

Module User

SignUp:

The screenshot shows a REST client interface with a POST request to `http://localhost:3001/api/v1/users/signup`. The request body is a JSON object with the following fields: `name` (José), `email` (jose@gmail.com), `password` (12345abc), and `role` (admin). The response is a 201 Created status with a response time of 733 ms and a body size of 463 B. The response body is a JSON object containing a `token` and a `user` object with `id` (57), `name` (José), and `email` (jose@gmail.com).

```
POST http://localhost:3001/api/v1/users/signup

{
  "name": "José",
  "email": "jose@gmail.com",
  "password": "12345abc",
  "role": "admin"
}
```

201 Created 733 ms 463 B Save as example

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NTcsImhhdCI6MTcwNDMyNzQwNCwiZXhwIjoxNzA0MzMxMDA0fQ.GUHf0qsgJOPzK3hx2rnoVCOhfYIQAru0ibuzzyE0z6GY",
  "user": {
    "id": 57,
    "name": "José",
    "email": "jose@gmail.com"
  }
}
```

Login

The screenshot shows a REST client interface with a POST request to `http://localhost:3001/api/v1/users/login`. The request body is a JSON object with the following fields: `email` (jose@gmail.com) and `password` (12345abc). The response is a 200 OK status with a response time of 671 ms and a body size of 458 B. The response body is a JSON object containing a `token` and a `user` object with `id` (57), `name` (José), and `email` (jose@gmail.com).

```
POST http://localhost:3001/api/v1/users/login

{
  "email": "jose@gmail.com",
  "password": "12345abc"
}
```

200 OK 671 ms 458 B Save as example

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NTcsImhhdCI6MTcwNDMyNzQwMSwiZXhwIjoxNzA0MzMxMTE0fQ.Kjdb42D3WzDZ8ALzjcp4zfkQgSQoL1wV9zArsDdq794",
  "user": {
    "id": 57,
    "name": "José",
    "email": "jose@gmail.com"
  }
}
```

Método Patch Sin token:

The screenshot shows a Postman interface for a PATCH request to `http://localhost:3001/api/v1/users/4`. The request body is in JSON format, containing an `email` field with the value `"Giovanni@gmail.com"`. The response status is `500 Internal Server Error` with a message `"jwt expired"` and a stack trace indicating the error occurred in the `TokenExpiredError` module.

```
1 {
2   "email": "Giovanni@gmail.com"
3 }
```

Body: `500 Internal Server Error` 17 ms 1.61 KB

```
1 {
2   "status": "fail",
3   "message": "jwt expired",
4   "stack": "TokenExpiredError: jwt expired\n    at D:\\Desarrollo\nWeb\\Academlo\\Node\\Entregables-de-Node\\ProyectoFinal\\AcademloMeals\\node_mod"
```

Método Patch Con Token Equivocado

The screenshot shows a Postman interface for a PATCH request to `http://localhost:3001/api/v1/users/4`. The request is configured with a `Bearer Token` authorization header. The response status is `401 Unauthorized` with a message `"You do not own this account"`.

Type: `Bearer Token`

Token: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV...`

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization

Body: `401 Unauthorized` 504 ms 1.12 KB

```
1 {
2   "status": "error",
3   "message": "You do not own this account",
```

Método Patch Satisfactoriamente

PATCH ▼ http://localhost:3001/api/v1/users/4 Send ▼

Params **Auth** Headers (9) Body Pre-req. Tests Settings Cookies

Type: Bearer Token ▼

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV...

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization

Body ▼ 200 OK 652 ms 527 B Save as example

Pretty Raw Preview Visualize JSON ▼

```
1 {
2   "message": "Info Changed 😊",
3   "user": {
4     "id": 4,
5     "name": "Giovanni",
6     "email": "Giovanni@gmail.com",
7     "password": "$2b$12$LYBnx5qcvAUGU8h5yqA0lM3TqMXDiTBrcU2Lwfq7ibmaD8uqOfG",
8     "role": "normal",
9     "status": true,
10    "createdAt": "2023-12-27T19:52:42.787Z",
11    "updatedAt": "2024-01-04T00:30:52.721Z"
12  }
13 }
```

Método Delete Sin Token

DELETE ▼ http://localhost:3001/api/v1/users/58 Send ▼

Params **Auth** Headers (7) Body Pre-req. Tests Settings Cookies

Type: Bearer Token ▼

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV...

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization

Body ▼ 500 Internal Server Error 9 ms 1.61 KB Save as example

Pretty Raw Preview Visualize JSON ▼

```
1 {
2   "status": "fail",
3   "message": "jwt expired",
4   "stack": "TokenExpiredError: jwt expired\n    at D:\\\\Desarrollo"
```

Método Delete Token Invalido

The screenshot shows a Postman interface for a DELETE request to `http://localhost:3001/api/v1/users/58`. The request is configured with a Bearer Token: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV...`. The response status is **401 Unauthorized** with a response time of 510 ms and a body size of 1.12 KB. The response body, shown in JSON format, contains an error message: `{ "status": "error", "message": "You do not own this account", "stack": "Error: You do not own this account\n at protectAccountOwner (file:///...)" }`.

Método Delete Satisfactoriamente

The screenshot shows a Postman interface for a DELETE request to `http://localhost:3001/api/v1/users/58`. The request is configured with the same Bearer Token as the previous screenshot. The response status is **200 OK** with a response time of 618 ms and a body size of 295 B. The response body, shown in JSON format, contains a success message: `{ "message": "User deleted successfully.! 😊" }`.

Método Get Order Sin Token

POST SignUp POST Login GET Get All Or GET http://loca + No Environment

Academlo-Meals / Users / Get All Orders Save

GET http://localhost:3001/api/v1/users/orders Send

Params Auth Headers (7) Body Pre-req. Tests Settings Cookies

Type Bearer Token Token eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV...

The authorization header will be automatically generated when you send the request. Learn more about

Body 401 Unauthorized 7 ms 2.01 KB Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "status": "error",
3   "message": "You are not logged in!. Please login to get access",
4   "stack": "Error: You are not logged in!. Please login to get access\n    at file:///.../Academlo-Meals/Node/Entregables de Node/ProyectoFinal/..."
```

Método Get Order

GET http://localhost:3001/api/v1/users/orders Send

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Body 200 OK 765 ms 1.36 KB Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Orders",
3   "orders": [
4     {
5       "id": 1,
6       "nameUser": "Artemio",
7       "UserNumber": 53,
8       "foodName": "Pizza de Champiñones rellena de queso",
9       "quantity": 2,
10      "pricePiece": 65,
11      "totalPrice": 130,
12      "restaurantId": 4,
13      "nameRestaurant": "Casa de las Pizzas"
14    },
15    {
16      "id": 2,
17      "nameUser": "Artemio",
18      "UserNumber": 53,
19      "foodName": "Tabla de quesos",
20      "quantity": 4,
21      "pricePiece": 60,
22      "totalPrice": 240,
23      "restaurantId": 3,
24      "nameRestaurant": "Los Portales"
25    }
26  ]
27 }
```

Método Get Con Id Sin Token

Postman interface showing a GET request to `http://localhost:3001/api/v1/users/orders/1`. The request is unauthorized, returning a 401 status and an error message.

Request:

- Method: GET
- URL: `http://localhost:3001/api/v1/users/orders/1`
- Auth: Bearer Token (Token: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV...`)

Response:

```
1 {
2   "status": "error",
3   "message": "You are not logged in!. Please login to get access",
4   "stack": "Error: You are not logged in!. Please login to get access\n    at file:///D:/Desarrollo%20Web/Academlo/Node/Entregables-de-Node/ProyectoFinal/
```

Método Get Con Id

Postman interface showing a GET request to `http://localhost:3001/api/v1/users/orders/6`. The request is successful, returning a 200 status and a JSON response.

Request:

- Method: GET
- URL: `http://localhost:3001/api/v1/users/orders/6`
- Auth: Bearer Token (Token: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV...`)

Response:

```
1 {
2   "message": "Order By Id: ",
3   "order": {
4     "id": 6,
5     "nameUser": "Giovanni",
6     "UserNumber": 4,
7     "foodName": "Tabla de quesos",
8     "quantity": 12,
9     "pricePiece": 60,
10    "totalPrice": 720,
11    "restaurantId": 3,
12    "nameRestaurant": "Los Portales"
13  }
14 }
```

Se debe encriptar la contraseña usando bcryptjs

id	name	email	password	role	status	createdAt	updatedAt
1	CésarínVe	velazquezcesarinve@...	\$2b\$12\$7bL/RKMrw918t6...	norm...	FALSE	2023-12-26 21:12:48.942+00	2023-12-28 04:54:46.827+00
2	Fernandita	fernandita@gmail.com	\$2b\$12\$7cPvpDCg.XiKtc...	norm...	FALSE	2023-12-27 19:13:43.805+00	2023-12-28 04:56:31.299+00
3	Genesis	gen@gmail.com	\$2b\$12\$twivYTTXOJAGPw...	norm...	TRUE	2023-12-27 19:14:03.112+00	2023-12-27 19:14:03.112+00
4	Giovanni	Giovanni@gmail.com	\$2b\$12\$LTy8nx5qcvAUgU...	norm...	TRUE	2023-12-27 19:52:42.787+00	2024-01-04 00:30:52.721+00
36	Janet	janet@gmail.com	\$2b\$12\$qba1hoikYrz3gt...	norm...	FALSE	2023-12-27 21:14:05.972+00	2023-12-28 04:53:36.6+00
53	Artemio	artemio@gmail.com	\$2b\$12\$JnBwcxbb1cOTUZ...	admin	TRUE	2023-12-28 06:01:34.653+00	2023-12-28 06:01:34.653+00
54	Jaimeee	jai@gmail.com	\$2b\$12\$ivQtY9tVBIIdusC...	norm...	FALSE	2024-01-02 20:00:02.126+00	2024-01-02 20:00:38.152+00
55	Josue	josue@gmail.com	\$2b\$12\$NVx1rCwalm1.8...	norm...	FALSE	2024-01-02 20:01:35.876+00	2024-01-02 20:02:01.506+00
56	Jeison	jey@gmail.com	\$2b\$12\$kNS/aaPu43wGvx...	norm...	FALSE	2024-01-02 20:02:45.256+00	2024-01-02 20:03:04.738+00
57	José	jose@gmail.com	\$2b\$12\$x4QnnqsgmGCW//...	admin	TRUE	2024-01-04 00:16:43.792+00	2024-01-04 00:16:43.792+00
58	Martin	martin@gmail.com	\$2b\$12\$Nth9AK7od83UCV...	norm...	FALSE	2024-01-04 00:21:52.282+00	2024-01-04 01:00:51.667+00

MODULE RESTAURANT

Método para crear un restaurant sin token

POSThttp://localhost:3001/api/v1/restaurants/Send

ParamsAuth●Headers (8)Body●Pre-req. Tests SettingsCookies

rawJSONBeautify

```
1 {
2   ... "name": "La Chipola",
3   ... "address": "Calle Venustiano",
4   "rating": 7
5 }
```

Body401 Unauthorized20 ms2.01 KBSave as example

PrettyRawPreviewVisualizeJSON

```
1 {
2   "status": "error",
3   "message": "You are not logged in!. Please login to get access",
4   "stack": "Error: You are not logged in!. Please login to get access\n  at file:///...
```

Método para crear un restaurant con una cuenta que no es de admin:

The screenshot shows a REST client interface with a POST request to `http://localhost:3001/api/v1/restaurants/`. The request body is a JSON object: `{ "name": "La Chipola", "address": "Calle Venustiano" }`. The response is a 403 Forbidden status with a message: `{ "status": "error", "message": "You do not have permission to perform this action" }`.

```
POST http://localhost:3001/api/v1/restaurants/

{
  "name": "La Chipola",
  "address": "Calle Venustiano",
}
```

Body 403 Forbidden 297 ms 336 B Save as example

```
{
  "status": "error",
  "message": "You do not have permission to perform this action"
}
```

Método Create con cuenta de admin pero con raiting fuera de los limites:

The screenshot shows a REST client interface with a POST request to `http://localhost:3001/api/v1/restaurants/`. The request body is a JSON object: `{ "name": "Barezzito", "address": "Calle Casones", "rating": 6 }`. The response is a 500 Internal Server Error status with a message: `{ "status": "fail", "message": "Raiting must be max 5 as a \n good place and min 1 as a place not a goodThe rating should be a maximum of 5 \n as an excellent place and a minimum of 1 as a not so good place." }`.

```
POST http://localhost:3001/api/v1/restaurants/

{
  "name": "Barezzito",
  "address": "Calle Casones",
  "rating": 6
}
```

Body 500 Internal Server Error 539 ms 488 B Save as example

```
{
  "status": "fail",
  "message": "Raiting must be max 5 as a \n good place and min 1 as a place not a goodThe rating should be a maximum of 5 \n as an excellent place and a minimum of 1 as a not so good place."
}
```


Método Create con cuenta de admin

The screenshot shows a REST client interface with a POST request to `http://localhost:3001/api/v1/restaurants/`. The request body is a JSON object with the following fields: `name` (Santos), `address` (Calle Amapolas), and `rating` (4). The response is a 201 Created status with a response time of 529 ms and a body size of 409 B. The response body is a JSON object with the following fields: `status` (true), `id` (8), `name` (Santos), `address` (Calle Amapolas), `rating` (4), `updatedAt` (2024-01-04T06:09:52.813Z), and `createdAt` (2024-01-04T06:09:52.813Z).

```
POST http://localhost:3001/api/v1/restaurants/

{
  "name": "Santos",
  "address": "Calle Amapolas",
  "rating": 4
}
```

Body

```
{
  "status": true,
  "id": 8,
  "name": "Santos",
  "address": "Calle Amapolas",
  "rating": 4,
  "updatedAt": "2024-01-04T06:09:52.813Z",
  "createdAt": "2024-01-04T06:09:52.813Z"
}
```

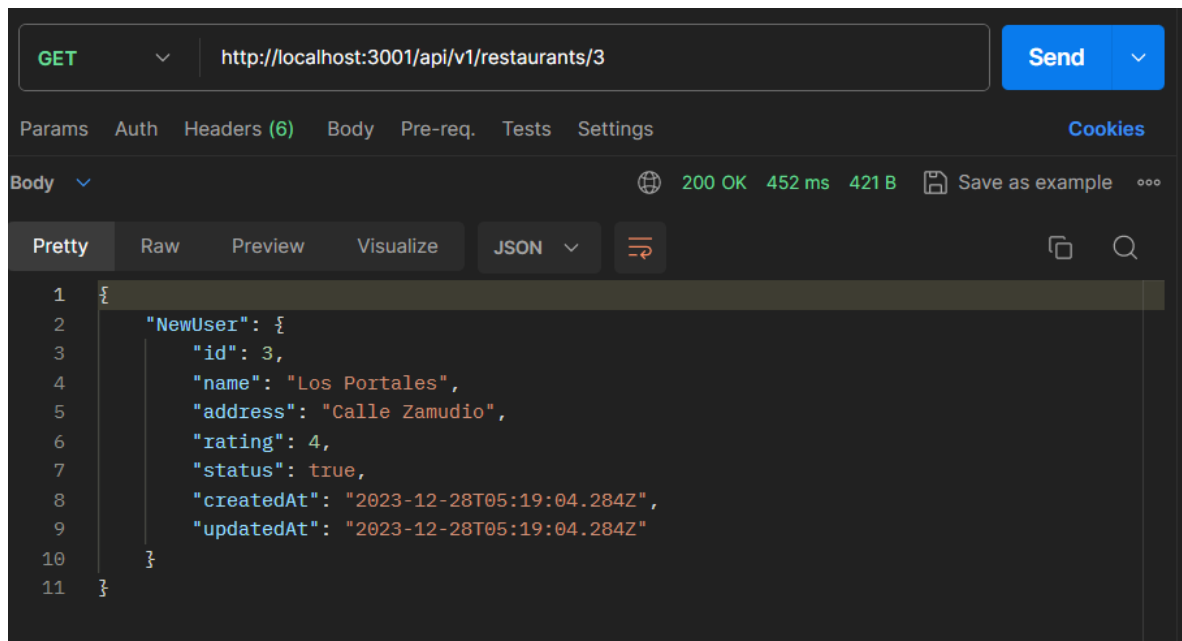
Método Get Para obtener todos los Restaurantes:

The screenshot shows a REST client interface with a GET request to `http://localhost:3001/api/v1/restaurants`. The response is a 200 OK status with a response time of 436 ms and a body size of 1.56 KB. The response body is a JSON object with the following fields: `message` (Users:), `NewUser` (array of restaurant objects), and `updatedAt` (2023-12-28T05:19:04.284Z).

```
GET http://localhost:3001/api/v1/restaurants

{
  "message": "Users: ",
  "NewUser": [
    {
      "id": 3,
      "name": "Los Portales",
      "address": "Calle Zamudio",
      "rating": 4,
      "status": true,
      "createdAt": "2023-12-28T05:19:04.284Z",
      "updatedAt": "2023-12-28T05:19:04.284Z"
    },
    {
      "id": 4,
      "name": "Casa de las Pizzas",
      "address": "Calle Zambrano",
      "rating": 5,
      "status": true,
      "createdAt": "2023-12-28T05:44:33.635Z",
      "updatedAt": "2023-12-28T05:44:33.635Z"
    }
  ],
  "updatedAt": "2023-12-28T05:19:04.284Z"
}
```

Obtener restaurant por id



GET <http://localhost:3001/api/v1/restaurants/3> Send

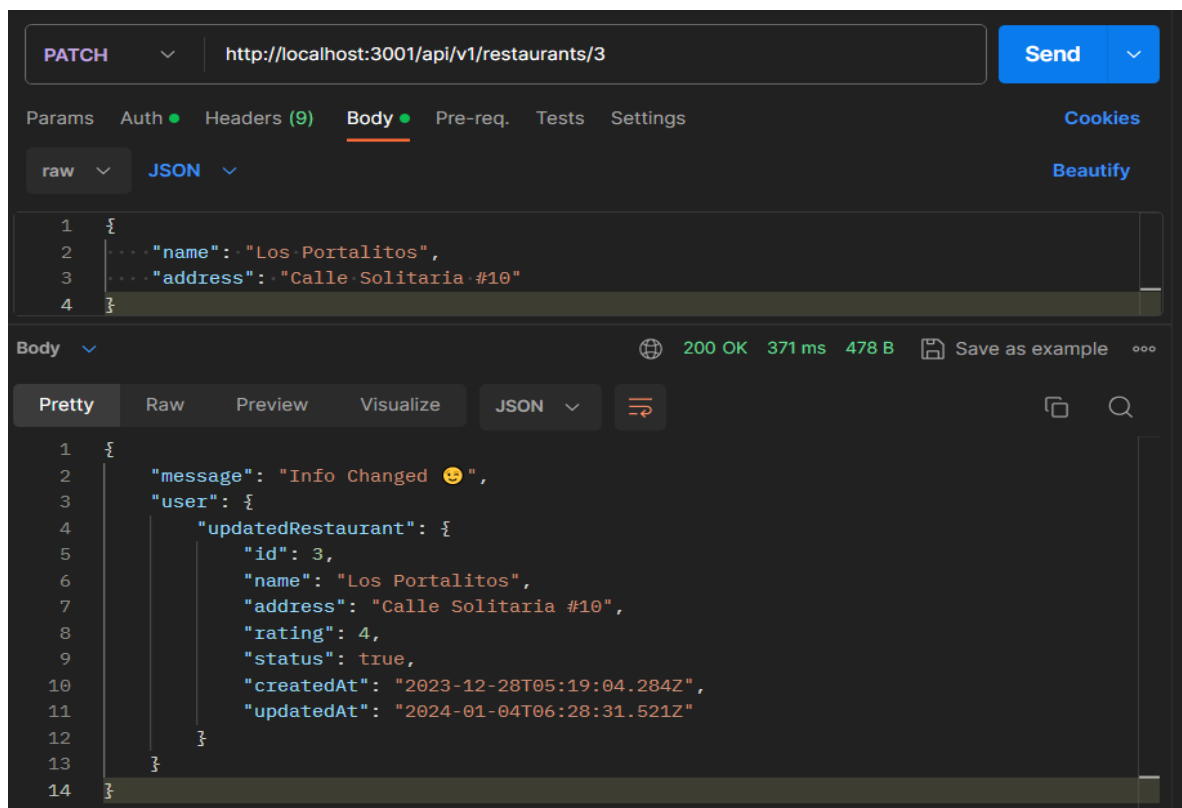
Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Body 200 OK 452 ms 421 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "NewUser": {
3     "id": 3,
4     "name": "Los Portales",
5     "address": "Calle Zamudio",
6     "rating": 4,
7     "status": true,
8     "createdAt": "2023-12-28T05:19:04.284Z",
9     "updatedAt": "2023-12-28T05:19:04.284Z"
10  }
11 }
```

Método Patch Actualizar restaurant (name, address):



PATCH <http://localhost:3001/api/v1/restaurants/3> Send

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

raw JSON Beautify

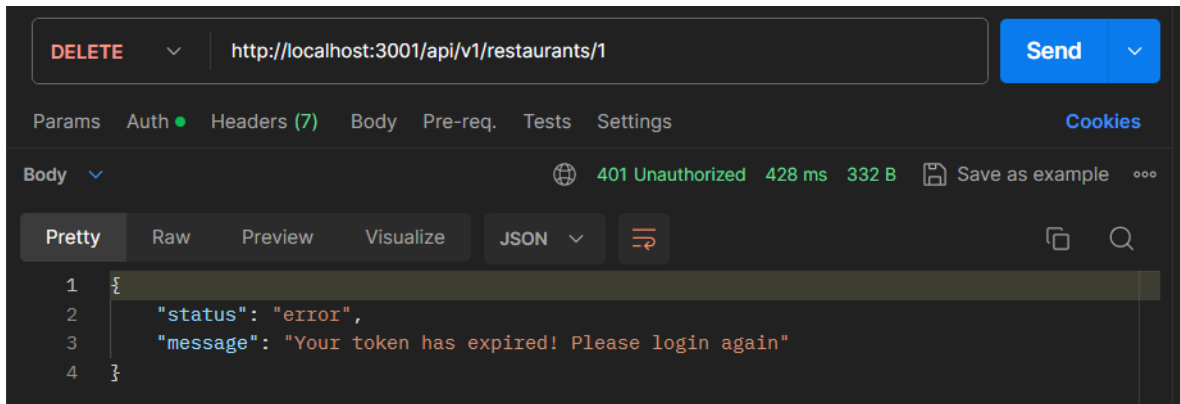
```
1 {
2   "name": "Los Portalitos",
3   "address": "Calle Solitaria #10"
4 }
```

Body 200 OK 371 ms 478 B Save as example

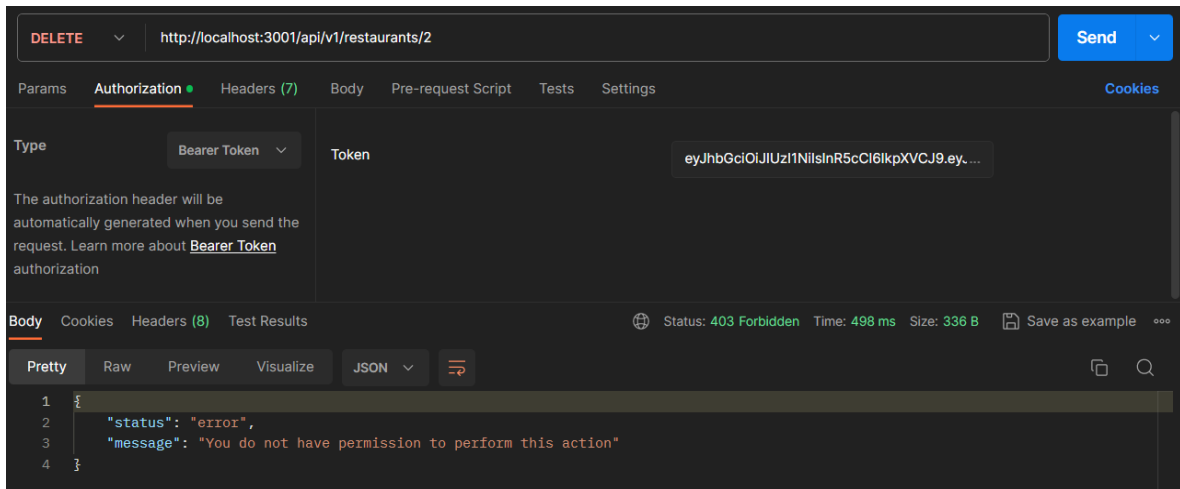
Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Info Changed 😊",
3   "user": {
4     "updatedRestaurant": {
5       "id": 3,
6       "name": "Los Portalitos",
7       "address": "Calle Solitaria #10",
8       "rating": 4,
9       "status": true,
10      "createdAt": "2023-12-28T05:19:04.284Z",
11      "updatedAt": "2024-01-04T06:28:31.521Z"
12    }
13  }
14 }
```

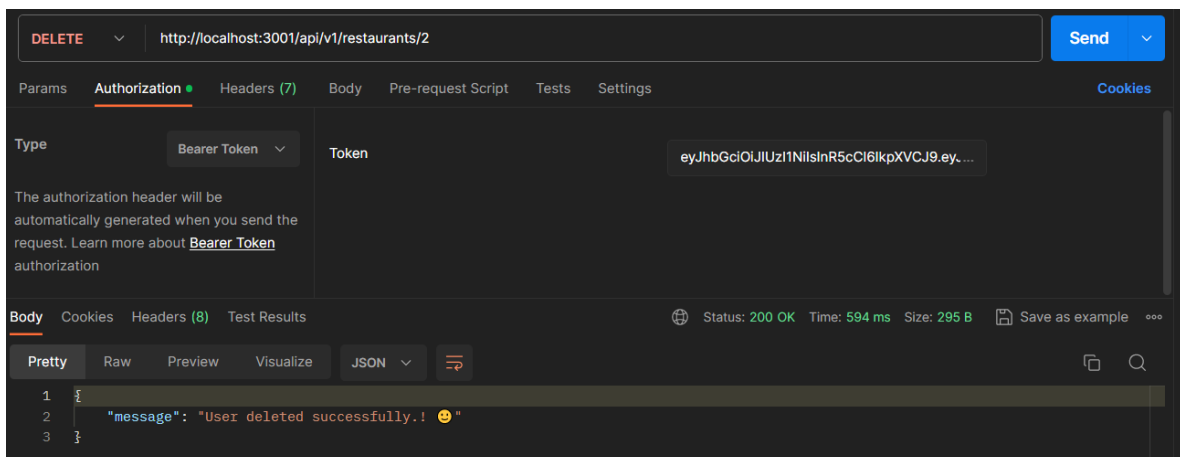
Método Delete Deshabilitar restaurant. Validación 1



Método Delete Deshabilitar restaurant. Validación 2



Método Delete Satisfactoriamente:



Crear una nueva reseña en el restaurant. Validación 1

The screenshot shows a REST client interface with a POST request to `http://localhost:3001/api/v1/restaurants/reviews/3`. The request body is a JSON object: `{ "comment": "Excelente11111", "raiting": 50 }`. The response is a 500 Internal Server Error with a message: `"message": "Raiting must be max 5 as a \n good place and min 1 as a place not a goodThe rating should be a maximum of 5 \n as an excellent place and a minimum of 1 as a not so good place."`

```
POST http://localhost:3001/api/v1/restaurants/reviews/3
{
  "comment": "Excelente11111",
  "raiting": 50
}
```

```
{
  "status": "fail",
  "message": "Raiting must be max 5 as a \n good place and min 1 as a place not a goodThe rating should be a maximum of 5 \n as an excellent place and a minimum of 1 as a not so good place."
}
```

Crear una nueva reseña en el restaurant.

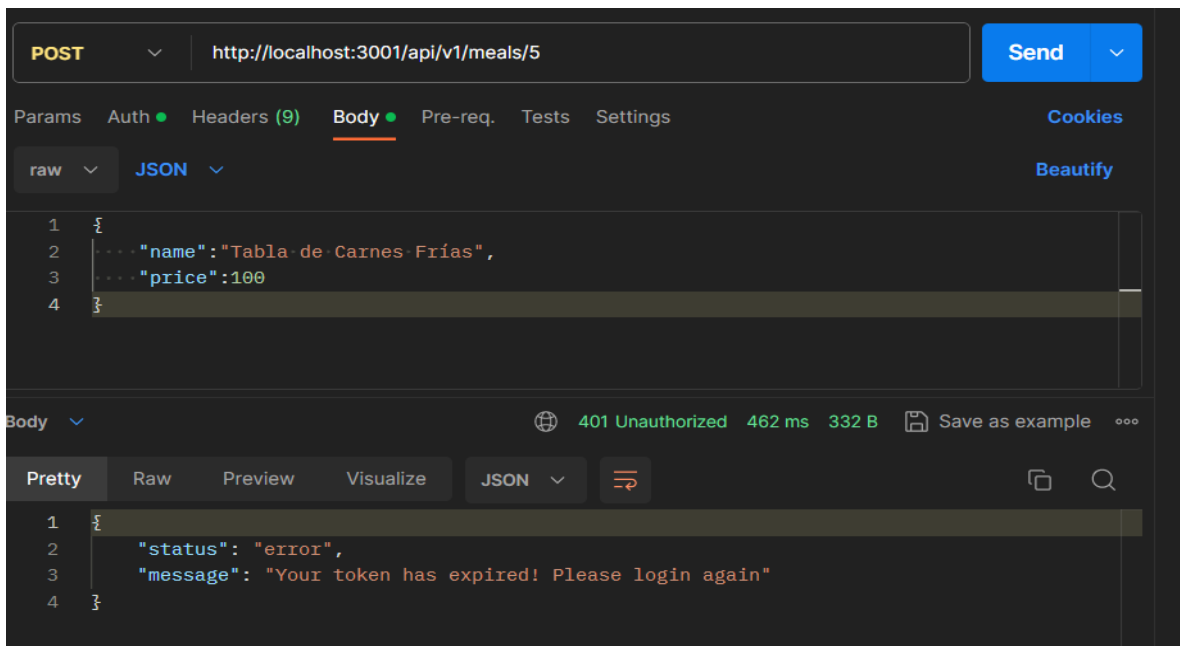
The screenshot shows the same REST client interface with a POST request to `http://localhost:3001/api/v1/restaurants/reviews/3`. The request body is a JSON object: `{ "comment": "Excelente", "raiting": 5 }`. The response is a 201 Created status with a body: `{ "RestaurantId": "3", "userId": 53, "comment": "Excelente", "raiting": 5 }`

```
POST http://localhost:3001/api/v1/restaurants/reviews/3
{
  "comment": "Excelente",
  "raiting": 5
}
```

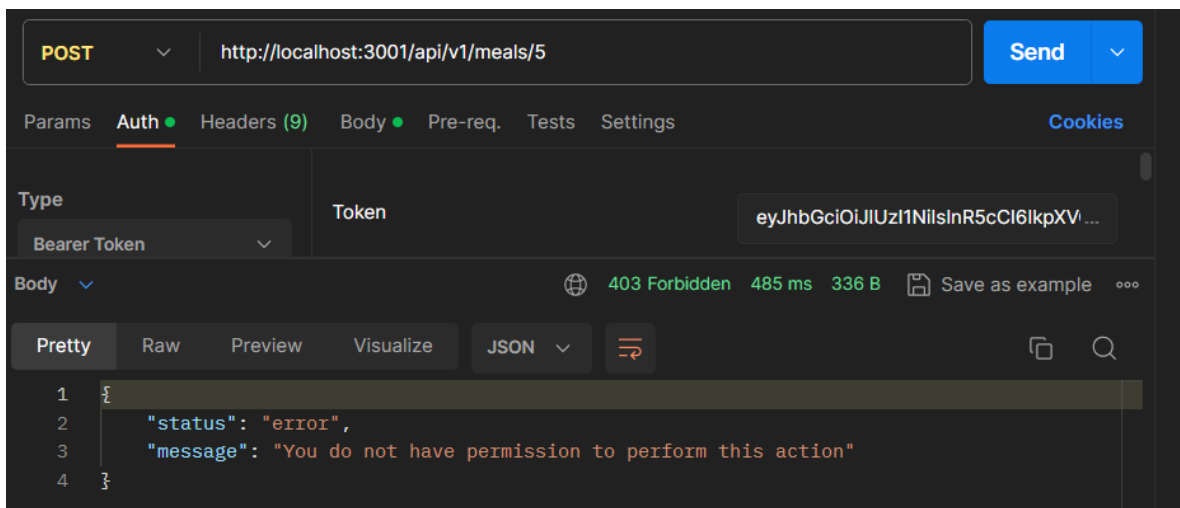
```
{
  "RestaurantId": "3",
  "userId": 53,
  "comment": "Excelente",
  "raiting": 5
}
```

MODULE MEALS

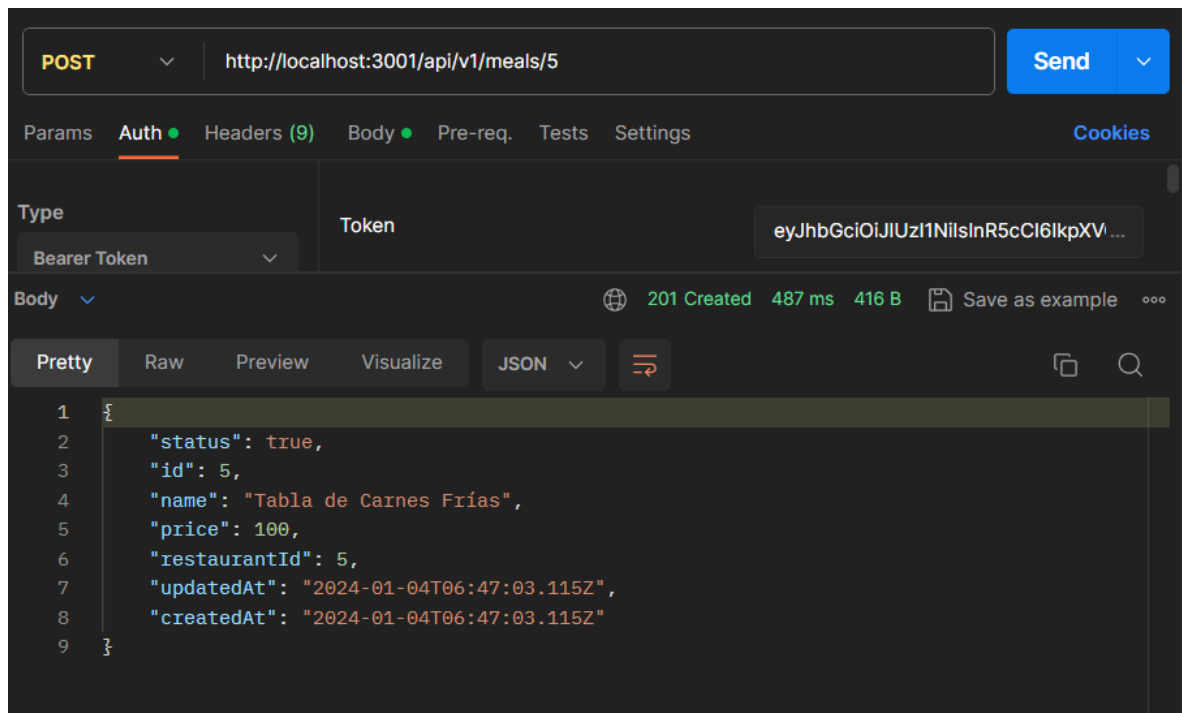
Método Post: Crear una nueva comida en el restaurant. Validación 1



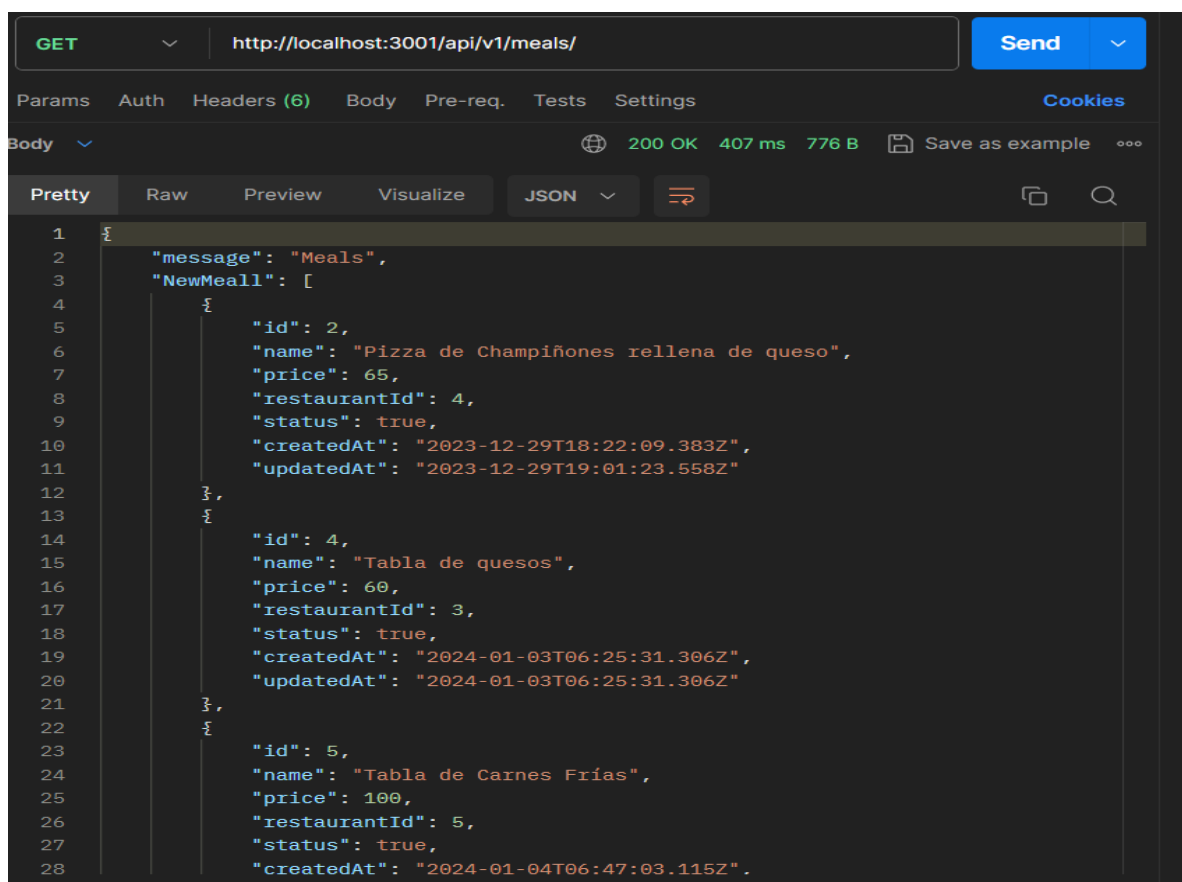
Método Post: Crear una nueva comida en el restaurant. Validación 2



Método Post: Crear una nueva comida en el restaurant Satisfactoriamente.



Método Get: Obtener todas las comidas con status Active.



Método Get One: Obtener por id una comida con status Active Validación 1

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:3001/api/v1/meals/1
- Status:** 500 Internal Server Error
- Response Time:** 166 ms
- Response Size:** 317 B
- Response Body (JSON):**

```
{
  "status": "fail",
  "message": "The id: 1 not found"
}
```

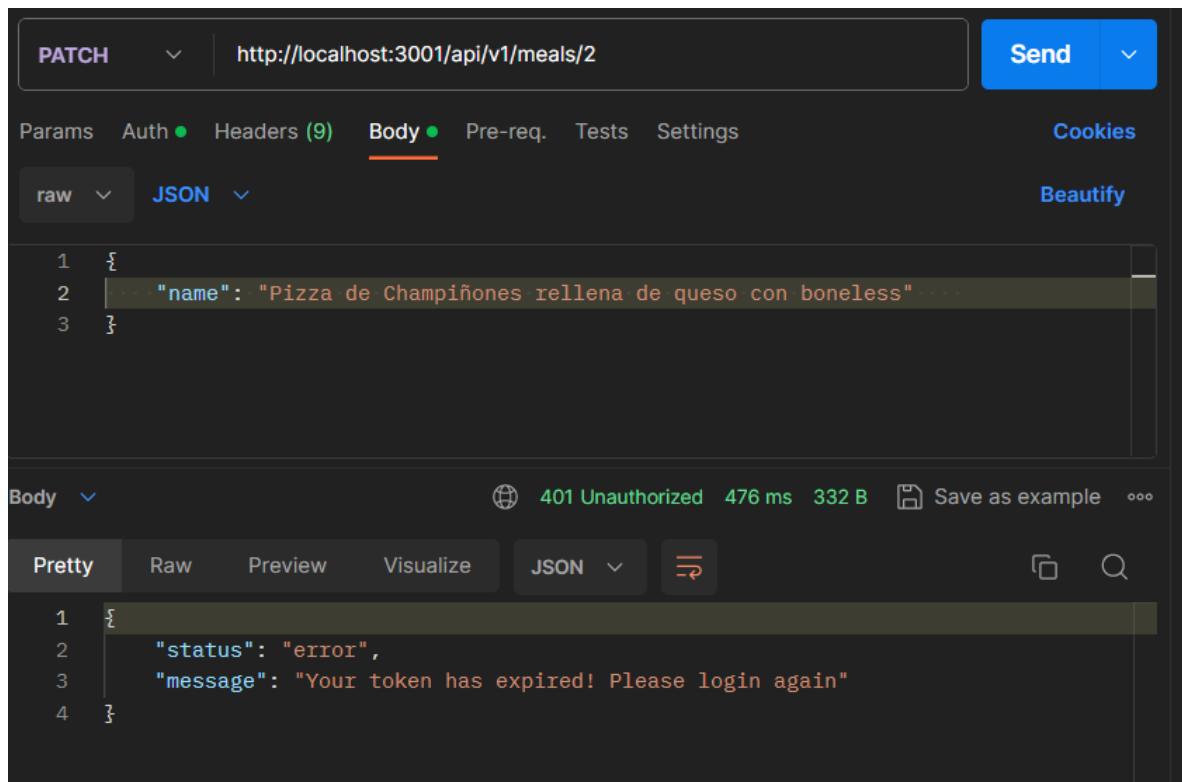
Método Get One: Obtener por id una comida con status Active

The screenshot shows a REST client interface with the following details:

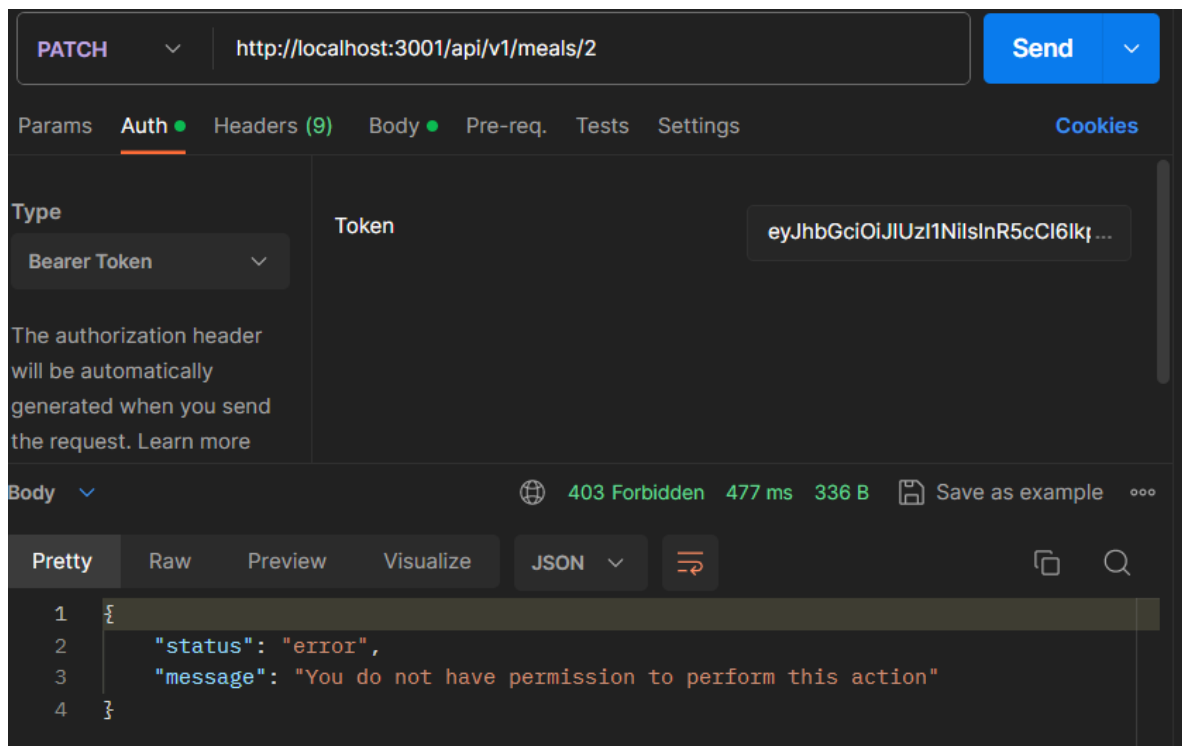
- Method:** GET
- URL:** http://localhost:3001/api/v1/meals/2
- Status:** 200 OK
- Response Time:** 418 ms
- Response Size:** 436 B
- Response Body (JSON):**

```
{
  "Meal": {
    "id": 2,
    "name": "Pizza de Champiñones rellena de queso",
    "price": 65,
    "restaurantId": 4,
    "status": true,
    "createdAt": "2023-12-29T18:22:09.383Z",
    "updatedAt": "2023-12-29T19:01:23.558Z"
  }
}
```

Método Patch: Actualizar comida (name, price).Validación 1



Método Patch: Actualizar comida (name, price).Validación 2



Método Patch: Actualizar comida (name, price).

The screenshot shows a REST client interface with the following details:

- Method:** PATCH
- URL:** http://localhost:3001/api/v1/meals/2
- Body (JSON):**

```
{  "name": "Pizza de Champiñones con boneless"}
```
- Response (JSON):**

```
{  "message": "Info changed 😊",  "meal": {    "id": 2,    "name": "Pizza de Champiñones con boneless",    "price": 65,    "restaurantId": 4,    "status": true,    "createdAt": "2023-12-29T18:22:09.383Z",    "updatedAt": "2024-01-04T07:07:54.474Z"  }}
```
- Status:** 200 OK, 583 ms, 461 B

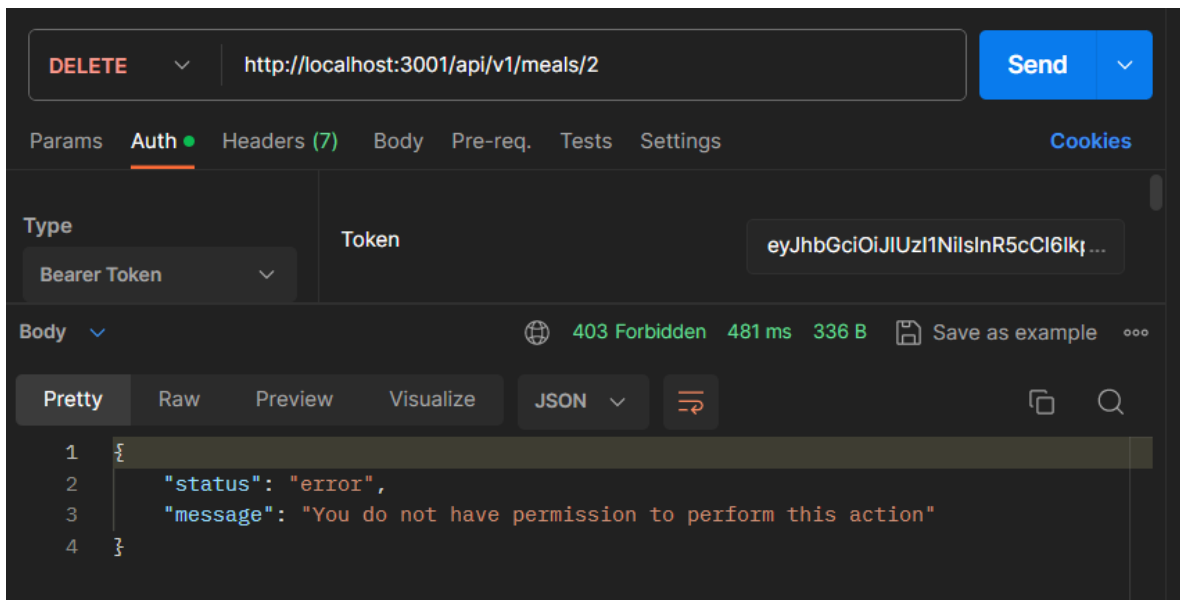
Método Delete: Deshabilitar comida Validación 1

The screenshot shows a REST client interface with the following details:

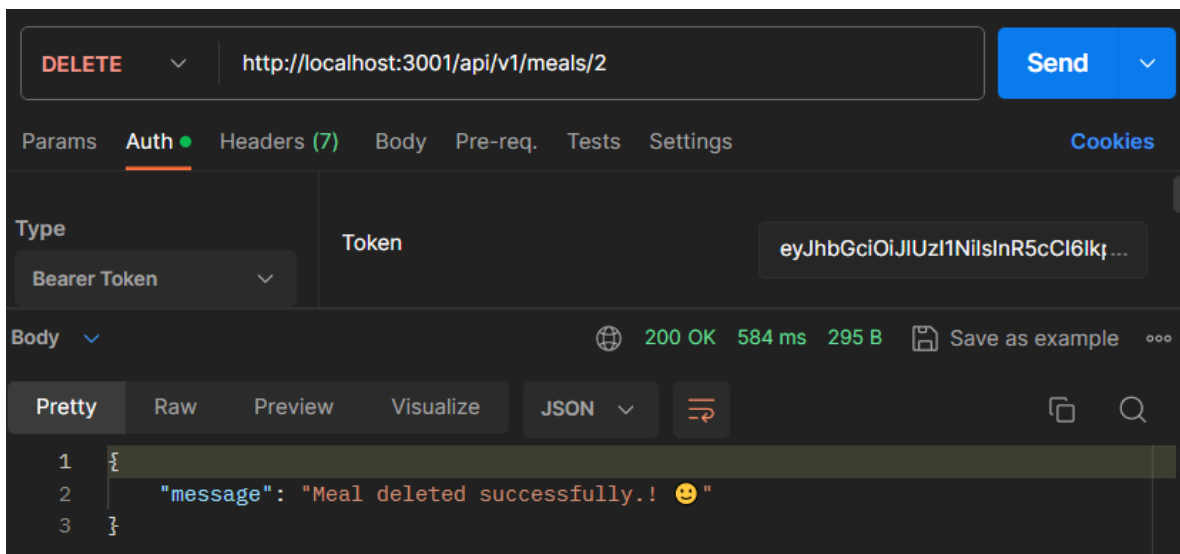
- Method:** DELETE
- URL:** http://localhost:3001/api/v1/meals/2
- Response (JSON):**

```
{  "status": "error",  "message": "Your token has expired! Please login again"}
```
- Status:** 401 Unauthorized, 480 ms, 332 B

Método Delete: Deshabilitar comida Validación 2

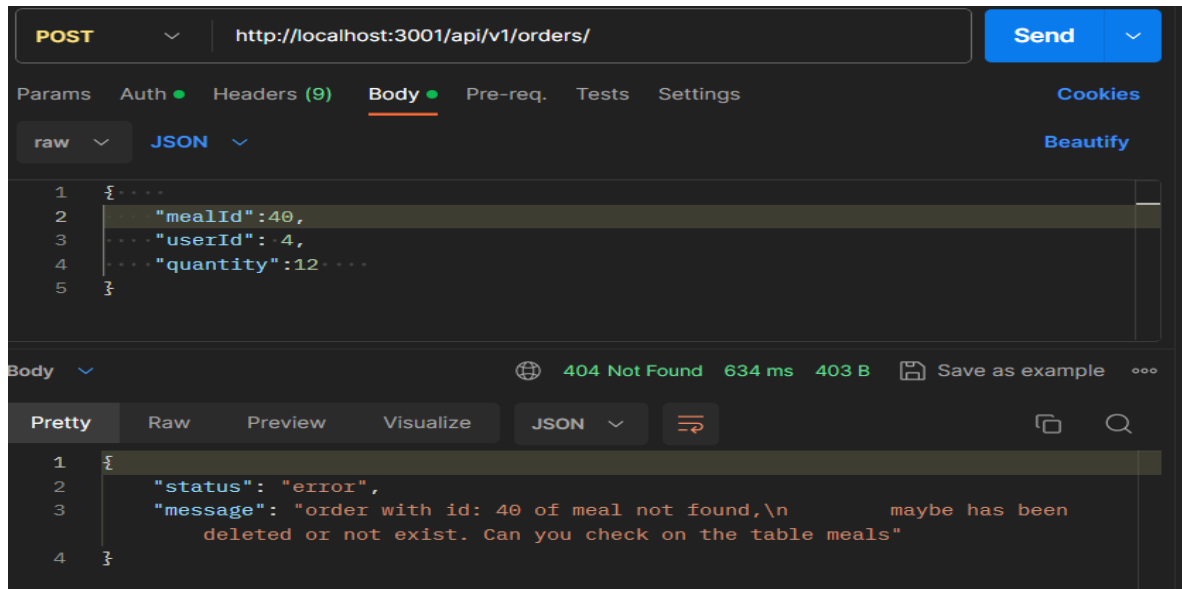


Método Delete: Deshabilitar comida.

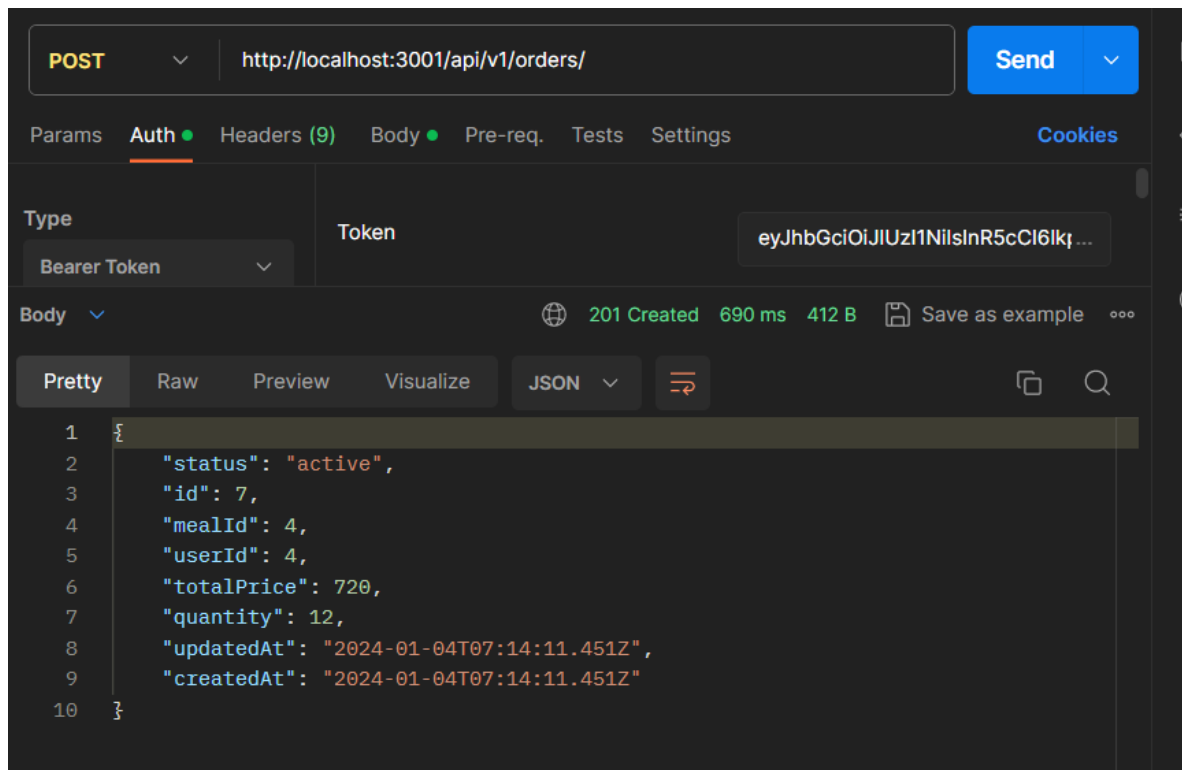


MODULE ORDERS

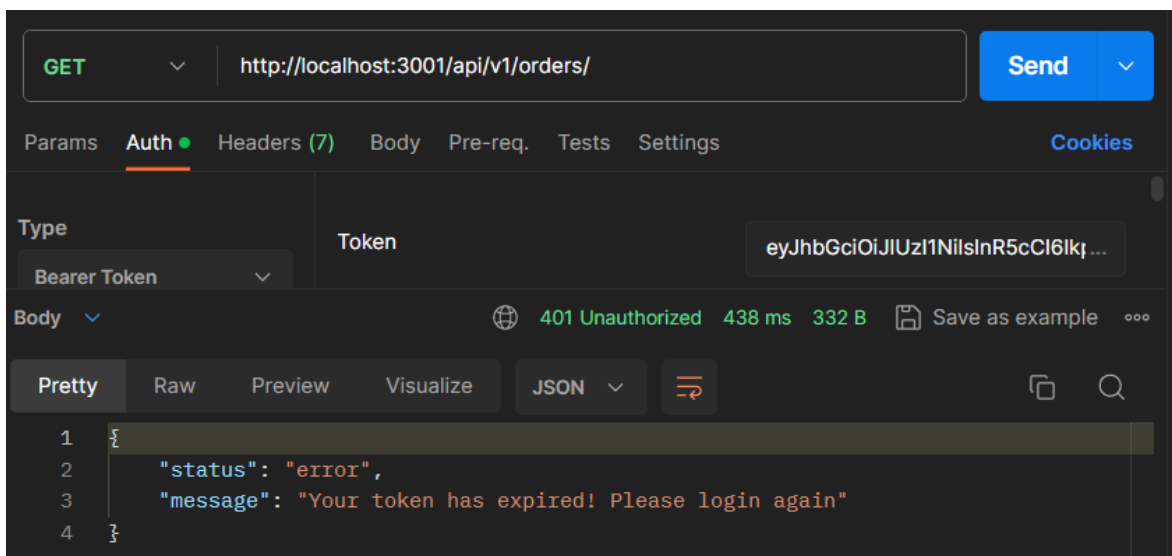
Método Post Crear una nueva order (enviar quantity y mealId por req.body) Validación 1



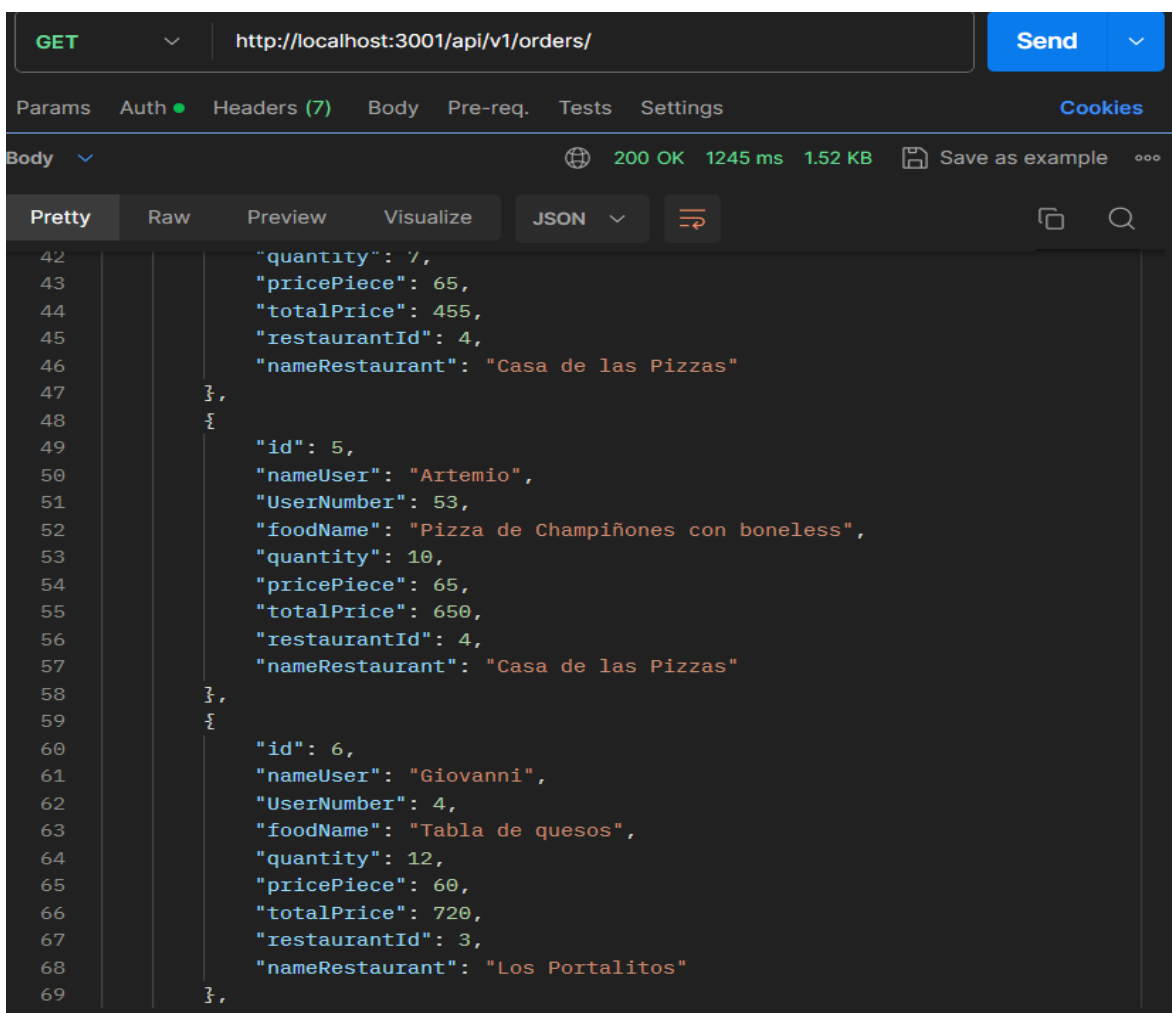
Método Post Crear una nueva order (enviar quantity y mealId por req.body)



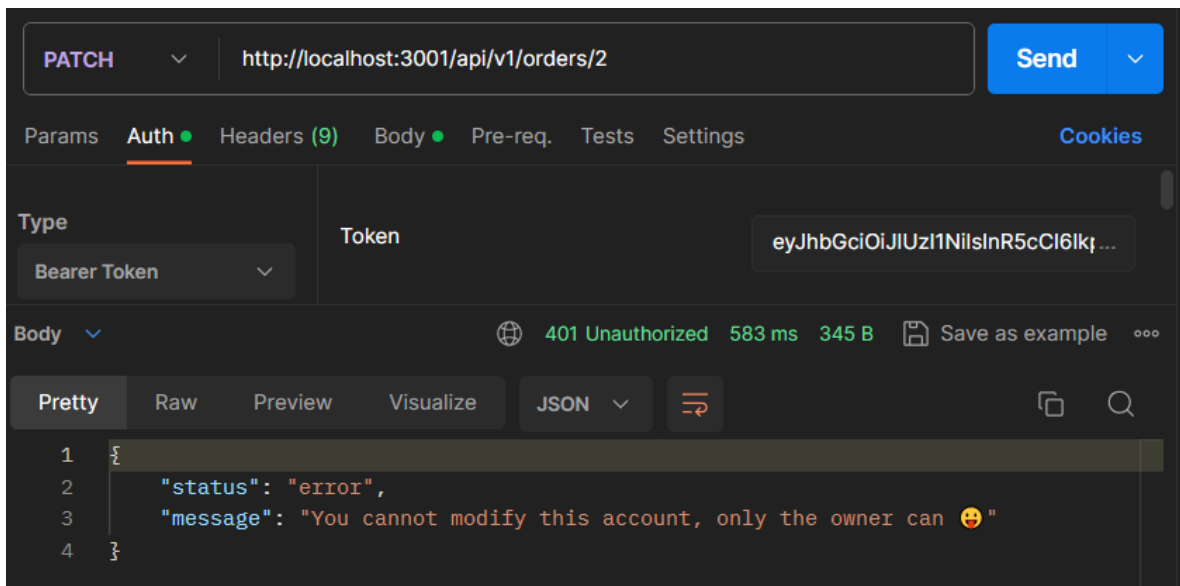
Método Get Obtener todas las órdenes del usuario Validación 1



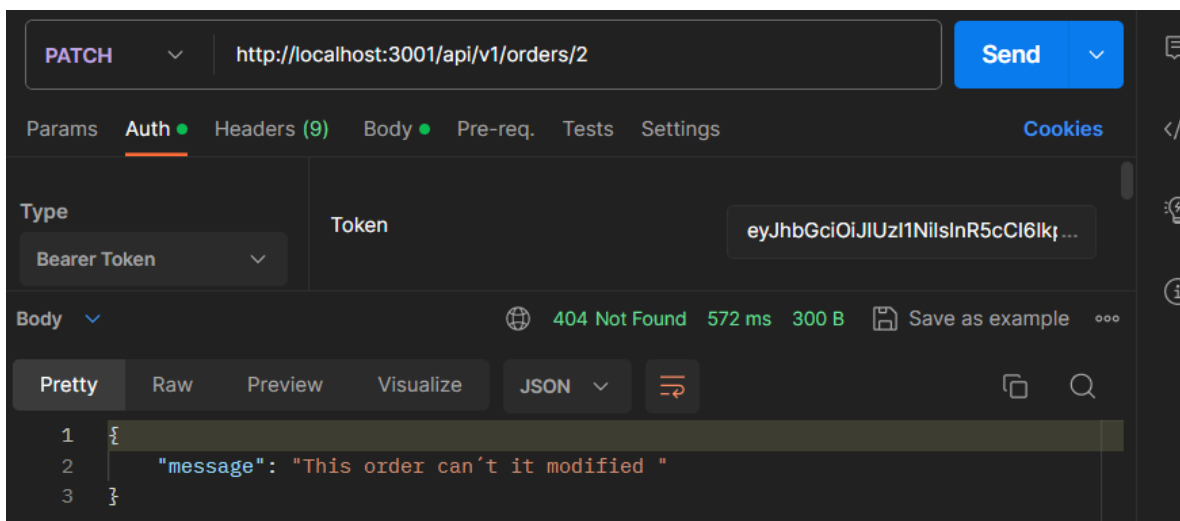
Método Get Obtener todas las órdenes del usuario.



Método Patch: Marcar una orden por id con status completed Validación 1



Método Patch: Marcar una orden por id con status completed Validación 2



Método Patch: Marcar una orden por id con status completed

The screenshot shows a REST client interface with the following details:

- Method:** PATCH
- URL:** http://localhost:3001/api/v1/orders/6
- Auth:** Bearer Token (Token: eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp1...
- Status:** 200 OK, 741 ms, 450 B
- Body (JSON):**

```
{
  "message": "Info changed 😊",
  "order": {
    "id": 6,
    "mealId": 4,
    "userId": 4,
    "totalPrice": 720,
    "quantity": 12,
    "status": "completed",
    "createdAt": "2024-01-03T23:01:16.975Z",
    "updatedAt": "2024-01-04T07:26:08.195Z"
  }
}
```

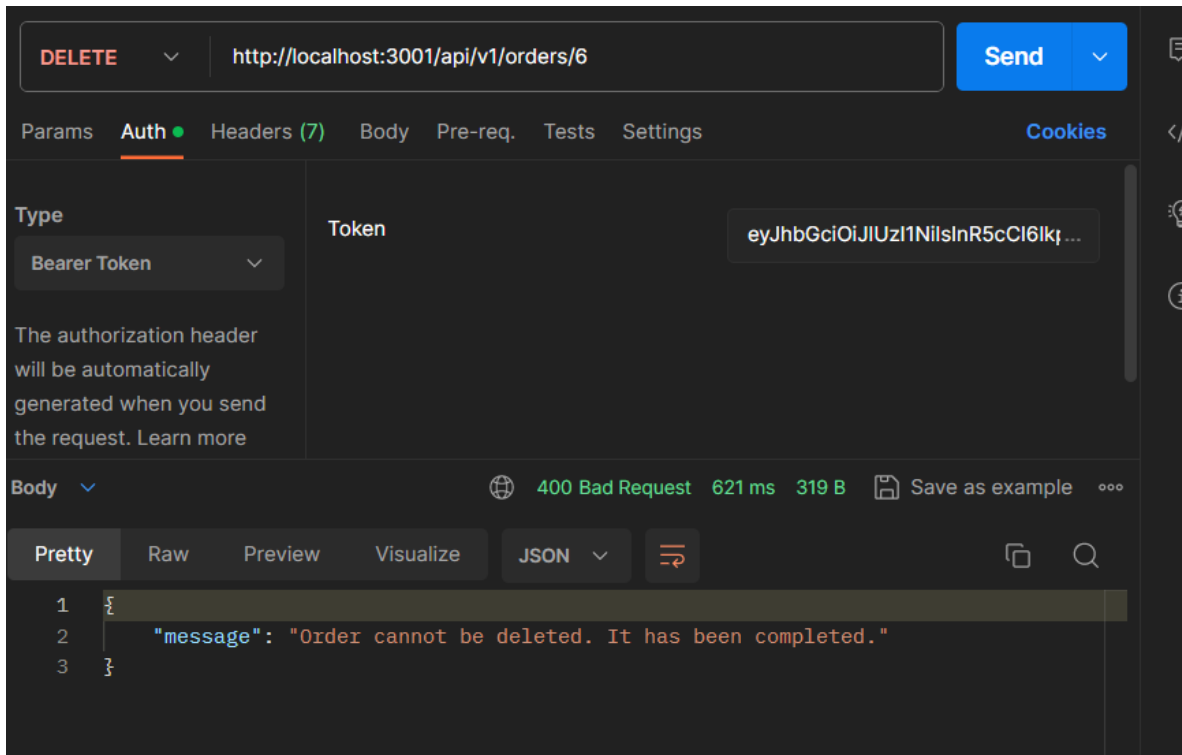
Método Delete: Validación 1

The screenshot shows a REST client interface with the following details:

- Method:** DELETE
- URL:** http://localhost:3001/api/v1/orders/5
- Auth:** Bearer Token (Token: eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp1...)
- Status:** 401 Unauthorized, 473 ms, 332 B
- Body (JSON):**

```
{
  "status": "error",
  "message": "Your token has expired! Please login again"
}
```

Método Delete: Validación 2



Método Delete:

