**API Documentation for Recipe App Backend**

**Base URL**  
<http://localhost:5000/api>

**Authentication**

**1. Register a new user**  
POST /users/register

Body (application/json):  
{  
"username": "john\_doe",  
"email": "[john@example.com](mailto:john@example.com)",  
"password": "mypassword"  
}

Success response (201):  
{  
"message": "User registered",  
"user\_id": 123  
}

Errors:

* 400 Bad Request for validation errors.

**2. Login**  
POST /users/login

Body (application/json):  
{  
"username": "john\_doe",  
"password": "mypassword"  
}

Success response (200):  
{  
"token": "JWT\_TOKEN\_HERE"  
}

Note: Use this token in the **Authorization** header for protected routes:  
**Authorization: Bearer JWT\_TOKEN\_HERE**

Errors:

* 401 Unauthorized for invalid credentials.

**User Operations**

**3. Update user**  
PUT /users/:id

Path param:

* **id** (int): User ID to update.

Body (application/json):  
{  
"email": "[newemail@example.com](mailto:newemail@example.com)", // optional  
"password": "newpassword" // optional  
}

Success (200):  
{  
"message": "User updated"  
}

Errors:

* 400 Bad Request for invalid fields
* 403 Forbidden if user cannot update another user (enforce auth)
* 404 Not Found if user doesn't exist

**4. Delete user**  
DELETE /users/:id

Path param:

* **id** (int): User ID to delete.

Success (200):  
{  
"message": "User deleted"  
}

Errors:

* 403 Forbidden if not allowed
* 404 Not Found if user doesn't exist

**Recipe Management**

**5. Add a recipe**  
POST /recipes

**Protected**: Requires an Authorization header with Bearer token (**Authorization: Bearer <token>**)

**Behavior**:

* Creates a new recipe and associates ingredients and optionally categories.
* Ingredients can be referenced by existing **ingredient\_id** or by **name** (normalized and created if missing).
* All database steps are transactional; rollback on any failure.
* Bulk insertion used for efficiency.

**Request body (application/json)**:  
{  
"title": "Pasta with Tomato Sauce",  
"instructions": "Boil pasta. Cook sauce...",  
"image\_url": "<https://example.com/pics/pasta.jpg>", // optional  
"ingredients": [  
{ "ingredient\_id": 3, "quantity": "200g" }, // existing ingredient  
{ "name": "Tomato", "quantity": "3" }, // new or existing by name  
{ "name": "Olive Oil" } // optional quantity  
],  
"categories": [  
{ "name": "Main Course" }, // optional category  
{ "category\_id": 5 } // by ID ]  
}

**Validation**:

* **title**: required, non-empty string.
* **instructions**: required, non-empty string.
* **image\_url**: optional, must be valid URL if provided.
* **ingredients**: array with at least one item.
* Ingredient items: each must include either **ingredient\_id** or non-empty **name**, optional **quantity**.
* **categories**: optional; same validation rules as ingredients.

**Success (201)**:  
{  
"message": "Recipe created",  
"recipe\_id": 123  
}

**Errors**:

* 400 Bad Request: validation errors, missing fields, invalid URLs, invalid ingredient/category references.
* 404 Not Found: referenced ingredient or category does not exist.
* 500: server errors.

1. **6. Search recipes  
   GET /recipes/search  
   Query parameters:  
   • ingredient\_id (optional): filter for recipes that include this specific ingredient. Accepts one or multiple IDs via comma-separated string or repeated query params.  
   • category\_id (optional): filter for recipes that include this category. Accepts one or multiple IDs.  
   • q (optional): text search matching title or instructions (case-insensitive substring).  
   • page (optional): page number for pagination (default 1).  
   • limit (optional): items per page (default 20, max 100).**

**Example:  
GET /recipes/search?ingredient\_id=2,3&category\_id=5&q=soup&page=2&limit=10**

**Response (200):  
json  
{  
"data": [  
{  
"id": 10,  
"title": "Tomato Soup",  
"instructions": "...",  
"image\_url": null,  
"author\_id": 2,  
"ingredients": [  
{ "id": 1, "name": "tomato", "quantity": "3" },  
// ... other ingredients  
],  
"categories": [  
{ "id": 1, "name": "Soup" },  
// ... other categories  
],  
"author\_username": "chefjohn"  
},  
// more recipes  
],  
"meta": {  
"total": 123,  
"page": 1,  
"limit": 20  
}  
}  
Errors:  
• 400 Bad Request if any of the IDs (ingredient\_id, category\_id) provided are not valid integers.  
• 500 Internal Server Error for server issues.**

**7. Record a recipe click  
POST /recipes/:id/click  
Description:  
Increment the click counter for a recipe. Can be called anonymously or with authentication. If user is logged in, their click is also logged.**

**Response (200):**

**json**

**{**

**"message": "click recorded",**

**"clicks": 123**

**}**

**Errors:  
• 404 Not Found: if the recipe does not exist.  
• 500 Internal Server Error: for server issues.**

**8. Vote on a recipe  
POST /recipes/:id/vote  
Description:  
Allows a logged-in user to cast a vote on a recipe: upvote (vote: 1), downvote (vote: -1), or remove vote (vote: 0). Updates vote counts atomically.**

**Request body (application/json):**

**json**

**{**

**"vote": 1 // or -1, or 0 to remove vote**

**}**

**Response (200):**

**json**

**{**

**"message": "vote updated",**

**"recipe\_id": 123,**

**"my\_vote": 1,**

**"upvotes": 45,**

**"downvotes": 10,**

**"score": 35**

**}**

**Errors:  
• 404 Not Found: if the recipe does not exist.  
• 400 Bad Request: if vote is not 1, -1, or 0.  
• 401 Unauthorized: if user is not logged in.**

**9. Get vote info for a recipe  
GET /recipes/:id/vote  
Description:  
Returns the total number of upvotes, downvotes, the current score, and, if logged in, the current user's vote.**

**Response (200):**

**json**

**{**

**"recipe\_id": 123,**

**"upvotes": 45,**

**"downvotes": 10,**

**"score": 35,**

**"my\_vote": 1 // or -1 or 0 if not logged in**

**}**

**Errors:  
• 404 Not Found: if the recipe does not exist.**

**10. Remove user's vote  
DELETE /recipes/:id/vote  
Description:  
Removes the current logged-in user's vote on the specified recipe.**

**Response (200):**

**json**

**{**

**"message": "vote removed",**

**"recipe\_id": 123,**

**"my\_vote": 0,**

**"upvotes": 45,**

**"downvotes": 10,**

**"score": 35**

**}**

**Errors:  
• 404 Not Found: if the recipe does not exist.  
• 401 Unauthorized: if user is not logged in.**

**11. Get all categories for suggestions  
GET /categories/suggest**

**Description:  
Fetches and returns a list of all categories (id and name) for use in frontend autocomplete or suggestions.**

**Response (200):  
[  
 { "id": 1, "name": "Main Course" },  
 { "id": 2, "name": "Dessert" },  
 { "id": 3, "name": "Salad" },  
// ... other categories  
]**

**Errors:**

* **500 Internal Server Error: if there's a database or server issue.**

**12. Add favorite  
POST /favorites/:recipeId**

***Protected*: requires valid JWT in Authorization header**

**Path param:**

* **recipeId (int): ID of the recipe to favorite**

**Success (200):  
{ "message": "Recipe added to favorites" }**

**Errors:**

* **404 Not Found: if recipe doesn't exist**
* **409 Conflict: if already favorited (optional)**

**13. Remove favorite  
DELETE /favorites/:recipeId**

***Protected*: requires valid JWT**

**Success (200):  
{ "message": "Removed from favorites" }**

**Errors:**

* **404 Not Found: if recipe doesn't exist**

**14. List favorites  
GET /favorites**

***Protected*: requires valid JWT**

**Response (200):  
[  
// array of favorite recipe objects (full or minimal details)  
]**

**15. List users  
GET /admin/users**

**Supports query params: page, limit for pagination.**

**16. Delete user  
DELETE /admin/users/:id**

**17. Ban user  
POST /admin/users/:id/ban**

**18. Suspend user  
POST /admin/users/:id/suspend  
Body: { "duration": 7 } // days**

**19. Promote user to admin  
POST /admin/promote/:id**

**Response:  
{ "message": "User promoted to admin" }**

**20. Get logistics / dashboard stats  
GET /admin/logistics**

**Response (200):  
{  
"totalUsers": 100,  
"totalRecipes": 250,  
// other metrics like activeUsers, favoritesCount, pendingReports, etc.  
}**

**Common notes:**

* **All admin routes require Authorization: Bearer and enforce role checks.**
* **Validation errors return status 400.**
* **Authentication failures return 401.**
* **Forbidden access returns 403.**
* **Missing resources give 404.**
* **Successful creation typically returns 201.**