Of course. Here is the API documentation in Markdown format.

```
# API Documentation: Food Listing App (JWT)

### **Version 2.0**

This document provides documentation for the Food Listing App's backend API. The
API uses stateless **JSON Web Tokens (JWT)** for authenticating admin routes.

---

## **Base URL**

The base URL for all endpoints is:
`http://localhost:3000`

---

## **Authentication Flow (JWT)**

Authentication is handled via Bearer Tokens.

1. **Login**: Send admin credentials to the `POST /api/admin/login` endpoint.
2. **Receive Token**: On success, the API will return a JSON object containing a
JWT.
3. **Store Token**: The client application (e.g., your frontend) must securely
store this token.
4. **Authorize Requests**: For all protected admin routes, include the token in
the `Authorization` header with the `Bearer` scheme.

**Example Header:**
`Authorization: Bearer <your_jwt_here>`

---

## **Admin Endpoints**

### **Admin Login**

`POST /api/admin/login`

Authenticates an administrator and returns a JWT.

**Request Body:** (`application/json`)
| Field      | Type   | Description             |
| :--------- | :----- | :---------------------- |
| `username` | String | The admin's username.   |
| `password` | String | The admin's password.   |

**Example Request:**
```json
{
```
```

```
    "username": "admin",
    "password": "password"
}
```

Success Response (200 OK): Returns the JWT.

```
{
    "message": "Admin login successful",
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwicm9sZSI6ImFkbWlu
IiwiaWF0IjoxNzUyNTMxMjAwLCJleHAiOjE3NTI2MTc2MDB9.some_signature_here"
}
```

Error Response (401 Unauthorized): `Invalid admin credentials`

Add Food Item

`POST /api/food`

Creates a new food item. Requires JWT authentication.

Headers:

Key	Value
`Authorization`	`Bearer <token>`

Request Body: (`multipart/form-data`)

Field	Type	Description
`name`	String	The name of the food item.
`recipe`	String	The recipe or description.
`image`	File	The image file for the item.

Responses:

- **201 Created:** Returns the newly created food item object.
- **400 Bad Request:** `Name, recipe, and image are required.`
- **401 Unauthorized:** `Access denied. No token provided.`
- **403 Forbidden:** `Invalid token.`

Update Food Item

`PUT /api/food/:id`

Updates an existing food item. Requires JWT authentication.

Headers:

Key	Value
`Authorization`	`Bearer <token>`

URL Parameters:

Parameter	Type	Description
`id`	Number	The unique ID of the food item.

Request Body: (`multipart/form-data`) - All fields are optional.

Field	Type	Description
`name`	String	The new name for the item.
`recipe`	String	The new recipe for the item.
`image`	File	A new image to replace the existing one.

Responses:

- **200 OK:** Returns the updated food item object.
- **401 Unauthorized / 403 Forbidden:** For token errors.
- **404 Not Found:** `Food item not found.`

Delete Food Item

`DELETE /api/food/:id`

Deletes a food item. Requires JWT authentication.

Headers:

Key	Value
`Authorization`	`Bearer <token>`

URL Parameters:

Parameter	Type	Description
`id`	Number	The unique ID of the food item.

Responses:

- **200 OK:** `{"message": "Food item deleted successfully."}`
- **401 Unauthorized / 403 Forbidden:** For token errors.
- **404 Not Found:** `Food item not found.`

Public User Endpoints

These endpoints are public and do not require authentication.

Get All Food Items

`GET /api/food`

Retrieves a list of all available food items.

Success Response (200 OK): Returns an array of food item objects.

```
[
    {
        "id": 1672531200000,
        "name": "Spaghetti Bolognese",
        "recipe": "A classic Italian pasta dish...",
        "image": "/public/uploads/1672531200000.jpg"
    }
]
```

Get Single Food Item

`GET /api/food/:id`

Retrieves a specific food item by its ID.

URL Parameters:

Parameter	Type	Description
`id`	Number	The unique ID of the food item.

Responses:

- **200 OK:** Returns the requested food item object.
- **404 Not Found:** `Food item not found.`

```

```