    │   ├── Controllers/
    │   │   └── PlansController.php
    │   ├── Requests/
    │   │   ├── StorePlanRequest.php
    │   │   ├── UpdatePlanRequest.php
    │   │   └── PlansIndexRequest.php
    │   ├── Resources/
    │   │   └── AdminSubscriptionPlanResource.php
    │   ├── Routes/
    │   │   └── api.php
    │   └── Services/
    │       └── PlansService.php
    └── Subscriptions/
        ├── Controllers/
        │   └── SubscriptionsController.php
        └── Services/
            └── SubscriptionsService.php
```

Shared Components (Used by Admin)

Component	Path	Purpose
BaseApiController	`app/Support/Api/BaseApiController.php`	Base controller with response helpers
ApiResponseTrait	`app/Support/Traits/Api/ApiResponseTrait.php`	JSON response wrapper methods
AdminAuthenticate	`app/Http/Middleware/AdminAuthenticate.php`	Sanctum token validation
AdminPermission	`app/Http/Middleware/AdminPermission.php`	Permission enforcement
Admin Model	`app/Models/Admin.php`	Admin user with Sanctum tokens, role morphing
Role Model	`app/Models/Role.php`	Role with permissions relationship
Permission Model	`app/Models/Permission.php`	Permission definitions
Categories, Brands, Places, etc.	`app/Models/`	Domain models accessed by Admin

3) How to Run Locally

Prerequisites

- PHP 8.1+
- Composer
- SQLite (default) or MySQL
- Laravel 10+
- Node.js (optional, for frontend)

Installation Steps

```
# 1. Clone repository
git clone <repo-url>
cd rate-it-backend

# 2. Install dependencies
composer install

# 3. Copy environment file
cp .env.example .env

# 4. Generate application key
php artisan key:generate

# 5. Run database migrations
php artisan migrate

# 6. Seed test data (includes admins, roles, permissions)
php artisan db:seed

# 7. Start development server
php artisan serve
```

Environment Variables (Admin-Related)

```
# Basic config
APP_NAME=RateIt
APP_ENV=local
APP_DEBUG=true
APP_URL=http://localhost:8000

# Authentication
AUTH_GUARD=web
SANCTUM_STATEFUL_DOMAINS=localhost:3000

# Database
DB_CONNECTION=sqlite
# Or for MySQL:
# DB_CONNECTION=mysql
# DB_HOST=127.0.0.1
# DB_PORT=3306
# DB_DATABASE=rate_it_db
# DB_USERNAME=root
# DB_PASSWORD=

# Session & Queue
SESSION_DRIVER=database
QUEUE_CONNECTION=database

# Mail (for notifications)
MAIL_MAILER=log
# MAIL_FROM_ADDRESS=noreply@rate-it.local

# Logging
LOG_CHANNEL=stack
LOG_LEVEL=debug
```

Database Setup

```
# Run all migrations
php artisan migrate

# Seed sample data
php artisan db:seed

# Rollback and re-migrate (destructive, dev only)
php artisan migrate:refresh --seed

# Create fresh DB with all migrations and seeds
php artisan migrate:fresh --seed
```

Queue & Cron (if needed)

Admin notifications may be queued. Run:

```
# Process queued jobs
php artisan queue:work

# Or in production with supervisor
supervisorctl start queue-worker
```

Testing

```
# Run all tests
composer test

# Run Admin tests only
composer test:admin
```

```
# Run specific test file
php artisan test tests/Feature/Admin/Auth/AuthTest.php

# Verbose output
composer test:admin:verbose
```

---

4) Architecture & Patterns

Request Lifecycle

```
Route (api.php)
  ↓
Middleware (AdminAuthenticate, AdminPermission)
  ↓
Controller (receives validated request)
  ↓
Service (business logic)
  ↓
Model / Eloquent Query
  ↓
Database
  ↓
Service returns result
  ↓
Controller calls Response Helper (success/error/paginated)
  ↓
Resource (transforms model to JSON DTO)
  ↓
Unified JSON Response to Client
```

Dependency Injection

Controllers receive services via constructor injection. Example:

```php
class CategoriesController extends BaseApiController
{
    protected CategoryService $service;

    public function __construct(CategoryService $service)
    {
        $this->service = $service;
    }

    public function store(StoreCategoryRequest $request)
    {
        $data = $request->only(['name_en', 'name_ar', 'is_active']);
        $cat = $this->service->create($data);
        return $this->created(new CategoryResource($cat), 'categories.created');
    }
}
```

Response Wrapper Format

Success Response (HTTP 200):

```
{
  "success": true,
  "message": "Categories list",
  "data": [
    { "id": 1, "name_en": "Food", "name_ar": "الطعام", "is_active": true }
  ],
  "meta": null
}
```

Error Response (HTTP 4xx/5xx):

```
{
  "success": false,
  "message": "Validation failed",
  "data": { "email": ["Email is required"] },
  "meta": null
}
```

Paginated Response (HTTP 200):

```
{
  "success": true,
  "message": "Users list",
  "data": [ { "id": 1, "name": "John", ... } ],
  "meta": {
    "page": 1,
    "limit": 15,
    "total": 150,
    "has_next": true,
    "last_page": 10
  }
}
```

Response Helper Methods (BaseApiController)

```
// Success with optional message key (for i18n)
$this->success($data, 'categories.list', $meta, 200);
```

```
// Error with message key
$this->error('Not found', null, 404);

// Created (201)
$this->created($data, 'categories.created');

// No Content (200 with null data)
$this->noContent('categories.deleted');

// Paginated (wraps LengthAwarePaginator)
$this->paginated($paginator, 'admin.users.list');
```

Validation Pattern (FormRequest)

```
class StoreCategoryRequest extends FormRequest
{
    public function authorize()
    {
        return true;  // Already checked by middleware
    }

    public function rules()
    {
        return [
            'name_en' => 'required|string|max:255',
            'name_ar' => 'nullable|string|max:255',
            'is_active' => 'nullable|boolean',
            'logo' => 'nullable|string|max:1024',
        ];
    }
}
```

In controller:

```
public function store(StoreCategoryRequest $request)
{
    $data = $request->validated();  // Auto-validated, safe to use
    // ...
}
```

Pagination & Filtering Conventions

- **Default Page Size**: 15 items per page (configurable via `per_page` query param)
- **Query Parameters**:
  - `page`: Page number (default 1)
  - `per_page`: Items per page (default 15, max varies by endpoint)
  - `sort_by`: Field to sort by (endpoint-specific)

  - `q`: Full-text search query
  - Date range: `date_from`, `date_to` (YYYY-MM-DD format)
  - Status filters: `is_active`, `is_hidden`, etc. (boolean)

Example:

```
GET /api/v1/admin/reviews?page=2&per_page=10&is_hidden=false&date_from=2026-01-
01&date_to=2026-01-25
```

File Uploads & Storage

Not heavily used in Admin module yet. When implemented:

- Use `Illuminate\Support\Facades\Storage`
- Disk: configured in `config/filesystems.php`
- Path: typically `storage/app/uploads/...`
- Validation: MIME type + size constraints in FormRequest

---

5) Authentication & Authorization (Admin)

Authentication Method: Sanctum Tokens

Guard Configuration (`config/auth.php`):

```
'guards' => [
    'admin' => [
        'driver' => 'sanctum',
        'provider' => 'admins',
    ],
],
'providers' => [
    'admins' => [
        'driver' => 'eloquent',
        'model' => App\Models\Admin::class,
    ],
]
```

Login / Token Flow

Step 1: Login

```
POST /api/v1/admin/auth/login
{
  "email": "admin@rate-it.local",
  "password": "password"
}
```

Response (HTTP 200):

```json
{
  "success": true,
  "message": "Login successful",
  "data": {
    "admin": {
      "id": 1,
      "name": "Super Admin",
      "email": "admin@rate-it.local",
      "is_active": true,
      "roles": [{ "id": 1, "name": "Super Admin" }],
      "permissions": [...]
    },
    "token": "X|plainTextTokenHere..."
  },
  "meta": null
}
```

Step 2: Authenticated Requests

```
Authorization: Bearer X|plainTextTokenHere...
GET /api/v1/admin/auth/me
```

Step 3: Logout

```
POST /api/v1/admin/auth/logout
Authorization: Bearer X|plainTextTokenHere...
```

Response:

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

Admin Model & Roles

Admin Table Schema:

Column	Type	Notes

Column	Type	Notes
`id`	bigint	Primary key
`name`	varchar	Admin's full name
`email`	varchar	Unique email
`phone`	varchar	Contact phone
`password_hash`	varchar	Bcrypt hashed
`role`	enum	'SUPER_ADMIN' or 'ADMIN' (legacy field)
`is_active`	boolean	Account status
`created_at`	timestamp	Creation timestamp
`updated_at`	timestamp	Last update timestamp
`deleted_at`	timestamp	Soft delete

Admin Model Methods:

```
$admin->verifyPassword($plainPassword);      // Hash verification
$admin->roles();                              // morphToMany(Role)
$admin->permissions();                        // Flattened collection of all
permissions
$admin->createToken($name);                   // Sanctum token generation
```

Role-Based Access Control (RBAC)

Tables

Roles Table:

```
id, name (unique), guard, description, created_at, updated_at
```

Permissions Table:

```
id, name (unique), guard, description, created_at, updated_at
```

model_has_roles Table (polymorphic):

```
role_id, model_type (App\Models\Admin), model_id, PRIMARY(role_id, model_type,
model_id)
```

role_has_permissions Table (linking roles to permissions):

```
role_id, permission_id, PRIMARY(role_id, permission_id)
```

Permission Naming Convention

Permissions are named hierarchically:

- `rbac.roles.manage` - Manage roles
- `rbac.permissions.manage` - Manage permissions
- `catalog.categories.create` - Create categories
- `reviews.manage` - Manage reviews (list, show, hide)
- `reviews.reply` - Reply to reviews
- `reviews.feature` - Feature/unfeature reviews
- `users.view` - View users
- `users.block` - Block/unblock users
- `dashboard.view` - View dashboard
- `notifications.templates.view` - View templates
- `notifications.broadcast.send` - Send broadcast
- `points.transactions.view` - View points transactions
- `loyalty.settings.manage` - Manage loyalty settings
- `invites.view` - View invites
- `subscriptions.plans.manage` - Manage subscription plans
- (And more per feature...)

Permission Checking

Middleware: `AdminPermission`

Routes are protected:

```
Route::middleware([
    AdminAuthenticate::class,
    AdminPermission::class . ':reviews.manage'
])->group(function () {
    Route::get('reviews', [ReviewsController::class, 'index']);
    Route::post('reviews/{id}/hide', [ReviewsController::class, 'hide']);
});
```

Middleware Logic:

```
public function handle(Request $request, Closure $next, $permission)
{
    $admin = $request->get('admin');  // Set by AdminAuthenticate
    // Get admin's role IDs
    $roles = DB::table('model_has_roles')
        ->where('model_type', Admin::class)
        ->where('model_id', $admin->id)
        ->pluck('role_id')
        ->toArray();

    // Check if any role has the permission
    $perm = DB::table('permissions')->where('name', $permission)->first();
    $has = DB::table('role_has_permissions')
        ->whereIn('role_id', $roles)
        ->where('permission_id', $perm->id)
        ->exists();

    return $has ? $next($request) : $this->forbidden();
}
```

Common Auth Errors & Responses

Scenario	HTTP Code	Response
Invalid credentials	401	`{success: false, message: "Invalid credentials"}`
Missing token	401	`{success: false, message: "Unauthenticated"}`
Token expired	401	`{success: false, message: "Unauthenticated"}`
Inactive admin	401	`{success: false, message: "Unauthenticated"}`
Missing permission	403	`{success: false, message: "Forbidden"}`

---

6) Database Reference (Admin-Side)

Tables Touched by Admin Features

Core Tables

Table	Purpose	Columns	Relationships
admins	Admin users	id, name, email, phone, password_hash, role, is_active, timestamps, soft_delete	Sanctum tokens, Roles (morphToMany)
roles	Admin roles	id, name (unique), guard, description, timestamps	Permissions, Admin (morphToMany)
permissions	Fine-grained access rules	id, name (unique), guard, description, timestamps	Roles

Table	Purpose	Columns	Relationships
model_has_roles	Assign roles to admins	role_id, model_type, model_id, PRIMARY(role_id, model_type, model_id)	FK→roles
role_has_permissions	Assign permissions to roles	role_id, permission_id, PRIMARY(role_id, permission_id)	FK→roles, FK→permissions
personal_access_tokens	Sanctum tokens (auto-created)	id, tokenable_type, tokenable_id, name, token (hashed), abilities, last_used_at, expires_at, timestamps	Links to admin

Catalog Tables

Table	Purpose	Key Columns	Touched By
categories	Product/service categories	id, name_en, name_ar, is_active, logo, sort_order, timestamps	Catalog module
subcategories	Sub-level categories	id, category_id, name_en, name_ar, is_active, sort_order, timestamps	Catalog module
brands	Brand entities	id, name_en, name_ar, is_active, logo, sort_order, timestamps	Catalog module
places	Business locations	id, name, logo, latitude, longitude, is_active, timestamps	Catalog module
branches	Sub-locations of places	id, place_id, name, address, phone, timestamps	Catalog module
branch_qr_sessions	QR code tracking	id, branch_id, qr_code_hash, is_active, expires_at, timestamps	Catalog module
rating_criteria	Review questions	id, question_en, question_ar, type (RATING/TEXT), is_active, timestamps	Catalog module
rating_criteria_choices	Predefined answers for rating criteria	id, criteria_id, value, order, timestamps	Catalog module
subcategory_rating_criteria	Map subcategories to rating questions	subcategory_id, criteria_id, order, FK(subcategory_id), FK(criteria_id)	Catalog module

Business Feature Tables

Table	Purpose	Key Columns	Admin Access

Table	Purpose	Key Columns	Admin Access
reviews	User reviews	id, user_id, place_id, branch_id, overall_rating, comment, is_hidden, is_featured, admin_reply_text, hidden_by_admin_id, replied_by_admin_id, timestamps	Reviews moderation
review_answers	Answers to rating criteria	id, review_id, criteria_id, choice_id, text_answer, timestamps	Reviews moderation
review_photos	Review images	id, review_id, path, timestamps	Reviews moderation
users	Regular platform users	id, full_name, email, phone, is_blocked, is_active, timestamps	Users management
points_transactions	Points ledger	id, user_id, type (EARN_REVIEW/REDEEM/etc), points, place_id, reason, timestamps	Points monitoring
points_settings	Loyalty point configurations	id, name_en, name_ar, is_active, version, created_by_admin_id, timestamps	Loyalty settings
invites	User referral invites	id, user_id (inviter), invitee_email, status (PENDING/ACCEPTED/REJECTED), timestamps	Invites monitoring
subscription_plans	Subscription tiers	id, name, price, billing_period (monthly/yearly), features_json, is_active, created_by_admin_id, timestamps	Subscriptions
subscriptions	User subscriptions	id, user_id, plan_id, status (ACTIVE/CANCELED), expires_at, timestamps	Subscriptions
notification_templates	Message templates	id, name, channel (email/sms/push), body_template, created_by_admin_id, updated_by_admin_id, timestamps	Notifications
admin_notifications	Sent notifications	id, user_id, template_id, status (SENT/FAILED), channel, timestamps	Notifications (send log)

IMPORTANT Relationships & Constraints

- **Foreign Keys**: All FKs have `onDelete('cascade')` or `onDelete('nullOnDelete')` to maintain referential integrity
- **Unique Constraints**:
  - `email` (admins, users)
  - `name` (roles, permissions, categories, brands)
  - `qr_code_hash` (branch_qr_sessions per place)
- **Soft Deletes**: admins, users, reviews, points_settings use soft delete (`deleted_at`)
- **Polymorphic**: `model_has_roles` allows assigning roles to any model via `model_type` + `model_id`

7) Admin API Reference (Complete)

Admin/Auth Module

POST /api/v1/admin/auth/login

Description: Authenticate admin and receive Sanctum token

Auth Required: No

Request:

```
{
  "email": "admin@rate-it.local",
  "password": "password"
}
```

Response (HTTP 200):

```
{
  "success": true,
  "message": "Login successful",
  "data": {
    "admin": {
      "id": 1,
      "name": "Super Admin",
      "email": "admin@rate-it.local",
      "phone": "+1234567890",
      "role": "SUPER_ADMIN",
      "is_active": true,
      "roles": [
        { "id": 1, "name": "Super Admin", "description": "Full access" }
      ],
      "permissions": [
        { "id": 1, "name": "rbac.roles.manage", "description": "Manage roles" }
      ],
      "timestamps": { "created_at": "...", "updated_at": "...", "deleted_at": null
}
    },
    "token": "X|plainTextTokenHere..."
  },
  "meta": null
}
```

Error (HTTP 401):

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

GET /api/v1/admin/auth/me

Description: Get authenticated admin details

Auth Required: Yes (Bearer token)

Permission: None (all authenticated admins)

Response (HTTP 200):

```json
{
  "success": true,
  "message": "Admin retrieved",
  "data": { /* AdminResource */ },
  "meta": null
}
```

Error (HTTP 401): Unauthenticated

POST /api/v1/admin/auth/logout

Description: Invalidate current token

Auth Required: Yes (Bearer token)

Response (HTTP 200):

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

Admin/Rbac Module

GET /api/v1/admin/roles

Description: List all roles

Auth Required: Yes

Permission: `rbac.roles.manage`

Query Parameters: None

Response (HTTP 200):

```json
{
  "success": true,
  "message": "Roles list",
  "data": [
    { "id": 1, "name": "Super Admin", "guard": "admin", "description": "Full
access", "created_at": "..." },
    { "id": 2, "name": "Category Manager", "guard": "admin", "description":
"Manage catalog", "created_at": "..." }
  ],
  "meta": null
}
```

POST /api/v1/admin/roles

Description: Create new role

Auth Required: Yes

Permission: `rbac.roles.manage`

Request:

```json
{
  "name": "Review Moderator"
}
```

Response (HTTP 201):

```json
{
  "success": true,
  "message": "Role created",
  "data": { "id": 3, "name": "Review Moderator", "guard": "admin", "description":
null, "created_at": "..." },
  "meta": null
}
```

Error (HTTP 422): Validation or duplicate name

POST /api/v1/admin/roles/{role_id}/sync-permissions

Description: Assign/sync permissions to a role (replaces existing)

Auth Required: Yes

Permission: `rbac.roles.manage`

Request:

```
{
  "permissions": [
    "reviews.manage",
    "reviews.reply",
    "reviews.feature"
  ]
}
```

Response (HTTP 200):

```
{
  "success": true,
  "message": "Permissions synced",
  "data": null,
  "meta": null
}
```

GET /api/v1/admin/permissions

Description: List all available permissions

Auth Required: Yes

Permission: `rbac.permissions.manage`

Response (HTTP 200):

```
{
  "success": true,
  "message": "Permissions list",
  "data": [
    { "id": 1, "name": "rbac.roles.manage", "guard": "admin", "description":
"Manage roles" },
    { "id": 2, "name": "reviews.manage", "guard": "admin", "description": "Manage
reviews" }
  ],
  "meta": null
}
```

Admin/Catalog Module

GET /api/v1/admin/categories

Description: List categories with optional filters

Auth Required: Yes

Permission: None (implied by route placement)

Query Parameters:

- `active` (boolean): Filter by is_active status

Response (HTTP 200):

```
{
  "success": true,
  "message": "categories.list",
  "data": [
    {
      "id": 1,
      "name_en": "Food",
      "name_ar": "الطعام",
      "is_active": true,
      "logo": "https://cdn.../logo.png",
      "sort_order": 1,
      "created_at": "..."
    }
  ],
  "meta": null
}
```

POST /api/v1/admin/categories

Description: Create category

Auth Required: Yes

Request:

```
{
  "name_en": "Restaurants",
  "name_ar": "المطاعم",
  "is_active": true,
  "logo": "https://..."
}
```

Response (HTTP 201):

```
{
  "success": true,
  "message": "categories.created",
  "data": { /* CategoryResource */ },
  "meta": null
}
```

GET /api/v1/admin/categories/{id}

Description: Get single category

Response (HTTP 200): CategoryResource

Error (HTTP 404): Not found

PUT /api/v1/admin/categories/{id}

Description: Update category

Request:

```
{
  "name_en": "Fast Food",
  "is_active": false,
  "sort_order": 2
}
```

Response (HTTP 200): CategoryResource

DELETE /api/v1/admin/categories/{id}

Description: Delete category

Response (HTTP 200):

```json
{
  "success": true,
  "message": "categories.deleted",
  "data": null,
  "meta": null
}
```

GET /api/v1/admin/subcategories

Description: List subcategories

Query Parameters:

- `category_id` (int): Filter by category
- `active` (boolean): Filter by status

Response: Array of SubcategoryResource

POST /api/v1/admin/subcategories

GET /api/v1/admin/subcategories/{id}

PUT /api/v1/admin/subcategories/{id}

DELETE /api/v1/admin/subcategories/{id}

(Same pattern as categories)

GET /api/v1/admin/brands

POST /api/v1/admin/brands

GET /api/v1/admin/brands/{id}

PUT /api/v1/admin/brands/{id}

DELETE /api/v1/admin/brands/{id}

(Same REST pattern as categories)

GET /api/v1/admin/places

Description: List places with filters

Query Parameters:

- `active` (boolean)
- `search` (string): Search by name or location

Response: Paginated array of PlaceResource

POST /api/v1/admin/places

Request:

```json
{
  "name": "Downtown Mall",
  "latitude": 40.7128,
  "longitude": -74.0060,
  "logo": "https://...",
  "is_active": true
}
```

GET /api/v1/admin/places/{id}

PUT /api/v1/admin/places/{id}

DELETE /api/v1/admin/places/{id}

GET /api/v1/admin/branches

Description: List branches

Query Parameters:

- `place_id` (int): Filter by place
- `active` (boolean)

Response: Paginated array of BranchResource

POST /api/v1/admin/branches

Request:

```json
{
  "place_id": 1,
  "name": "Main Branch",
  "address": "123 Main St",
  "phone": "+1234567890"
}
```

GET /api/v1/admin/rating-criteria

Description: List rating criteria (questions)

Query Parameters:

- `active` (boolean)
- `type` (string): RATING or TEXT

Response: Paginated array of RatingCriteriaResource

POST /api/v1/admin/rating-criteria

Request:

```
{
  "question_en": "How was the service?",
  "question_ar": "كيف كانت الخدمة؟",
  "type": "RATING",
  "is_active": true
}
```

GET /api/v1/admin/rating-criteria-choices

POST /api/v1/admin/rating-criteria-choices

(For managing predefined answers)

GET /api/v1/admin/subcategories/{id}/rating-criteria

Description: List rating criteria assigned to a subcategory

Response:

```
{
  "success": true,
  "message": "...",
  "data": [
    {
      "id": 1,
      "question_en": "Quality?",
      "question_ar": "...",
      "order": 1,
      "choices": [ { "id": 1, "value": "Excellent" } ]
    }
  ],
  "meta": null
}
```

POST /api/v1/admin/subcategories/{id}/rating-criteria/sync

Description: Assign rating criteria to subcategory

Request:

```
{
  "criteria_ids": [1, 3, 5]
}
```

Response (HTTP 200): Sync successful

POST /api/v1/admin/subcategories/{id}/rating-criteria/reorder

Description: Reorder rating criteria for subcategory

Request:

```
{
  "orders": [
    { "criteria_id": 1, "order": 1 },
    { "criteria_id": 3, "order": 2 }
  ]
}
```

DELETE /api/v1/admin/subcategories/{id}/rating-criteria/{criteria_id}

Description: Remove rating criteria from subcategory

Admin/Dashboard Module

GET /api/v1/admin/dashboard/summary

Description: Get KPI summary (users count, reviews, avg rating, points metrics)

Auth Required: Yes

Permission: `dashboard.view`

Query Parameters (optional):

- `from` (date): Start date (YYYY-MM-DD)
- `to` (date): End date (YYYY-MM-DD)

Response (HTTP 200):

```json
{
  "success": true,
  "message": "admin.dashboard.summary",
  "data": {
    "users_count": 1250,
    "reviews_count": 3450,
    "avg_rating": 4.25,
    "points_issued_total": 125000,
    "points_redeemed_total": 35000
  },
  "meta": {
    "from": "2026-01-01",
    "to": "2026-01-25"
  }
}
```

GET /api/v1/admin/dashboard/top-places

Description: Get top places ranked by metric

Permission: `dashboard.view`

Query Parameters:

- `metric` (string): 'reviews_count', 'avg_rating', 'points_issued' (default: 'reviews_count')
- `limit` (int): 1-50 (default: 10)
- `min_reviews` (int): Minimum review count to include (default: 1)
- `category_id` (int): Filter by category
- `from`, `to` (date): Date range

Response (HTTP 200):

```json
{
  "success": true,
  "message": "admin.dashboard.top_places",
  "data": [
    {
      "id": 1,
      "name": "Downtown Mall",
      "reviews_count": 345,
      "avg_rating": 4.6,
      "points_issued": 12000
    }
  ],
  "meta": null
}
```

GET /api/v1/admin/dashboard/reviews-chart

Description: Get timeseries chart of reviews count and avg rating

Permission: `dashboard.view`

Query Parameters:

- `from` (date, required): Start date
- `to` (date, required): End date
- `interval` (string): 'day', 'week', 'month' (auto-selected if omitted)
- `place_id`, `branch_id`, `category_id` (int): Optional filters

Response (HTTP 200):

```
{
  "success": true,
  "message": "admin.dashboard.reviews_chart",
  "data": {
    "interval": "day",
    "dataPoints": [
      { "date": "2026-01-01", "reviews_count": 10, "avg_rating": 4.2 },
      { "date": "2026-01-02", "reviews_count": 15, "avg_rating": 4.5 }
    ]
  },
  "meta": null
}
```

Admin/Reviews Module

GET /api/v1/admin/reviews

Description: List reviews with filters and search

Auth Required: Yes

Permission: `reviews.manage`

Query Parameters:

- `page` (int): Page number (default: 1)
- `per_page` (int): Items per page (default: 15)
- `place_id`, `branch_id`, `user_id` (int): Filters
- `date_from`, `date_to` (date): Date range

- `rating_min`, `rating_max` (float): Rating range
- `is_hidden` (boolean): Filter by hidden status
- `is_featured` (boolean): Filter by featured status
- `q` (string): Search in comment, user name, place name

Response (HTTP 200):

```
{
  "success": true,
  "message": "admin.reviews.list",
  "data": [
    {
      "id": 1,
      "user": { "id": 5, "full_name": "John Doe", "phone": "+1234567890" },
      "place": { "id": 2, "name": "Downtown Mall", "logo": "..." },
      "overall_rating": 4.5,
      "comment": "Great place!",
      "is_hidden": false,
      "is_featured": true,
      "answers_count": 3,
      "photos_count": 2,
      "created_at": "..."
    }
  ],
  "meta": {
    "page": 1,
    "limit": 15,
    "total": 3450,
    "has_next": true,
    "last_page": 230
  }
}
```

GET /api/v1/admin/reviews/{id}

Description: Get review details

Response (HTTP 200):

```
{
  "success": true,
  "message": "admin.reviews.details",
  "data": {
    "id": 1,
    "user": { /* full user */ },
    "place": { /* place */ },
    "branch": { /* branch */ },
    "overall_rating": 4.5,
    "comment": "Great place!",
    "is_hidden": false,
    "hidden_reason": null,
    "is_featured": true,
    "admin_reply_text": "Thank you!",
    "replied_at": "2026-01-20T10:00:00Z",
    "replied_by_admin_id": 1,
    "answers": [
      {
        "id": 1,
        "criteria": { "id": 1, "question_en": "Service?" },
        "choice": { "id": 1, "value": "Excellent" },
        "text_answer": null
      }
    ],
    "photos": [
      { "id": 1, "path": "/storage/photos/review_1_photo_1.jpg" }
    ],
    "created_at": "..."
  },
  "meta": null
}
```

POST /api/v1/admin/reviews/{id}/hide

Description: Hide/unhide review

Permission: `reviews.manage`

Request:

```
{
  "is_hidden": true,
  "reason": "Inappropriate content"
}
```

Response (HTTP 200): AdminReviewResource

POST /api/v1/admin/reviews/{id}/reply

Description: Reply to review

Permission: `reviews.reply`

Request:

```
{
  "reply_text": "Thank you for your feedback!"
}
```

Response (HTTP 200): AdminReviewResource

POST /api/v1/admin/reviews/{id}/mark-featured

Description: Mark review as featured

Permission: `reviews.feature`

Request:

```
{
  "is_featured": true
}
```

Response (HTTP 200): AdminReviewResource

Admin/Users Module

GET /api/v1/admin/users

Description: List users

Permission: `users.view`

Query Parameters:

- `page`, `per_page`: Pagination
- `is_blocked` (boolean): Filter
- `q` (string): Search by name or email
- `date_from`, `date_to`: Registration date range

Response: Paginated AdminUserResource[]

GET /api/v1/admin/users/{id}

Description: Get user details

Permission: `users.view`

Response (HTTP 200):

```
{
  "success": true,
  "message": "admin.users.details",
  "data": {
    "id": 1,
    "full_name": "John Doe",
    "email": "john@example.com",
    "phone": "+1234567890",
    "is_active": true,
    "is_blocked": false,
    "created_at": "...",
    "updated_at": "..."
  },
  "meta": null
}
```

POST /api/v1/admin/users/{id}/block

Description: Block/unblock user

Permission: `users.block`

Request:

```
{
  "is_blocked": true
}
```

Response (HTTP 200): AdminUserResource

GET /api/v1/admin/users/{id}/reviews

Description: Get user's reviews

Permission: `users.reviews.view`

Query Parameters:

- `page`, `per_page`: Pagination

Response: Paginated array of reviews

GET /api/v1/admin/users/{id}/points

Description: Get user's points balance and transactions

Permission: `users.points.view`

Response (HTTP 200):

```
{
  "success": true,
  "message": "admin.users.points",
  "data": {
    "user_id": 1,
    "balance": 5000,
    "transactions": [
      { "id": 1, "type": "EARN_REVIEW", "points": 100, "created_at": "..." }
    ]
  },
  "meta": null
}
```

Admin/Points Module

GET /api/v1/admin/points/transactions

Description: List all points transactions

Permission: `points.transactions.view`

Query Parameters:

- `page`, `per_page`: Pagination
- `user_id` (int): Filter by user
- `type` (string): EARN_REVIEW, REDEEM, etc.
- `date_from`, `date_to`: Date range

Response (HTTP 200):

```
{
  "success": true,
  "message": "admin.points.transactions.list",
  "data": [
    {
      "id": 1,
      "user_id": 5,
      "type": "EARN_REVIEW",
      "points": 100,
      "place_id": 2,
      "reason": "Posted review",
      "created_at": "..."
    }
  ],
  "meta": { "page": 1, "limit": 15, "total": 5000 }
}
```

GET /api/v1/admin/points/transactions/{id}

Description: Get single transaction

Response (HTTP 200): AdminPointsTransactionResource

Admin/LoyaltySettings Module

GET /api/v1/admin/loyalty-settings

Description: List loyalty point settings versions

Permission: `loyalty.settings.view`

Response (HTTP 200):

```
{
  "success": true,
  "message": "admin.loyalty_settings.list",
  "data": [
    {
      "id": 1,
      "version": 1,
      "name_en": "Default Settings",
      "name_ar": "الإعدادات الافتراضية",
      "points_per_review": 100,
      "points_per_invite": 50,
      "is_active": false,
      "created_by_admin_id": 1,
      "created_at": "..."
    }
  ],
  "meta": { "page": 1, "limit": 15, "total": 3 }
}
```

POST /api/v1/admin/loyalty-settings

Description: Create new settings version (immutable; creates new record)

Permission: `loyalty.settings.manage`

Request:

```
{
  "name_en": "Premium Settings",
  "name_ar": "إعدادات الخصم",
  "points_per_review": 150,
  "points_per_invite": 75
}
```

Response (HTTP 201): AdminLoyaltySettingResource

POST /api/v1/admin/loyalty-settings/{id}/activate

Description: Activate a settings version (deactivates previous)

Permission: `loyalty.settings.manage`

Response (HTTP 200): AdminLoyaltySettingResource

Admin/Notifications Module

GET /api/v1/admin/notifications/templates

Description: List notification templates

Permission: `notifications.templates.view`

Query Parameters:

- `page`, `per_page`: Pagination
- `channel` (string): email, sms, push (filter)

Response (HTTP 200):

```json
{
  "success": true,
  "message": "admin.notifications.templates.list",
  "data": [
    {
      "id": 1,
      "name": "Welcome Email",
      "channel": "email",
      "body_template": "Welcome {{name}}!",
      "created_by_admin_id": 1,
      "created_at": "..."
    }
  ],
  "meta": { "page": 1, "limit": 15, "total": 10 }
}
```

POST /api/v1/admin/notifications/templates

Description: Create notification template

Permission: `notifications.templates.manage`

Request:

```
{
  "name": "New Review Alert",
  "channel": "email",
  "body_template": "You have a new review: {{comment}}"
}
```

Response (HTTP 201): AdminNotificationTemplateResource

PUT /api/v1/admin/notifications/templates/{id}

Description: Update template

Response (HTTP 200): AdminNotificationTemplateResource

POST /api/v1/admin/notifications/broadcast

Description: Send broadcast notification to all/filtered users

Permission: `notifications.broadcast.send`

Request:

```
{
  "template_id": 1,
  "filters": {
    "min_points": 100,
    "is_blocked": false
  }
}
```

Response (HTTP 200):

```
{
  "success": true,
  "message": "admin.notifications.broadcast.sent",
  "data": {
    "sent_count": 1250,
    "failed_count": 5,
    "total_targeted": 1255
  },
  "meta": null
}
```

POST /api/v1/admin/users/{id}/notifications

Description: Send notification to single user

Permission: `notifications.user.send`

Request:

```
{
  "template_id": 1,
  "variables": { "name": "John" }
}
```

Response (HTTP 200):

```
{
  "success": true,
  "message": "Notification sent",
  "data": { "notification_id": 1, "status": "SENT" },
  "meta": null
}
```

Admin/Invites Module

GET /api/v1/admin/invites

Description: List user invitations

Permission: `invites.view`

Query Parameters:

- `page`, `per_page`: Pagination
- `status` (string): PENDING, ACCEPTED, REJECTED
- `user_id` (int): Filter by inviter
- `date_from`, `date_to`: Date range

Response (HTTP 200):

```
{
  "success": true,
  "message": "admin.invites.list",
  "data": [
    {
      "id": 1,
      "user_id": 5,
      "user_name": "Jane Doe",
      "invitee_email": "invitee@example.com",
      "status": "ACCEPTED",
      "created_at": "..."
    }
  ],
  "meta": {
    "page": 1,
    "limit": 15,
    "total": 500,
    "status_counts": {
      "PENDING": 100,
      "ACCEPTED": 350,
      "REJECTED": 50
    }
  }
}
```

GET /api/v1/admin/invites/{id}

Description: Get invite details

Response (HTTP 200): AdminInviteResource

Admin/Subscriptions Module

GET /api/v1/admin/subscription-plans

Description: List subscription plans

Permission: `subscriptions.plans.view`

Query Parameters:

- `page`, `per_page`: Pagination
- `active` (boolean): Filter by active status

Response (HTTP 200):

```
{
  "success": true,
  "message": "admin.subscriptions.plans.list",
  "data": [
    {
      "id": 1,
      "name": "Basic",
      "price": 9.99,
      "billing_period": "monthly",
      "features": ["Feature 1", "Feature 2"],
      "is_active": true,
      "created_by_admin_id": 1,
      "created_at": "..."
    }
  ],
  "meta": { "page": 1, "limit": 15, "total": 5 }
}
```

POST /api/v1/admin/subscription-plans

Description: Create subscription plan

Permission: `subscriptions.plans.manage`

Request:

```
{
  "name": "Premium",
  "price": 29.99,
  "billing_period": "monthly",
  "features": ["Advanced features", "Priority support"]
}
```

Response (HTTP 201): AdminSubscriptionPlanResource

PUT /api/v1/admin/subscription-plans/{id}

Description: Update plan

Response (HTTP 200): AdminSubscriptionPlanResource

POST /api/v1/admin/subscription-plans/{id}/activate

Description: Activate plan

Permission: `subscriptions.plans.manage`

Response (HTTP 200): AdminSubscriptionPlanResource

GET /api/v1/admin/subscriptions

Description: List active subscriptions

Permission: `subscriptions.view`

Query Parameters:

- `page`, `per_page`: Pagination
- `user_id` (int): Filter by user
- `plan_id` (int): Filter by plan
- `status` (string): ACTIVE, CANCELED

Response (HTTP 200): Paginated SubscriptionResource[]

---

8) Feature Deep Dives (Admin)

Admin/Auth — Authentication & Account Management

Purpose: Secure admin login, session management, and admin account lifecycle.

Core Files:

- AuthController.php
- AuthService.php
- AdminResource.php

Flows:

1. **Login**: Admin submits email + password → AuthService verifies hash → Sanctum token generated
2. **Me**: Admin requests authenticated details → middleware validates token → returns AdminResource with roles/permissions
3. **Logout**: Client sends token → service hashes and deletes from `personal_access_tokens`

Key Classes:

- `Admin` model (app/Models/Admin.php): `verifyPassword()`, `createToken()`, role morphing
- `AuthService`: login logic, token generation/revocation
- `AdminAuthenticate` middleware: validates bearer token, attaches admin to request

Database:

- `admins`: name, email, password_hash, is_active, role (legacy enum)
- `personal_access_tokens`: Sanctum-managed token storage

Validations:

- Email must be valid format
- Password required (8+ chars recommended)
- Admin must exist and is_active=true

Edge Cases:

- Token expiration: Not set by default; tokens persist until logout or manual deletion
- Inactive admins: Rejected even with valid token
- Password case-sensitive (bcrypt)

Admin/Rbac — Role-Based Access Control

Purpose: Define roles, assign permissions, enforce granular access to features.

Core Files:

- RolesController.php
- PermissionsController.php
- AdminPermission.php (middleware)

Flows:

1. **Roles Management**: Create/list roles; sync permissions to roles
2. **Permission Checking**: AdminPermission middleware checks if admin's roles have the required permission
3. **Assignment**: Admins assigned to roles via model_has_roles (polymorphic)

Permission Naming Hierarchy:

- `rbac.*` - RBAC management
- `catalog.*` - Catalog feature
- `reviews.*` - Review moderation
- `users.*` - User management
- `dashboard.*` - Analytics
- `notifications.*` - Notifications
- `points.*` - Points monitoring
- `loyalty.*` - Loyalty settings
- `subscriptions.*` - Subscription plans
- `invites.*` - Invites monitoring

Database Tables:

- `roles`: id, name (unique), guard, description
- `permissions`: id, name (unique), guard, description
- `model_has_roles`: (role_id, model_type='App\Models\Admin', model_id) — PRIMARY(role_id, model_type, model_id)
- `role_has_permissions`: (role_id, permission_id) — PRIMARY(role_id, permission_id)

Permission Enforcement:

```
// Route-level (middleware)
Route::middleware([AdminAuthenticate::class,
AdminPermission::class.':reviews.manage'])
     ->post('reviews/{id}/hide', ...)

// Middleware checks:
$roles = DB::table('model_has_roles')
    ->where('model_type', Admin::class)
    ->where('model_id', $admin->id)
    ->pluck('role_id');
```

```
$perm = DB::table('permissions')->where('name', $permission)->first();
$has = DB::table('role_has_permissions')
    ->whereIn('role_id', $roles)
    ->where('permission_id', $perm->id)
    ->exists();
```

Key Validations:

- Role name must be unique
- Permission name must exist in DB
- Admin must have at least one role to access protected routes

Important Notes:

- Permissions are seeded during migration/seeding; not all created via API
- No "super admin" flag; instead, assign all permissions to Super Admin role
- Role deletion cascades to remove associated permissions and admin assignments

---

Admin/Catalog — Categories, Brands, Places, Branches, Rating Criteria

Purpose: Master data management for the review platform's taxonomy and locations.

Core Files:

- CategoriesController.php + Service
- BrandsController.php + Service
- PlacesController.php + Service
- BranchesController.php + Service
- RatingCriteriaController.php + Service

Flows:

1. **Categories/Subcategories**: Admin creates top-level categories → assigns subcategories → assigns rating criteria
2. **Brands**: Create brand entries (bilingual, with logo)
3. **Places**: Create place locations with coordinates → assign categories/subcategories
4. **Branches**: Create sub-locations under places → each with unique QR code for in-venue reviews
5. **Rating Criteria**: Define review questions (RATING or TEXT type) → assign choices → link to subcategories

Key Classes:

- `Category` model: bilingual (name_en, name_ar), is_active, sort_order
- `Subcategory` model: category_id FK, bilingual
- `Brand` model: bilingual, logo, sort_order
- `Place` model: name, latitude, longitude, logo, categories/subcategories relationship
- `Branch` model: place_id FK, name, address, phone, soft delete
- `RatingCriteria` model: bilingual question, type (RATING|TEXT), is_active
- `RatingCriteriaChoice` model: criteria_id FK, value (answer option)

- `SubcategoryRatingCriteria` pivot: subcategory_id, criteria_id, order

Database:

- `categories`, `subcategories`, `brands`, `places`, `branches`, `rating_criteria`, `rating_criteria_choices`
- Pivot table: `subcategory_rating_criteria` (for managing which questions belong to which subcategories)

Key Validations:

- Category/brand names must be unique
- Subcategory must belong to existing category
- Place coordinates must be valid lat/long (if provided)
- Branch name must be unique per place (or globally, check code)
- Rating criteria type must be RATING or TEXT
- Ordering constraints: sort_order should be sequential

Important Relationships:

- Place → Brands (many-to-many)
- Place → Categories/Subcategories (many-to-many)
- Subcategory → RatingCriteria (many-to-many via subcategory_rating_criteria, ordered)
- Branch → QR sessions (one-to-many)

Edge Cases:

- Deleting a category cascades to subcategories → review failures if reviews reference those
- Changing rating criteria assignments affects which questions users see when reviewing
- QR code regeneration (per branch) should invalidate previous codes

---

Admin/Dashboard — Analytics & KPIs

Purpose: High-level platform metrics, top performers, and trends visualization.

Core Files:

- DashboardController.php
- DashboardService.php

Flows:

1. **Summary KPIs**: Count users, reviews; average rating; points issued/redeemed (optionally filtered by date)
2. **Top Places**: Rank places by reviews_count, avg_rating, or points_issued; with optional date range
3. **Reviews Chart**: Timeseries data (daily/weekly/monthly) of review count and average rating trends

Key Methods:

- `getSummary(filters)`: Returns {users_count, reviews_count, avg_rating, points_issued_total, points_redeemed_total}

- `getTopPlaces(filters)`: Returns top places with pagination and metric selection
- `getReviewsChart(filters)`: Returns timeseries array with date, reviews_count, avg_rating

Database Queries:

- Reviews, Users, PointsTransactions tables
- Aggregations: COUNT(), AVG(), SUM()
- Date range filtering via `whereDate()`, `where('created_at', '>=', ...)`

Important Logic:

- Points issued: WHERE points > 0
- Points redeemed: WHERE points < 0 (absolute value)
- Average rating: Only include non-hidden reviews
- Interval selection: Auto-select day/week/month based on date range width

Edge Cases:

- Empty date range: May return 0s
- No reviews: avg_rating = 0.0
- Interval mismatch: If range is 1 day but interval='month', may return empty chart

---

Admin/Reviews — Moderation & Management

Purpose: Monitor, moderate, and engage with user-submitted reviews.

Core Files:

- ReviewsController.php
- ReviewModerationService.php
- AdminReviewResource.php

Flows:

1. **List Reviews**: Paginated list with filters (place, user, hidden status, date range, rating range, search)
2. **Show Details**: Get full review with answers, photos, admin history (hidden_by, replied_by)
3. **Hide/Unhide**: Mark review as inappropriate or restore it
4. **Reply**: Add admin response to review (visible to users)
5. **Feature**: Mark review as featured/showcased (pinned, highlighted)

Key Methods:

- `list(filters)`: Returns LengthAwarePaginator with complex filtering
- `find(id)`: Eager-load review with user, place, branch, answers, photos
- `hide(id, data)`: Set is_hidden, hidden_reason, hidden_at, hidden_by_admin_id
- `reply(id, data)`: Set admin_reply_text, replied_at, replied_by_admin_id
- `feature(id, data)`: Toggle is_featured flag

Database:

- `reviews`: user_id, place_id, branch_id, overall_rating, comment, is_hidden, is_featured, admin_reply_text, hidden_by_admin_id, replied_by_admin_id, timestamps
- `review_answers`: review_id, criteria_id, choice_id, text_answer
- `review_photos`: review_id, path

Filtering Logic:

- Full-text search: comment, user.full_name, user.phone, place.name, branch.name (via whereHas)
- Date filtering: whereDate('created_at', '>=', from) / <= to
- Rating range: where('overall_rating', '>=', min) / <= max
- Ordering: desc by created_at (newest first)

Important Constraints:

- Hiding requires reason field (optional but recommended)
- Admin ID tracked for audit trail
- is_featured and is_hidden are independent flags (can be both true)

Edge Cases:

- Deleted users: Reviews still visible (user_id FK no onDelete)
- Missing branch: Branch may be deleted after review posted
- Concurrent edits: Last write wins (no locking)

Admin/Users — User Management & Blocking

Purpose: Monitor user accounts, view user-specific metrics, and manage user access.

Core Files:

- UsersController.php
- UserAdminService.php

Flows:

1. **List Users**: Paginated list with search and filtering
2. **Show User**: Get user profile and account status
3. **Block/Unblock**: Disable/enable user account access
4. **User Reviews**: List reviews posted by user
5. **User Points**: Display points balance and transaction history

Key Methods:

- `list(filters)`: Paginated users with search (name, email)
- `find(id)`: User details
- `block(id, data)`: Set is_blocked flag
- `reviews(id, filters)`: User's reviews paginated
- `points(id, filters)`: User's points balance and transactions

Database:

- `users`: id, full_name, email, phone, is_blocked, is_active, timestamps
- Related: points_transactions, reviews

Important Logic:

- Blocked users can't post reviews or redeem points (enforced elsewhere, not here)
- Points balance: Sum of all transactions for that user
- is_active vs is_blocked: Both flags exist (is_active may be system-set, is_blocked admin-set)

Edge Cases:

- Blocking user doesn't delete their data
- User review count shown; photos/answers can be queried separately

---

Admin/Points — Points Transaction Monitoring

Purpose: Track all points ledger entries and ensure system integrity.

Core Files:

- PointsController.php
- PointsAdminService.php

Flows:

1. **List Transactions**: View all points transactions with filtering
2. **Show Transaction**: Get details of single transaction

Key Methods:

- `list(filters)`: Paginated transactions
- `find(id)`: Single transaction details

Database:

- `points_transactions`: id, user_id, type (EARN_REVIEW, REDEEM, etc.), points (can be negative), place_id, reason, timestamps

Filtering:

- By user_id, type, date range
- Can show earned (points > 0) vs redeemed (points < 0)

Important Note: This is read-only; points are issued by other modules (Reviews, Invites, etc.)

Admin/LoyaltySettings — Points Configuration

Purpose: Manage point issuance rules and loyalty program parameters (versioned, immutable).

Core Files:

- LoyaltySettingsController.php

- LoyaltySettingsService.php

Flows:

1. **List Settings**: View all configuration versions
2. **Create Settings**: Create new version (doesn't modify existing; creates new record)
3. **Activate Settings**: Activate a version (deactivates previous)

Key Logic:

- Settings are **immutable**: Once created, can't be edited; instead, create new version
- Only one version can be active at a time
- `version` field tracks iteration count
- created_by_admin_id for audit trail

Database:

- `points_settings`: id, name_en, name_ar, version, points_per_review, points_per_invite, currency, is_active, created_by_admin_id, timestamps

Important Constraints:

- Activation: Must deactivate any previous is_active=true record
- Field scope: Check actual migration for exact columns

Edge Cases:

- Gap in versions if rollback: OK, not enforced to be sequential
- No active version: System falls back to default (check app logic)

Admin/Notifications — Templates, Broadcast, & User Notifications

Purpose: Manage notification templates and send announcements/targeted messages to users.

Core Files:

- TemplatesController.php
- TemplatesService.php
- BroadcastController.php
- BroadcastService.php
- SendController.php
- SendService.php

Flows:

1. **Templates**: Create/list/update message templates (email/sms/push) with variable placeholders
2. **Broadcast**: Send template to all users or filtered subset
3. **Send to User**: Send template to single user

Key Classes:

- `TemplatesService`: Template CRUD

- `BroadcastService`: Bulk send (may use queues)
- `SendService`: Single user send

Database:

- `notification_templates`: id, name, channel (email|sms|push), body_template (with {{placeholder}} syntax), created_by_admin_id, updated_by_admin_id, timestamps
- `admin_notifications`: id, user_id, template_id, status (SENT|FAILED|PENDING), channel, timestamps

Important Logic:

- Templates support variable substitution ({{name}}, {{points}}, etc.)
- Channels (email, sms, push) may be configured separately
- Broadcast may be async (queued) for large volume
- Filters: min_points, is_blocked, category_id, etc. (check actual code)

Edge Cases:

- Invalid placeholders: System should warn or skip
- Undeliverable emails: May retry or log failure
- Queue failures: May require manual retry

Admin/Invites — Invitation Campaign Monitoring

Purpose: Monitor user-to-user invitation system and track referral status.

Core Files:

- InvitesController.php
- InvitesService.php

Flows:

1. **List Invites**: View all invitations with status filtering
2. **Show Invite**: Get details of single invitation
3. **Status Counts**: Meta data showing breakdown by status

Key Methods:

- `list(filters)`: Paginated invites with status, user, date range filters
- `find(id)`: Single invite details
- `statusCounts()`: Summary counts per status

Database:

- `invites`: id, user_id (inviter), invitee_email, status (PENDING|ACCEPTED|REJECTED), timestamps

Important: Admins can only view; invites are created by users (elsewhere in codebase).

Admin/Subscriptions — Plans & Monitoring

Purpose: Manage subscription offerings and monitor active subscriptions.

Core Files:

- PlansController.php
- PlansService.php
- SubscriptionsController.php
- SubscriptionsService.php

Flows:

1. **Plans**: Create/list/update subscription plans
2. **Activate Plan**: Activate a plan (may deactivate others)
3. **Subscriptions**: View active user subscriptions

Key Classes:

- `PlansService`: Plan CRUD + activation
- `SubscriptionsService`: Monitoring subscriptions

Database:

- `subscription_plans`: id, name, price, billing_period (monthly|yearly), features_json, is_active, created_by_admin_id, timestamps
- `subscriptions`: id, user_id, plan_id, status (ACTIVE|CANCELED|EXPIRED), expires_at, timestamps

Important Logic:

- features_json: JSON array of feature strings
- is_active: Only active plans can be purchased (likely enforced in user module)
- expires_at: Can be null for indefinite or auto-renewing

Edge Cases:

- Deactivating plan doesn't affect existing subscriptions
- Price change: Applies only to new subscriptions, not retroactively
- Timezone handling for expires_at: Check if UTC or user timezone

---

9) Background Jobs, Events, Notifications (Admin)

Queued Jobs Used by Admin

Admin module doesn't directly queue jobs, but downstream operations may:

- **Broadcast notifications**: May queue SendNotificationJob for large volume
- **Bulk operations**: Catalog import/export could use batch jobs

Actual queue implementation: Check `config/queue.php` and `app/Jobs/`

Events/Listeners

Not explicitly documented in Admin routes. Check event/listener mappings if used for:

- Review hidden/featured events
- User blocked events
- Points transaction events

Notification Channels

Admin can send via:

- **Email** (via MAIL_MAILER config)
- **SMS** (if configured; check config/services.php)
- **Push** (if push service enabled)

Required Workers/Cron

If using queues:

```
# Long-running worker
php artisan queue:work --queue=default --timeout=300

# Or supervisor (production)
[program:queue-worker]
command=php /app/artisan queue:work --queue=default
```

For scheduled tasks (if any):

```
# In Kernel.php, check if Admin scheduling exists
php artisan schedule:run
```

---

10) Observability & Operations

Logging

Configuration: `config/logging.php`

Channels: stack, single, daily

Admin Context: No specific admin logging layer; relies on Laravel defaults

- Logs location: `storage/logs/`
- Format: Single, daily rotation, or stack (multiple channels)
- Log level: Set via `LOG_LEVEL` env var (debug, info, warning, error)

Audit Trail: Partial in code

- `hidden_by_admin_id`, `replied_by_admin_id`, `created_by_admin_id` fields track admin actions
- Not a separate audit log table; recommend adding if compliance needed

Error Handling

Strategy: Standard Laravel exception handling

- FormRequest validation → 422 response with error details
- Model not found → 404 via controller
- Permission denied → 403 via middleware
- Unauthenticated → 401 via middleware

Custom Exceptions: Check `app/Exceptions/Handler.php` for custom mappings

HTTP Codes:

Code	Use Case
200	Success, OK
201	Resource created
400	Bad request (generic)
401	Unauthenticated
403	Forbidden (permission denied)
404	Not found
422	Validation error
500	Server error

Metrics/Tracing

Currently: Not built-in. If needed, integrate:

- Laravel Telescope (dev)
- APM tools (New Relic, DataDog, etc.)
- Custom instrumentation via events

Health Checks

Not implemented in Admin module. If needed, add:

```
Route::get('/health', function () {
    return response()->json(['status' => 'ok']);
});
```

---

11) Testing & QA Notes

Run Tests

```
# All tests
composer test

# Admin tests only
composer test:admin

# Specific test file
php artisan test tests/Feature/Admin/Auth/AuthTest.php

# Specific test method
php artisan test --filter test_admin_login_with_valid_credentials

# Watch mode (if available)
composer test:watch
```

Test Structure

Located in `tests/Feature/Admin/`

Test Coverage:

- Auth: Login, logout, me endpoint, inactive admin rejection
- Rbac: Role creation, permission listing, permission syncing
- Catalog: CRUD for categories, subcategories, brands, places, branches, rating criteria
- Reviews: Listing, filtering, hiding, replying, featuring
- Users: Listing, blocking, user reviews, user points
- Dashboard: Summary KPIs, top places, charts
- Points: Transaction listing
- Loyalty: Settings creation and activation
- Notifications: Template CRUD, broadcast, single send
- Invites: Listing with status filters
- Subscriptions: Plan CRUD, subscriptions monitoring

Common Test Patterns:

```
// Setup: Create admin with token
$admin = Admin::factory()->create();
$token = $admin->createToken('test')->plainTextToken;

// Request with token
$response = $this->withToken($token)->get('/api/v1/admin/...');

// Assert response
$response->assertStatus(200)->assertJson(['success' => true]);
```

Postman Collection

Located in `postman/admin/`

Usage:

1. Import collection into Postman
2. Set environment variables (token, base_url)
3. Run requests sequentially (token flows from login)
4. Use collection runner for regression testing

Test Accounts (Seeded)

Check `database/seeders/AdminSeeder.php` for default accounts:

- Super Admin: email, password (usually: admin@rate-it.local / password)
- Regular Admin: (if created)

Note: Do not commit test accounts with known passwords; use factory-generated in tests.

Suggested Test Scenarios Per Feature

Feature	Scenario
Auth	Login with valid creds, invalid email, wrong password, inactive admin, logout token invalidation
Catalog	Create/edit/delete categories; unique name validation; subcategory category FK; place coordinates validation
Reviews	List with filters (place, user, date, rating, hidden status); hide with reason; reply; feature
Users	List with search; show user; block/unblock toggle; view user reviews paginated; view user points balance
Dashboard	Summary KPIs with and without date range; top places by different metrics; chart timeseries
Permissions	Attempt access to endpoint without permission (403); with permission (200); role assignment
Concurrency	Two admins editing same resource; last write wins

12) Deployment Notes (Admin Impact)

Environment Variables Required (Production)

```
# Core
APP_ENV=production
APP_DEBUG=false
APP_KEY=base64:...
APP_URL=https://api.rate-it.com

# Database
DB_CONNECTION=mysql
DB_HOST=db.prod.example.com
```

```
DB_DATABASE=rate_it_prod
DB_USERNAME=app_user
DB_PASSWORD=<secure-password>

# Authentication
SANCTUM_STATEFUL_DOMAINS=api.rate-it.com,admin.rate-it.com

# Mail (for notifications)
MAIL_MAILER=smtp
MAIL_HOST=smtp.sendgrid.net
MAIL_PORT=587
MAIL_USERNAME=apikey
MAIL_PASSWORD=<sendgrid-key>
MAIL_FROM_ADDRESS=noreply@rate-it.com

# Logging
LOG_CHANNEL=stack
LOG_LEVEL=warning

# Queue
QUEUE_CONNECTION=redis
REDIS_HOST=redis.prod.example.com
REDIS_PASSWORD=<redis-password>
```

Migrations Order & Safety

```
# Before deployment
php artisan down --secret=<token>  # Maintenance mode

# Run migrations (in order)
php artisan migrate --force

# Optional: seed only if adding new data
php artisan db:seed --class=AdminSeeder

# Back online
php artisan up
```

Safety Checks:

- Backup database before migration
- Test migrations in staging first
- Use `--force` only in production (no confirmation)
- Verify data integrity post-migration

Migration Files (order matters):

1. Admins table
2. Roles and permissions tables (model_has_roles, role_has_permissions)
3. Catalog tables (categories, subcategories, brands, places, branches, rating_criteria)

4. Business tables (reviews, points_transactions, subscription_plans, etc.)

Zero-Downtime Considerations

- Sanctum tokens survive deployments (persisted in DB)
- No session state stored in code; safe to restart
- Queued jobs may delay during restart; consider supervisor management
- DB schema changes: Plan backwards compatibility (add column before removing)

Storage/Queue Dependencies

- **Queue**: Redis or database (configured in .env)
- **File Storage**: Local disk or S3 (configured in config/filesystems.php)
- Admin module assumes files persisted (logos, photos)

Backward Compatibility Concerns

- Permission names: If changed, role assignments break; migrate carefully
- Response schema: Adding fields is safe; removing/renaming breaks clients
- Endpoint paths: If routes change, update client/docs

---

13) Security Notes

Sensitive Data Handling

In Transit:

- All Admin APIs should use HTTPS (enforced by APP_URL config)
- Sanctum tokens should never be logged or exposed

At Rest:

- Passwords: Hashed with bcrypt (verified in Admin::verifyPassword())
- Tokens: Hashed before storage in personal_access_tokens table
- Logs: Ensure logs/storage directory not publicly accessible

In Code:

- Never commit .env with real credentials
- Use env() for sensitive values
- Do not output token in responses (already handled; Token sent once, hash stored)

Rate Limiting

Currently: Not implemented in Admin module. Recommend adding:

```
Route::middleware(['throttle:60,1'])->group(function () {
    Route::post('auth/login', ...);
});
```

Permission Pitfalls

- **Missing Permission Check**: Forgetting AdminPermission middleware allows unauthorized access
- **Hierarchical Permissions**: "reviews.manage" should imply "reviews.view" (check if implemented)
- **Role Assignment**: Ensure admin assigned to at least one role; no-role-admins can't access anything

Input Validation Hotspots

Endpoint	Validation
Login	Email format, password non-empty
Create Category	name_en required, max 255 chars; logo URL valid
Update Place	Coordinates valid lat/long; no injection in address
Hide Review	Reason optional; is_hidden boolean
Broadcast	Template ID exists; filters valid JSON

SQL Injection Prevention

- All queries use Eloquent ORM (parameterized)
- Raw queries rare; if used, verify via code review
- FormRequest validation provides some protection

CSRF / CORS

- **CSRF**: Not applicable (stateless API with token auth)
- **CORS**: Configure in `config/cors.php` if frontend on different domain

---

14) Known Issues / Tech Debt / TODO

Found in Code / Obvious Gaps

1. **Audit Logging**
   - Admin actions tracked by admin_id in some models (hidden_by, replied_by)
   - Missing: Comprehensive audit log table for all mutations
   - **Recommendation**: Add audit log table, middleware to auto-track all changes
2. **Soft Deletes**
   - Admins use soft delete; careful with role/permission cleanup
   - **Recommendation**: Test cascading soft deletes
3. **Pagination Defaults**
   - Hard-coded 15 items per page in some services
   - **Recommendation**: Make configurable per endpoint
4. **No Rate Limiting**

  - Login endpoints could be brute-forced
  - **Recommendation**: Add throttle middleware to auth endpoints

5. **Notification Queue**
   - Broadcast notifications may be synchronous (slow for large volume)
   - **Recommendation**: Queue all notification sends; implement retry logic

6. **Permission Seeding**
   - Permissions must be seeded; if missed, routes return 403
   - **Recommendation**: Document all required permissions in seeder comments

7. **Concurrency**
   - No optimistic locking on updates
   - **Recommendation**: Add version field to critical tables (catalog, settings)

8. **Testing**
   - Postman collection may be incomplete or outdated
   - **Recommendation**: Auto-generate from OpenAPI/Swagger spec

9. **Documentation**
   - Some services lack PHPDoc comments
   - **Recommendation**: Add detailed comments to service methods

10. **Error Messages**
    - Generic "Forbidden" error for missing permissions; no hint which permission needed
    - **Recommendation**: Return permission name in 403 response (with flag to enable)

Prioritized Recommendations

Priority	Item	Effort	Impact
P0	Rate limit login endpoint	Low	High (security)
P0	Audit logging middleware	Medium	High (compliance)
P1	Queue notification sends	Medium	Medium (performance)
P1	Optimize dashboard queries (add indexes)	Low	Medium (performance)
P2	Add OpenAPI spec generation	Medium	Medium (devex)
P2	Soft delete cascade tests	Low	Medium (reliability)

---

15) Change Log

Version History

Version	Date	Changes
1.0	2026-01-25	Initial handover document; complete Admin module coverage

Future Updates

- [ ] Post-launch: Add production metrics
- [ ] Post-launch: Document actual audit log implementation
- [ ] Q2 2026: API v2 breaking changes (if planned)
- [ ] Q2 2026: Admin mobile app (if planned)

Appendix: Quick Reference

Most Common Admin Operations

Login & Get Token:

```
curl -X POST http://localhost:8000/api/v1/admin/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"admin@rate-it.local","password":"password"}'
```

Create Category:

```
curl -X POST http://localhost:8000/api/v1/admin/categories \
  -H "Authorization: Bearer TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "name_en": "Restaurants",
    "name_ar": "المطاعم",
    "is_active": true
  }'
```

List Reviews:

```
curl http://localhost:8000/api/v1/admin/reviews?page=1&per_page=10&is_hidden=false
\
  -H "Authorization: Bearer TOKEN"
```

Get Dashboard Summary:

```
curl 'http://localhost:8000/api/v1/admin/dashboard/summary?from=2026-01-
01&to=2026-01-25' \
  -H "Authorization: Bearer TOKEN"
```

Document Signature

Prepared by: Backend Team
Date: January 25, 2026
Status: Ready for handover to Dev/QA/DevOps teams
Next Review: Post-deployment (30 days)

END OF HANDOVER DOCUMENT