

Arquitectura Web

# **Informe Trabajo Practico**

---

Lucas Javier Martinez

Primer Cuatrimestre de 2025

## Índice de contenidos

Índice de contenidos .....	1
Enunciado .....	2
Descripción de la aplicación.....	4
Decisiones técnicas y justificaciones .....	5
Especificaciones endpoints .....	6
Testing sobre endpoints.....	8

## Enunciado

Desarrollar una aplicación web de su interés, cumpliendo las funcionalidades descritas en el punto siguiente. La misma debe cumplir con los estándares según los temas vistos en los módulos. Los lenguajes de programación utilizados para el desarrollo son de su elección, debe justificar por qué lo selecciona. Algunos ejemplos de aplicaciones: manejo de stock, administración de usuarios, etc.

Funcionalidad que debe contener la aplicación web a desarrollar:

### 1. FrontEnd:

Se encargará de la presentación solamente. Se puede utilizar alguna library/framework (React, Vue, Angular) o directamente en vanilla JS. Se evaluará el correcto uso de HTML y CSS también.

1.1. Se debe implementar al menos 1 CRUD o ABM.

1.2. Se debe implementar al menos 1 pantalla de reporte.

La implementación de login es completamente opcional.

### 2.Backend:

Implementa la lógica de negocio y la persistencia de datos.

2.1. Debe exponer una RESTful API (nivel 2 mínimo) para consumir los distintos servicios. Acá se evalúa el diseño y las buenas prácticas.

2.1. Para la persistencia se puede utilizar algún motor de base de datos a elección o directamente persistir en memoria. En ambos casos, cuando la aplicación se levante, deberá cargar un set de datos por default para poder testear correctamente.

### 3.Testing:



3.1. Realizar testing sobre los endpoints. Pueden ser utilizando alguna herramienta para tal fin, o bien codificados. No realizar unit testing en el frontend.

#### 4.Build y Ejecución

4.1 Incluir un readme con los pasos para hacer un build y ejecutar el proyecto. Incluir algún script o hacer uso de alguna herramienta (npm por ejemplo).

Presentación y entrega:

Carátula.

Descripción de la aplicación desarrollada (no olvidar el alcance).

Justificación de la arquitectura, lenguajes de programación y toda decisión en torno al proyecto.

Por cada endpoint del backend: Endpoint, verbo HTTP, descripción.

Archivo comprimido .zip con el código fuente y el documento (no se aceptan .rar).

## Descripción de la aplicación

La aplicación desarrollada consiste en una lista de tareas (o todo list en ingles), la misma tiene por objetivo facilitar la organización del usuario, permitiéndole crear carpetas e items dentro de cada carpeta, cada item puede ser marcado y desmarcado como completado. Las funcionalidades que se ofrecen son las de alta, baja y modificación tanto de carpetas como de items, en lo que respecta a la modificación de carpetas puede ser mediante el nombre o el agregado de items, por otro lado, los items pueden cambiar su descripción o marcarse como finalizados.

Por otro lado, se le ofrece al usuario la posibilidad de ordenar las carpetas por nombre o cantidad de items tanto de manera ascendente como descendente, análogamente es posible ordenar items por nombre o por fecha de creación, además con respecto a los filtros se pueden filtrar entre completados o pendientes.

## Decisiones técnicas y justificaciones

Se adoptó una arquitectura cliente-servidor, lo que facilitó la separación de responsabilidades y el desacoplamiento entre el Front-end y el Back-end. Esta decisión permitió centralizar la gestión de datos en el servidor, lo que simplificó el control de acceso y mejoró la organización del sistema.

Además, al tener un Back-end independiente y estandarizado mediante una API RESTful, se abre la posibilidad de escalar la aplicación en el futuro. Por ejemplo, se podría incorporar fácilmente soporte para clientes móviles, ya que estos podrían comunicarse con el servidor sin inconvenientes a través del protocolo HTTP.

Las tecnologías elegidas para el backend fueron el framework springboot con el lenguaje de programación JAVA. La principal razón de la elección de este stack tecnológico es debido a la familiaridad que tenía con este mismo, y por lo tanto acelero considerablemente los tiempos de desarrollo. Por otro lado, springboot ofrece algunas ventajas para el desarrollo de API rest como lo son las annotation para realizar controller rest, manejo de concurrencia y un ecosistema amplio (ademas de springboot, la familia spring incluye springdata, springsecurity, springbatch, etc). Ademas de esto, no debemos olvidar que JAVA es un lenguaje fuertemente tipado, cosa que lenguajes como python y javascript no cumplen, esto es muy util ya que nos salva de errores de tipado en tiempos de ejecucion.

Con respecto a las tecnologias empleadas en el frontend, se opto por la libreria de javascript ReactJS, las ventajas que esta ofrece con respecto a frameworks como angularJS y vueJS, son la libertad arquitectonica, la posibilidad de reutilizar componentes, y gran cantidad de librerias disponibles; todo esto sin mencionar que es el framework mas utilizado en la industria, y que su curva de aprendizaje es mucho mas amigable que la de los otros frameworks, lo que hace sencilla la tarea de desarrollo.

En cuanto a la persistencia de datos, primero decidi utilizar bases relacionales, debido a la consistencia que estas ofrecen, puntualmente se selecciono la base mysql por sobre otras debido a la facilidad de uso, performance en lecturas y facil integracion con frameworks.

En lo que refiere a testing, se utilizo la libreria mockito de java, para poder realizar “mocks” sobre los endpoints y asi validar que estos retornan los valores y excepciones correspondientes.

Por ultimo, se decidio que el despliegue sera utilizando docker, ya que esta herramienta ofrece la posibilidad de utilizar docker-compose, que permite poner en funcionamiento la aplicacion con un simple comando.

## Especificaciones endpoints

**Endpoint:** /folders

**Verbo HTTP:** GET

**Parametros opcionales:** direction y sortBy

**Descripcion:** Devuelve las carpetas ordenadas segun los query params dados, en caso que no se provean query params, devuelve todas las carpetas ordenadas segun el nombre de forma ascendente por defecto

**Endpoint:** /folders

**Verbo HTTP:** POST

**Body:** JSON con campo name (ej: {"name": "test"}).

**Descripción:** Crea una nueva carpeta con el nombre proporcionado.

**Validación:** El nombre no puede estar vacío.

**Endpoint:** /folders/{id}

**Verbo HTTP:** DELETE

**Descripción:** Elimina la carpeta con el ID especificado.

**Endpoint:** /folders/{id}

**Verbo HTTP:** PATCH

**Body:** JSON con campo name (ej: {"name": "nombre\_cambiado"}).

**Descripción:** Actualiza el nombre de la carpeta con el ID especificado.

**Validación:** El nombre no puede estar vacío.

**Endpoint:** /folders/{folderId}/items

**Verbo HTTP:** GET

**Descripción:** Obtiene todos los ítems de la carpeta especificada.

**Parámetros opcionales:** sortBy, direction, state1,state2

**Endpoint:** /folders/{folderId}/items

**Verbo HTTP:** POST

**Body:** JSON con campo description (ej: {"description": "prueba2"}).

**Descripción:** Crea un nuevo ítem en la carpeta especificada.

**Validación:** La descripción no puede estar vacía.

**Endpoint:** /items/{id}

**Verbo HTTP:** PATCH

**Body:** JSON con campo description (ej: {"description": "aaaprueba"}).

**Descripción:** Actualiza la descripción del ítem con el ID especificado.

**Validación:** La descripción no puede estar vacía.

**Endpoint:** /items/{id}/toggle

**Verbo HTTP:** PATCH

**Descripción:** Cambia el estado del ítem (marca como completado o pendiente).

**Endpoint:** /items/{id}

**Verbo HTTP:** DELETE

**Descripción:** Elimina el ítem con el ID especificado.



## Testing sobre endpoints

### Obtencion de carpetas ordenadas por defecto

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

- URL:** `http://localhost:8080/folders`
- Method:** `GET`
- Params:** Authorization, Headers (7), Body, Scripts, Tests, Settings
- Query Params:** A table with columns Key, Value, and Description.
- Body:** JSON format, showing a 200 OK status, 43 ms response time, and 745 B body size.
- JSON Response:** A list of four folder objects, each with name, folder\_id, and itemCount.

```
1 [
2   {
3     "name": "Folder 0",
4     "folder_id": 1,
5     "itemCount": 14
6   },
7   {
8     "name": "Folder 1",
9     "folder_id": 2,
10    "itemCount": 14
11  },
12  {
13    "name": "Folder 2",
14    "folder_id": 3,
15    "itemCount": 14
16  },
17  {
18    "name": "Folder 3",
19    "folder_id": 4,
20    "itemCount": 14
21  },
22 ]
```

## Creación de carpeta

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

- URL Bar:** `HTTP` icon, `todo-list / folders / create folder`
- Method and URL:** `POST` (yellow text), `http://localhost:8080/folders`
- Tab Bar:** `Params`, `Authorization`, `Headers (9)`, `Body` (selected, green dot), `Scripts`, `Tests`, `Settings`
- Body Type Selection:** `none`, `form-data`, `x-www-form-urlencoded`, `raw` (selected, blue dot), `binary`, `GraphQL`
- Request Body (Raw):**

```
1  {  
2    "name" : "test"  
3  }
```
- Response Section:** `Body` (selected), `Cookies`, `Headers (8)`, `Test Results`, `Refresh` icon
- Response Body (JSON):** `{}` JSON icon, `Preview` (play icon), `Visualize` (chart icon), `Dropdown` arrow
- Response Body (JSON):**

```
1  {  
2    "name": "test",  
3    "folder_id": 11,  
4    "itemCount": 0  
5  }
```

## Creación de carpeta con nombre vacío da error

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

- URL:** `http://localhost:8080/folders`
- Method:** `POST`
- Body Type:** `raw` (selected), with `JSON` also visible in the dropdown.
- Request Body:**

```
1 {  
2   "name" : ""  
3 }
```
- Response Body:**

```
1 {  
2   "mensaje": "No pueden haber objetos con nombre vacío"  
3 }
```

The response message indicates that creating an object with an empty name is not allowed.

## Eliminar carpeta

The screenshot shows a REST client interface with a dark theme. At the top, a breadcrumb trail reads 'todo-list / folders / delete folder'. Below this, the HTTP method is set to 'DELETE' and the URL is 'http://localhost:8080/folders/10'. The 'Params' tab is selected, showing an empty table with columns 'Key' and 'Value'. The 'Body' tab is also visible. The response status is '200 OK' with a message 'Carpeta eliminada' in the JSON body. The response is displayed in a code editor with line numbers 1, 2, and 3.

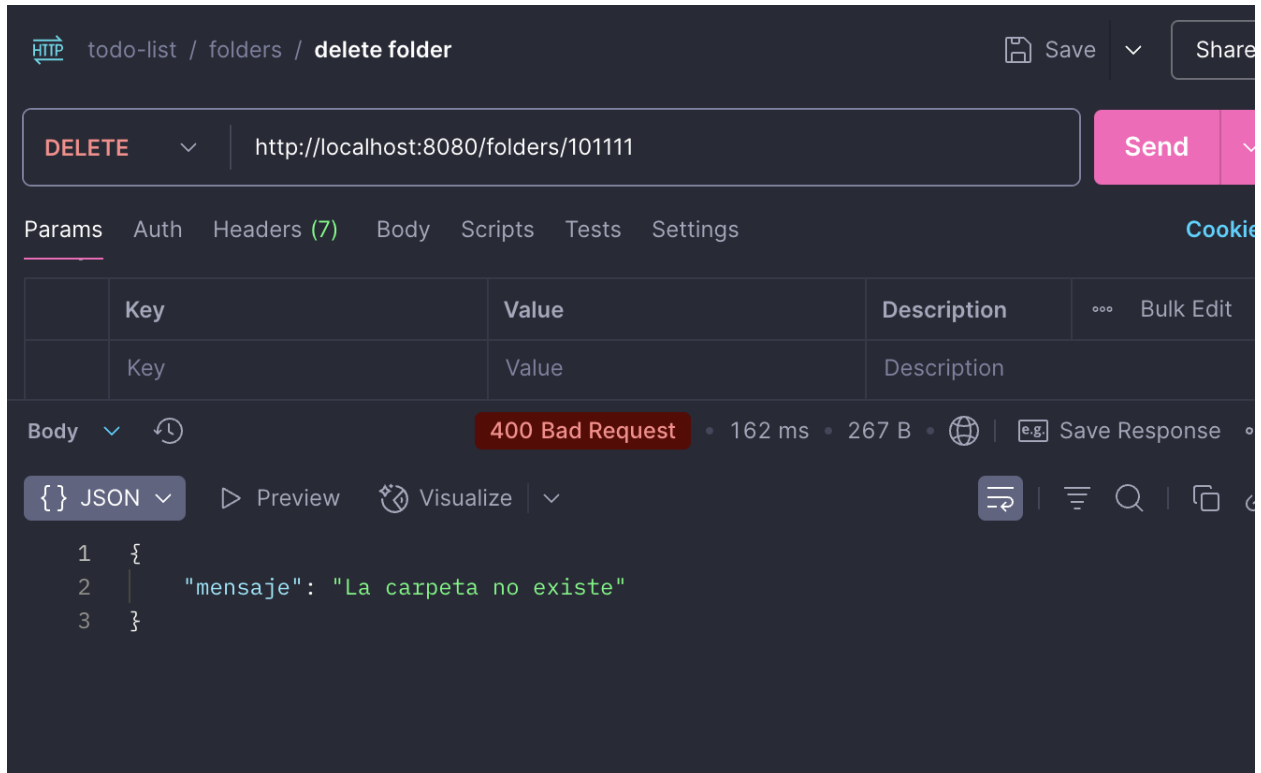
DELETE http://localhost:8080/folders/10

Key	Value	Description
Key	Value	Description

200 OK • 61 ms • 284 B • Save Response

```
1 {  
2   "message": "Carpeta eliminada"  
3 }
```

## No se puede eliminar una carpeta inexistente



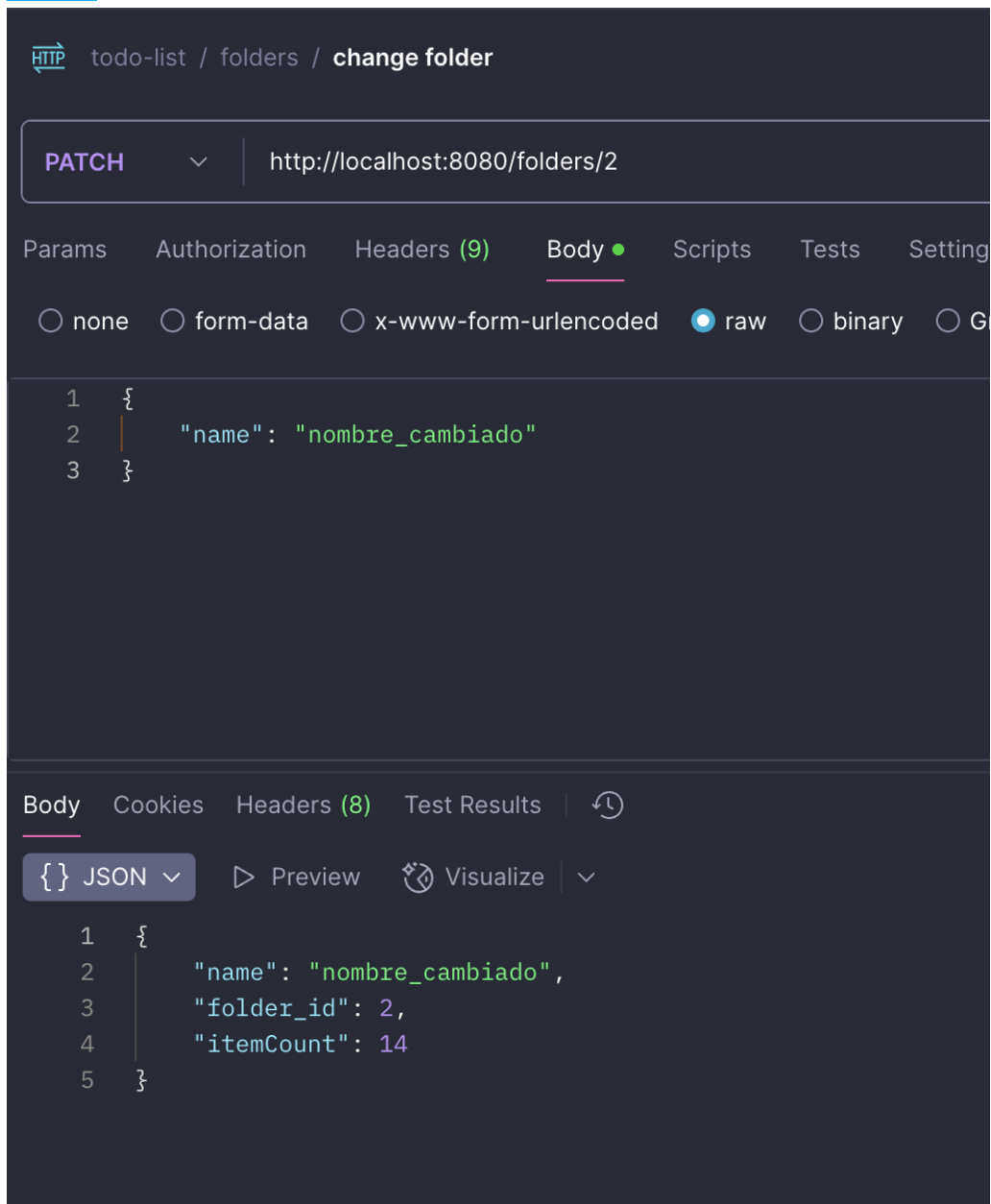
The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8080/folders/101111`
- Method:** `DELETE`
- Status:** `400 Bad Request` (162 ms, 267 B)
- Response Body (JSON):**

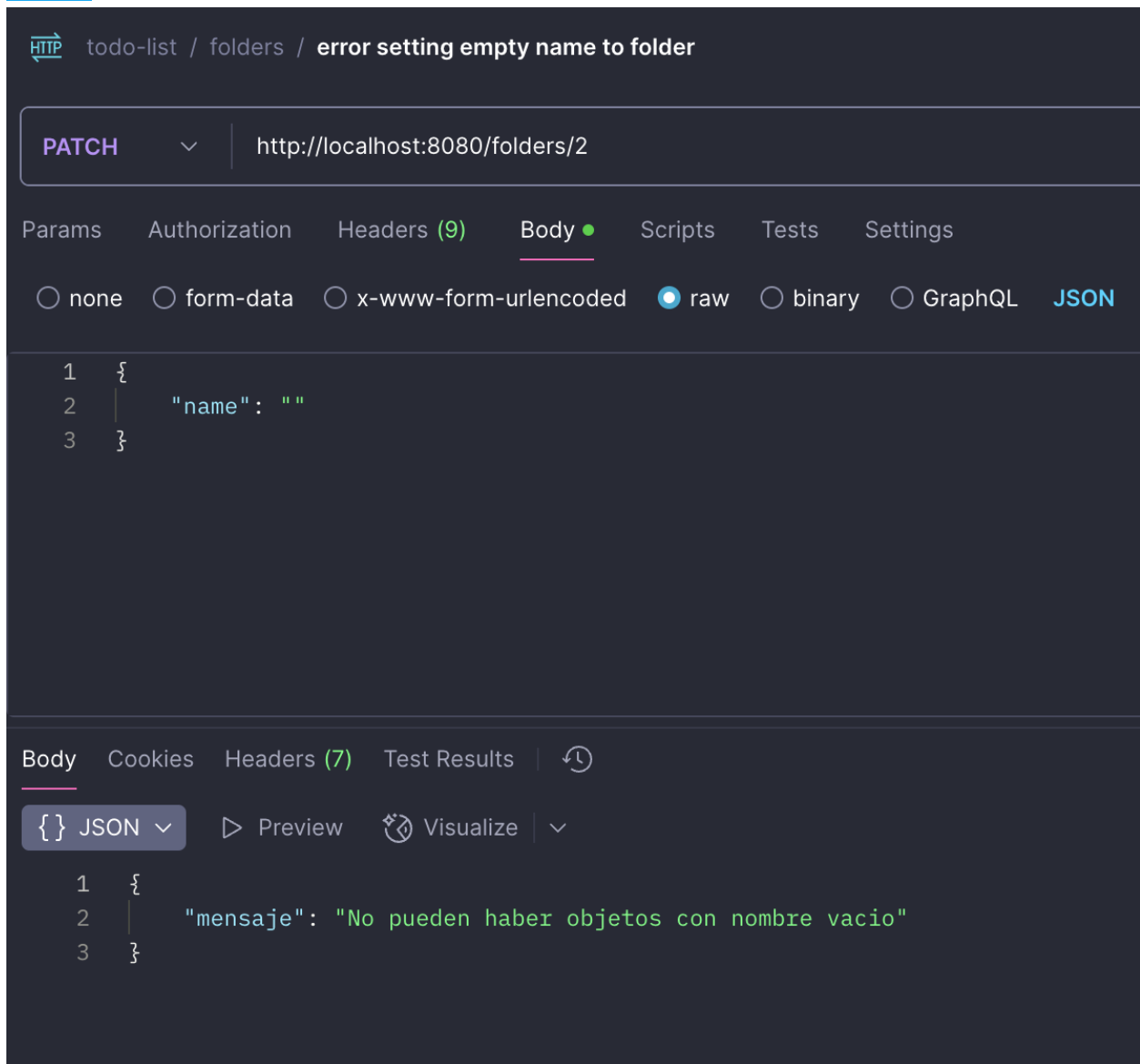
```
{  "mensaje": "La carpeta no existe"}
```

Key	Value	Description
Key	Value	Description

## Se cambia el nombre a una carpeta



**No es posible dejarle el nombre vacío a una carpeta**



**Carpetas ordenadas por nombre**

HTTP todo-list / folders / **get folders ordered by name**

**GET** ▼ http://localhost:8080/folders?direction=desc

Params • Authorization Headers (7) Body Scripts Tests Settings

Query Params

<input checked="" type="checkbox"/>	Key	Value
<input checked="" type="checkbox"/>	direction	desc
	Key	Value

Body Cookies Headers (8) Test Results ↺

{} JSON ▼ ▶ Preview 🔗 Visualize ▼

```
1  [  
2    {  
3      "name": "nombre_cambiado",  
4      "folder_id": 2,  
5      "itemCount": 14  
6    },  
7    {  
8      "name": "Folder 9",  
9      "folder_id": 10,  
10     "itemCount": 10  
11   },  
12   {  
13     "name": "Folder 8",  
14     "folder_id": 9,  
15     "itemCount": 10  
16   },  
17   {  
18     "name": "Folder 7",  
19     "folder_id": 8,  
20     "itemCount": 10  
21   },  
22 ]
```

**Carpetas ordenadas por cantidad de items**



HTTP todo-list / folders / **get folders orderd by amount**

**GET** http://localhost:8080/folders?direction=asc&sortBy=itemCount

Params • Authorization Headers (7) Body Scripts Tests Settings

Query Params

<input checked="" type="checkbox"/> Key	Value
<input checked="" type="checkbox"/> direction	asc
<input checked="" type="checkbox"/> sortBy	itemCount
Key	Value

Body Cookies Headers (8) Test Results ↺

{ } JSON Preview Visualize

```

18     "name": "Folder 8",
19     "folder_id": 9,
20     "itemCount": 10
21   },
22   {
23     "name": "Folder 9",
24     "folder_id": 10,
25     "itemCount": 10
26   },
27   {
28     "name": "Folder 0",
29     "folder_id": 1,
30     "itemCount": 14
31   },
32   {
33     "name": "nombre_cambiado",
34     "folder_id": 2,
35     "itemCount": 14
36   },
37   {
38     "name": "Folder 2",
39     "folder_id": 3,

```

## Obtener items de una carpeta con orden default

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

- URL:** `http://localhost:8080/folders/1/items`
- Method:** `GET`
- Query Params:** A table with 2 columns: `Key` and `Value`.
- Body:** A JSON array of 3 items, each with `item_id`, `description`, `state`, and `createdAt`.

Key	Value
Key	Value

```
[
  {
    "item_id": 10,
    "description": "Item 9",
    "state": "TODO",
    "createdAt": "2025-05-06T14:54:08.195627"
  },
  {
    "item_id": 9,
    "description": "Item 8",
    "state": "TODO",
    "createdAt": "2025-05-06T14:54:08.193008"
  },
  {
    "item_id": 8,
    "description": "Item 7",
    "state": "TODO",
    "createdAt": "2025-05-06T14:54:08.190762"
  },
]
```

## Crear item

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

- URL:** `http://localhost:8080/folders/1/items`
- Method:** **POST**
- Body Type:** **raw** (selected)
- Request Body (JSON):**

```
1  {  
2    "description": "prueba2"  
3  }
```
- Response Body (JSON):**

```
1  {  
2    "item_id": 121,  
3    "description": "prueba2",  
4    "state": "TODO",  
5    "createdAt": "2025-05-06T15:03:34.472483"  
6  }
```

**No es posible crear items con nombre vacio**

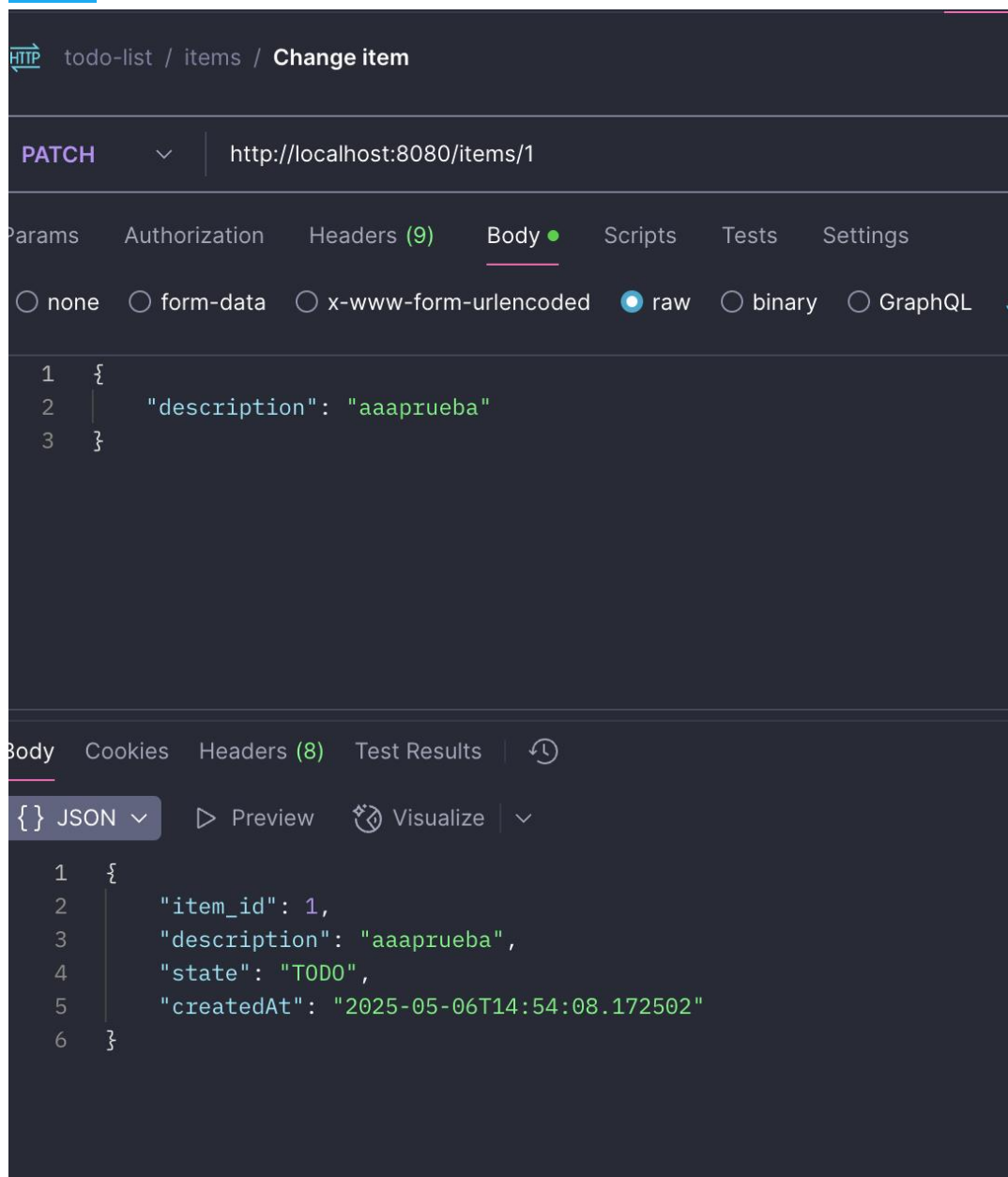
The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8080/folders/1/items`
- Method:** `POST`
- Body Type:** `raw` (selected)
- Request Body:**

```
1 {  
2   "description": ""  
3 }
```
- Response Body:**

```
1 {  
2   "mensaje": "No pueden haber objetos con nombre vacio"  
3 }
```

Se cambia el nombre de un item



**No se puede dejar un item con nombre vacio**

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

- URL:** `http://localhost:8080/items/1`
- Method:** `PATCH`
- Body Type:** `raw` (selected)
- Request Body (JSON):**

```
1 {  
2   "description": ""  
3 }
```
- Response Body (JSON):**

```
1 {  
2   "mensaje": "No pueden haber objetos con nombre vacio"  
3 }
```

**Arroja error al cambiar el estado de un item que no existe**

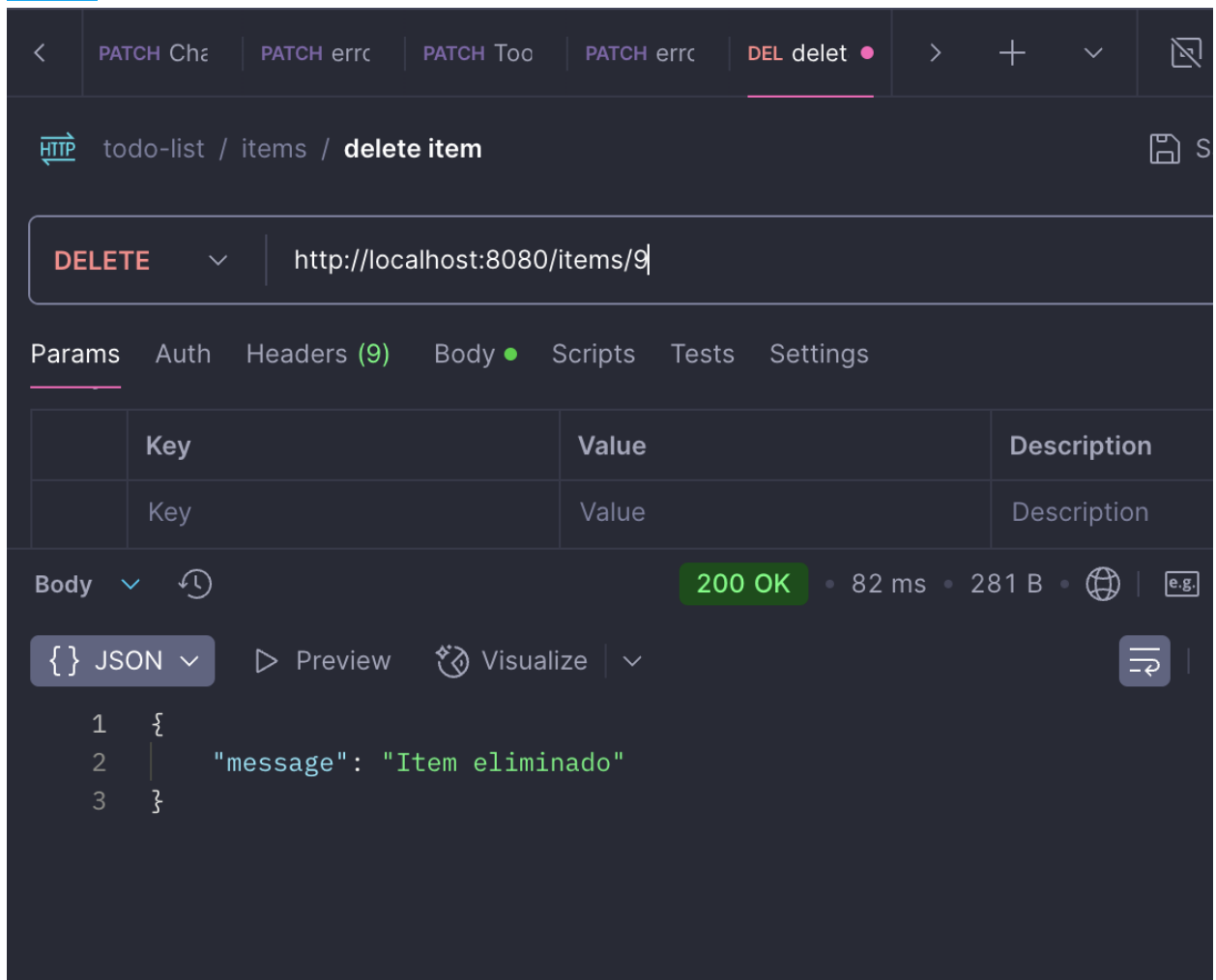
The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8080/items/1111111111/toogle`
- Method:** PATCH
- Headers (9):** (tab selected)
- Body:** (tab selected, showing JSON)
- Response:** A JSON object with the message: `"mensaje": "El Item no existe"`

Key	Value
Key	Value

```
1 {
2   "mensaje": "El Item no existe"
3 }
```

Se elimina un ítem



No es posible eliminar un item inexistente



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

- URL:** `http://localhost:8080/items/10111`
- Method:** `DELETE`
- Params:** A table with 2 columns: `Key` and `Value`.

Key	Value
Key	Value
- Body:** A red banner indicates a `400 Bad Request` error. The response body is shown in JSON format:

```
{  "mensaje": "El Item no existe"}
```

Se obtienen los items ordenados por fecha de creacion

HTTP todo-list / items / **get items from folder ordered by date**

**GET** ⌵ http://localhost:8080/folders/1/items?sortBy=createdAt&direction=desc

Params • Authorization Headers (7) Body Scripts Tests Settings

Query Params

<input checked="" type="checkbox"/>	Key	Value
<input checked="" type="checkbox"/>	sortBy	createdAt
<input checked="" type="checkbox"/>	direction	desc
	Key	Value

Body Cookies Headers (8) Test Results ↺

{ } JSON ⌵ ▶ Preview 🔗 Visualize ⌵

```
1  [  
2    {  
3      "item_id": 121,  
4      "description": "prueba2",  
5      "state": "TODO",  
6      "createdAt": "2025-05-06T15:03:34.472483"  
7    },  
8    {  
9      "item_id": 104,  
10     "description": "Item 14",  
11     "state": "TODO",  
12     "createdAt": "2025-05-06T14:54:08.383984"  
13   },  
14   {  
15     "item_id": 103,  
16     "description": "Item 13",  
17     "state": "TODO",  
18     "createdAt": "2025-05-06T14:54:08.382234"  
19   },  
20   {  
21     "item_id": 102,
```

Se obtienen solo los items pendientes

HTTP todo-list / items / get items from folder done

GET ⌵ http://localhost:8080/folders/1/items?state1=TODO&state2=TODO

Params ● Authorization Headers (7) Body Scripts Tests Settings

Query Params

<input checked="" type="checkbox"/>	Key	Value
<input checked="" type="checkbox"/>	state1	TODO
<input checked="" type="checkbox"/>	state2	TODO
	Key	Value

Body Cookies Headers (8) Test Results ↻

{} JSON ⌵ ▶ Preview 🔗 Visualize ⌵

```
1  [
2    {
3      "item_id": 121,
4      "description": "prueba2",
5      "state": "TODO",
6      "createdAt": "2025-05-06T15:03:34.472483"
7    },
8    {
9      "item_id": 9,
10     "description": "Item 8",
11     "state": "TODO",
12     "createdAt": "2025-05-06T14:54:08.193008"
13   },
14   {
15     "item_id": 8,
16     "description": "Item 7",
17     "state": "TODO",
18     "createdAt": "2025-05-06T14:54:08.190762"
19   },
20   {
21     "item_id": 7,
22     "description": "Item 6",
```