

Implementación API: Gestión layouts

Vamos a explicar como se implementa la arquitectura de gestión de layouts de tiendas de Mango utilizando recurso manejados por AWS:

- **DynamoDB:** base de datos noSQL
- **Lambdas:** funciones implementado el CRUD de layouts de tienda
- **API Gateway:** diseño de endpoints y rutas acceso a la lambdas de forma securizada.

Estos son los pasos a seguir a la hora de aprovisionar la infraestructura antes diseñada:

STEP 01: Creamos una tabla de DynamoDB

Primero se crea la tabla que va a guardar el estado de todos los layouts de las tiendas através de este [link de AWS](#)

Debemos rellenar:

- **Table Name:** Layout
- **Partition key:** id de tipo string. Esta columna representa el indentificador único de la tienda, en nuestro caso será el código de la misma.

Create table

Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name
This will be used to identify your table.

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

1 to 255 characters and case sensitive.

Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

1 to 255 characters and case sensitive.

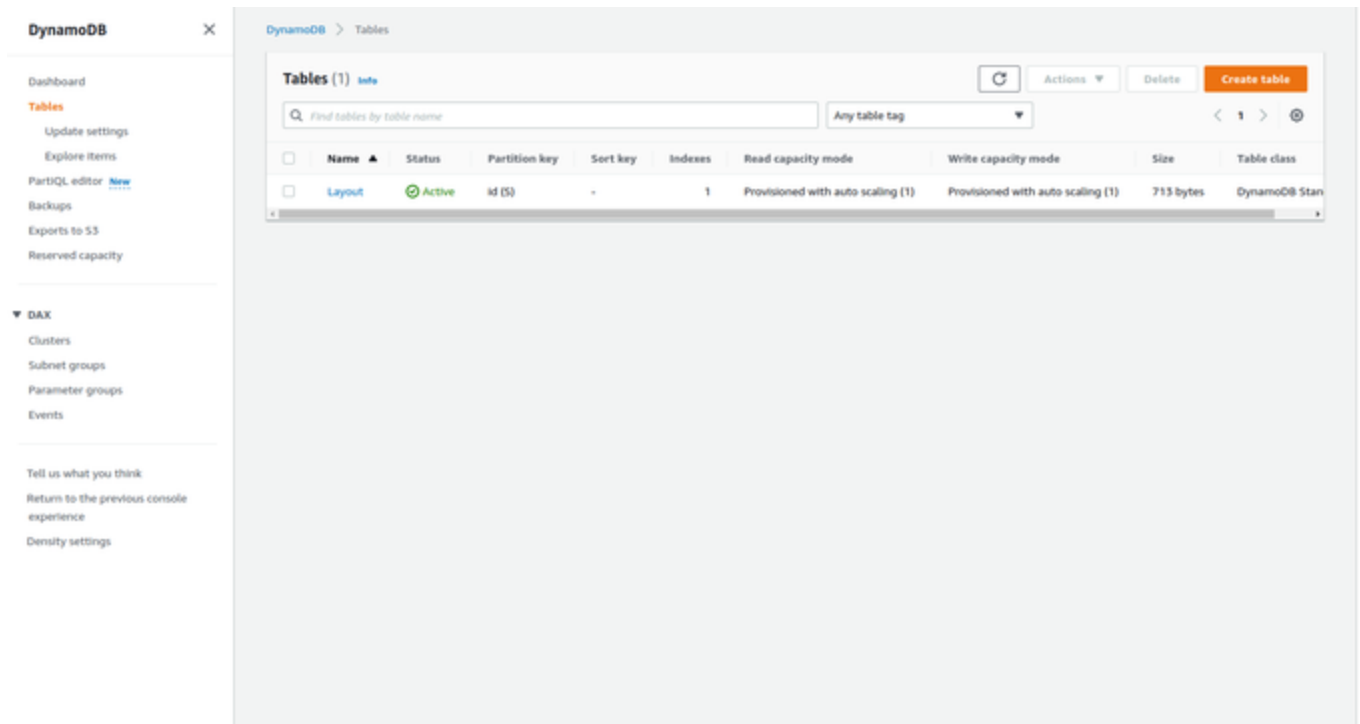
Settings

☒ **Default settings**
The fastest way to create your table. You can modify these settings now or after your table has been created.

☐ **Customize settings**
Use these advanced features to make DynamoDB work better for your needs.

Default settings

Lista de Tablas creadas



STEP 02: Creamos Un lamba implemente el CRUD de Layouts

Creamos este lambda desde este [link de AWS](#).

Introducimos los siguientes campos:

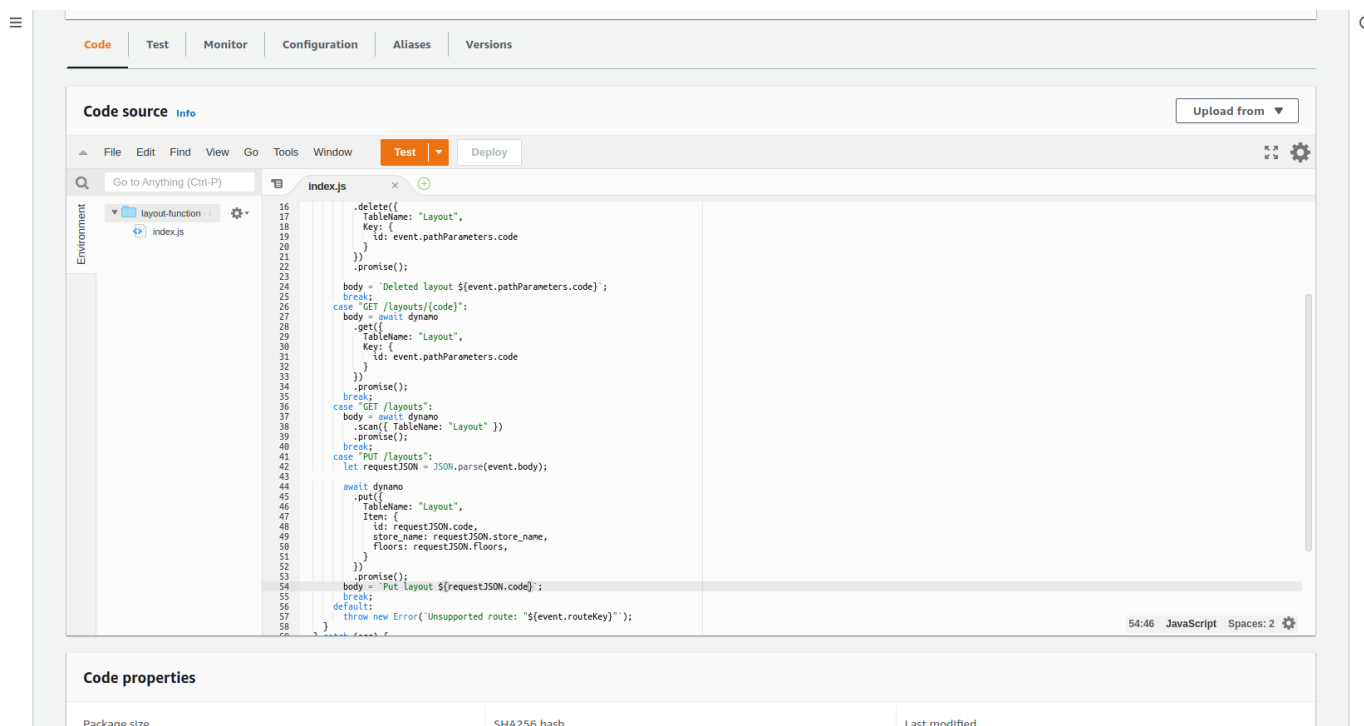
- **Function Name:** layout-function
- **Runtime:** Node.js 14.x
- **Architecture:** x86_64
- **Permission:** creamos un nuevo role en AWS apartir de las políticas de templates de AWS para que este lambda pueda acceder a DynamoDB. Seleccionamos la política llamada: **Simple microservices permissions**

STEP 03: Implementamos el lambda

Implementamos el lambda siguiendo esta tabla de endpoints:

Nom bre	Endpoint	Scope	URI Para ms	Body	Descripción
findAll	<LAMBDA_URI>/layouts	GET			Recuperar todos los layouts
findByld	<LAMBDA_URI>/layouts/{id}	GET	MANGO002		Recupera un layout dado su identificador único (código de tienda)

save	<LAMBDA_URI> /layouts	PUT		<pre> { "id": "MANGO002", "store_name": "Gijon, Calle Corrida N1", "floors": [{ "floor_name": "Planta Caballeros", "areas": [{ "area_name": "Areal", "area_port": "81", "area_IP": "192.168.1.1", "enable": true, "enter_delay_in_secs": 10, "exit_delay_in_secs": 12, "fitting_rooms": [{ "name": "1", "sensor_index": "1", "enable": true }, { "name": "2", "sensor_index": "2", "enable": true }] }] }] } </pre>	Guarda o actualizar un layout. Si no existe el identificador único crea el layout, si existe actualiza el layout.
delete	<LAMBDA_URI> /layouts/{id}	DELETE	MANGO 002		Borra un layout dado su identificador único (código de tienda)

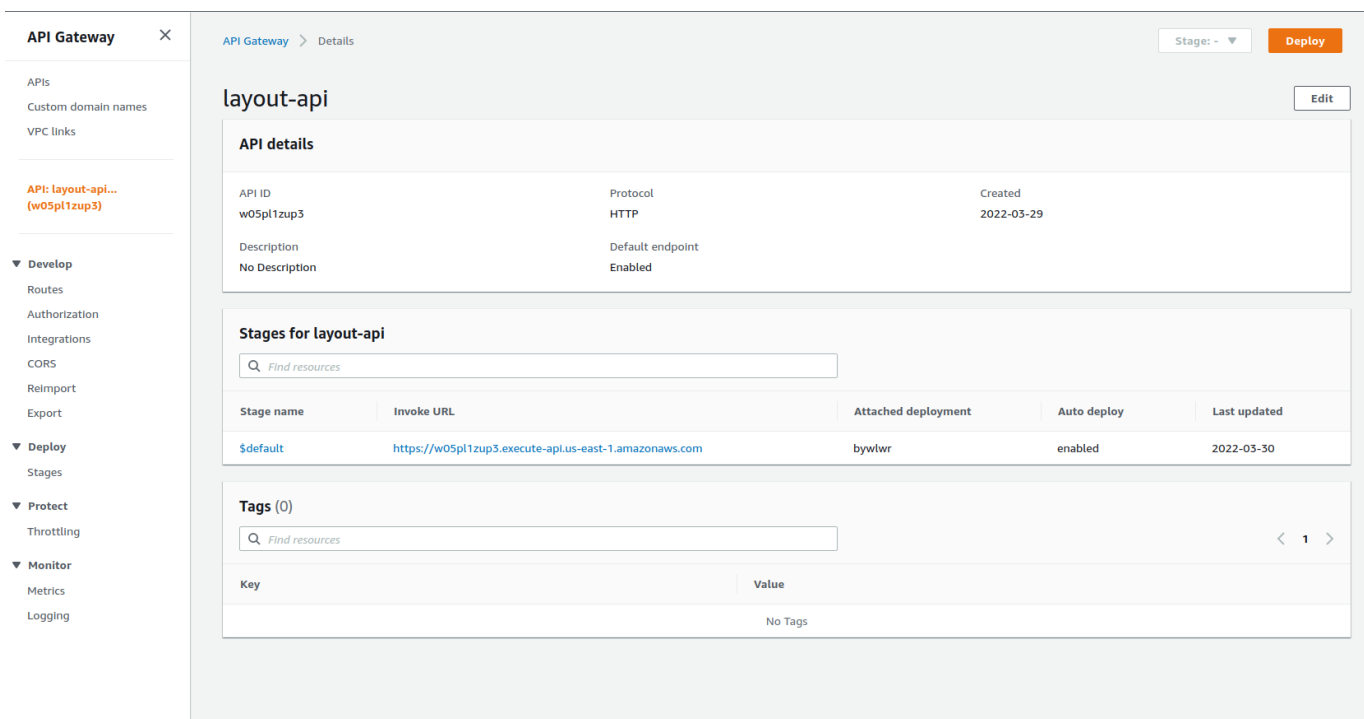


STEP 04: Creamos un API

Los API proporcionan un punto de enlace HTTP para la función de Lambda. Crearemos este API desde este [link de AWS](#).

Introducimos los siguientes campos:

- **Seleccionamos tipo de API:** API HTTP
- **API name:** layout-api
- Dejamos la **ruta** para ser configurar posteriormente



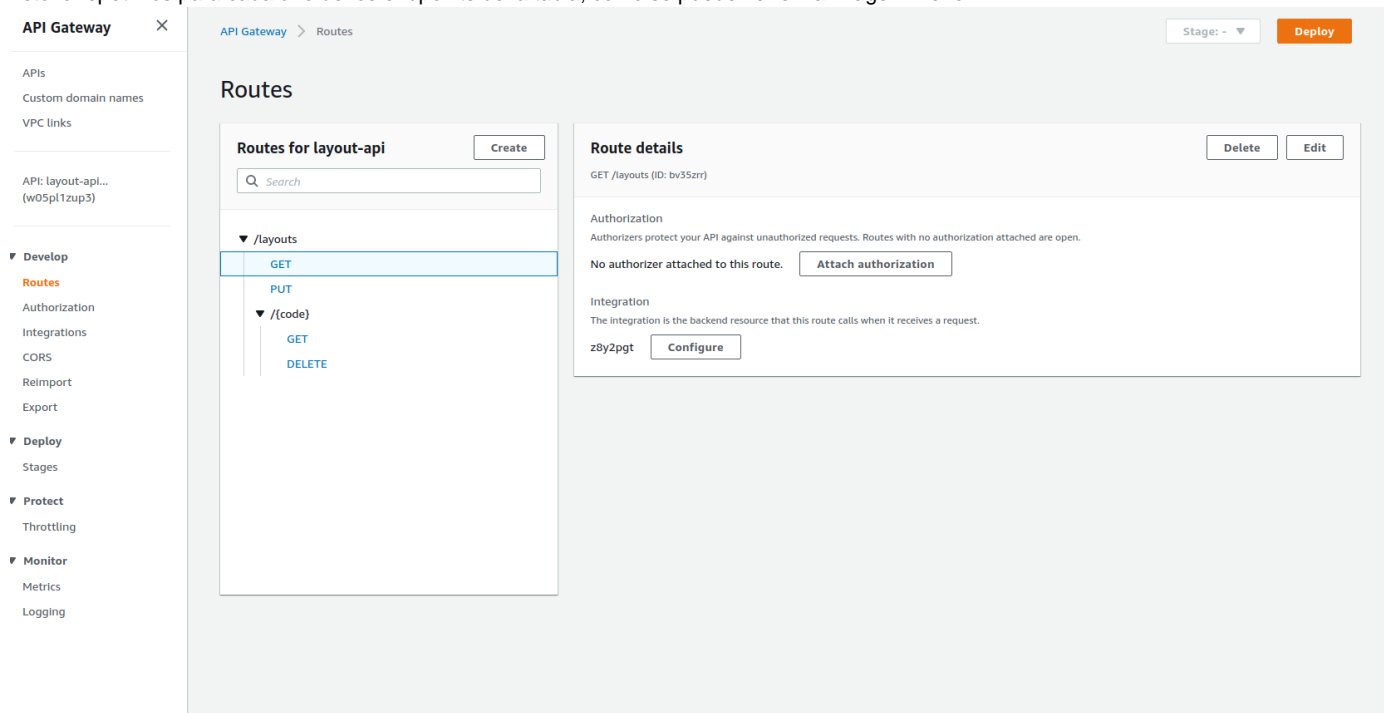
STEP 05: Creamos las rutas del API

Seleccionamos el API HTTP antes creado y dentro del mismo escogemos la opción de rutas y creamos las mismas basándonos en la tabla de endpoints antes diseñada. Por ejemplo para crear el endpoint de tipo findById.

- **Escogemos el método:** GET

Diseñamos la ruta: /layouts/{id}

Esto lo repetimos para cada uno de los endpoints de la tabla, como se puede ver en la imagen inferior:



STEP 06: Creamos las integraciones de la API

Ahora que ya tenemos nuestra función lambda implementado el CRUD, vamos a crear una integración que utilice esta función lambda.

- **Seleccionamos tipo de integración:** Función Lambda
- **Escogemos el lambda que implementa la integración:** layout-function

STEP 07: Conectamos la integración a las rutas

Ahora con la integración creada la conectamos a cada una de las rutas antes creadas:

- Seleccionamos cada una de las rutas desde la opción de rutas y vamos asignando a cada una de ellas la integración antes creada asociada la función lambda llamada layout-function a cada una de las rutas. Al final veremos como todas las rutas ya tienen esa integración asociada como vemos en la imagen inferior:

API Gateway

APIs

Custom domain names

VPC links

API: layout-api... (w05pl1zup3)

Develop

Routes

Authorization

Integrations

CORS

Reimport

Export

Deploy

Stages

Protect

Throttling

Monitor

Metrics

Logging

API Gateway > Integrations

Stage: -

Deploy

Integrations

Attach integrations to routes

Manage integrations

Routes for layout-api

Search

/layouts

GET AWS Lambda

PUT AWS Lambda

/[code]

GET AWS Lambda

DELETE AWS Lambda

Integration details for route

Detach integration

Manage integration

GET /layouts (bw35zrr)

Lambda function

layout-function (us-east-1)

Integration ID

z8y2pgt

Description

-

Payload format version

The parsing algorithm for the payload sent to and returned from your Lambda function. [Learn more.](#)

2.0 (interpreted response format)

Invoke permissions

The resource policy of the Lambda function determines if API Gateway can invoke it. You can run the AWS CLI command snippet below to give API Gateway permission to invoke your AWS Lambda function.

Example policy statement

Timeout

The number of milliseconds that API Gateway should wait for a response from the integration before timing out.

30000

Request parameter mapping

Not configured

Response parameter mappings

Not configured

STEP 08: Probar la API

Ahora que ya tenemos implementada la integración con DynamoDB con una función lambda en Node.js y creadas las rutas con todos los métodos y conectadas a esta función lambda con una integración de API, podemos probarlas. Lo primer es conocer el URI ofrecido por el API HTTP para lanzarle peticiones, este dato lo podemos conocer seleccionando nuestro API recién creado bajo el título **Invoke URL**, como se puede ver en la imagen inferior:

APIs

Custom domain names

VPC links

API: layout-api... (w05pl1zup3)

Develop

Routes

Authorization

Integrations

CORS

Reimport

Export

Deploy

Stages

Protect

Throttling

Monitor

Metrics

Logging

layout-api

Edit

API details

API ID

w05pl1zup3

Protocol

HTTP

Created

2022-03-29

Description

No Description

Default endpoint

Enabled

Stages for layout-api

Find resources

Stage name	Invoke URL	Attached deployment	Auto deploy	Last updated
\$default	https://w05pl1zup3.execute-api.us-east-1.amazonaws.com	bywlwr	enabled	2022-03-30

Tags (0)

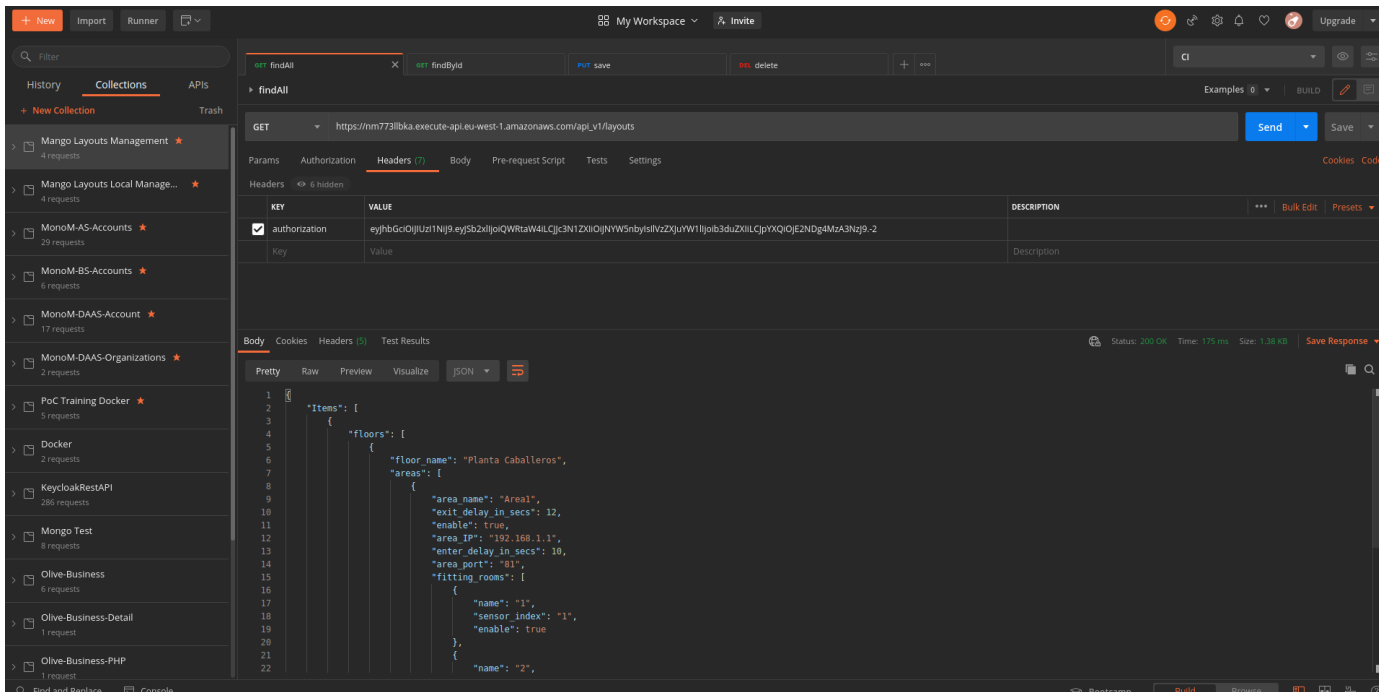
Find resources

< 1 >

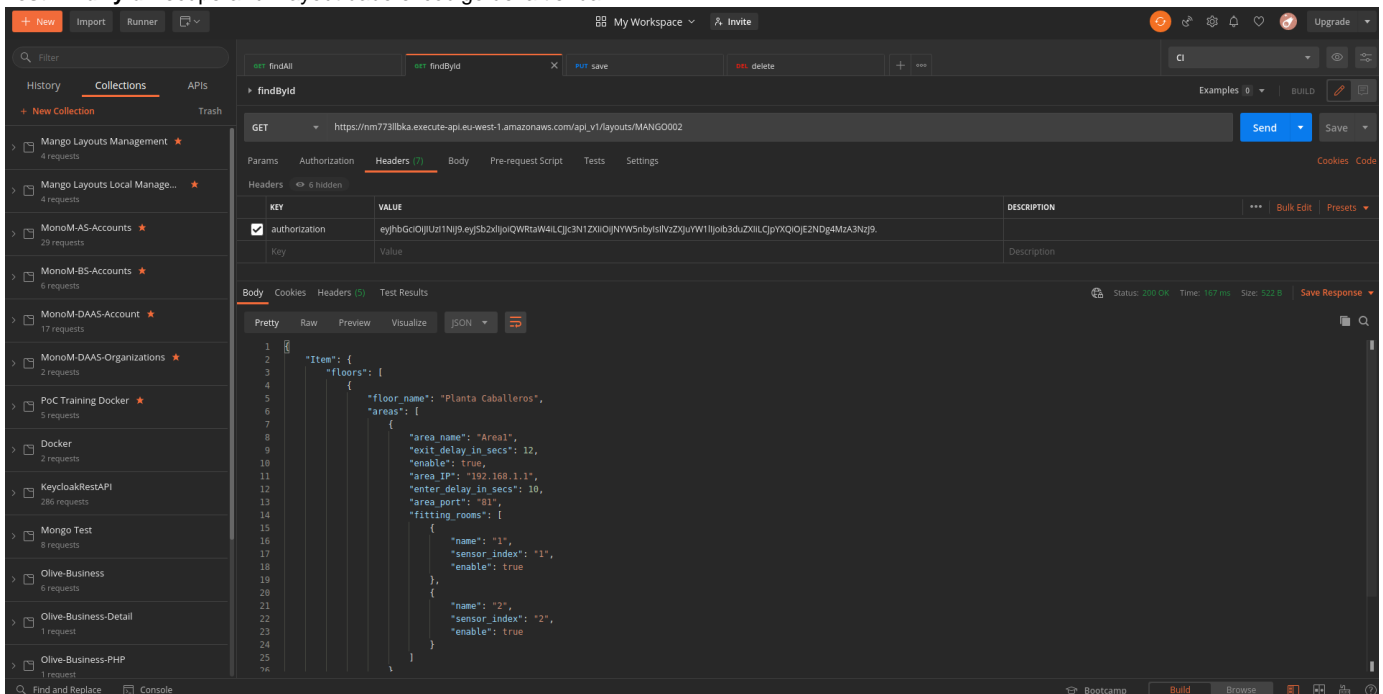
Key	Value
-----	-------

Usaremos Postman para probarlo e implementaremos todos los métodos ofrecidos por el API HTTP configurado previamente:

Test findAll: recuperar todas las tiendas configuradas

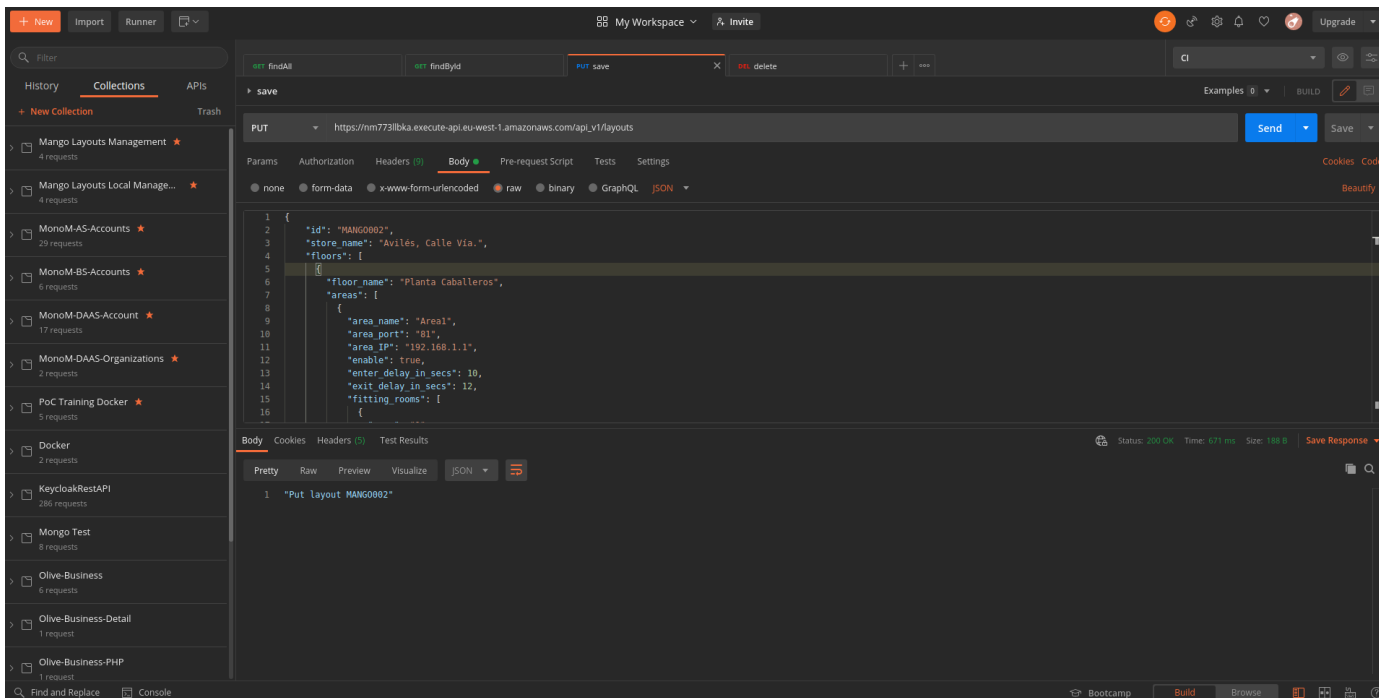


Test findById: recuperar un layout dado el código de la tienda

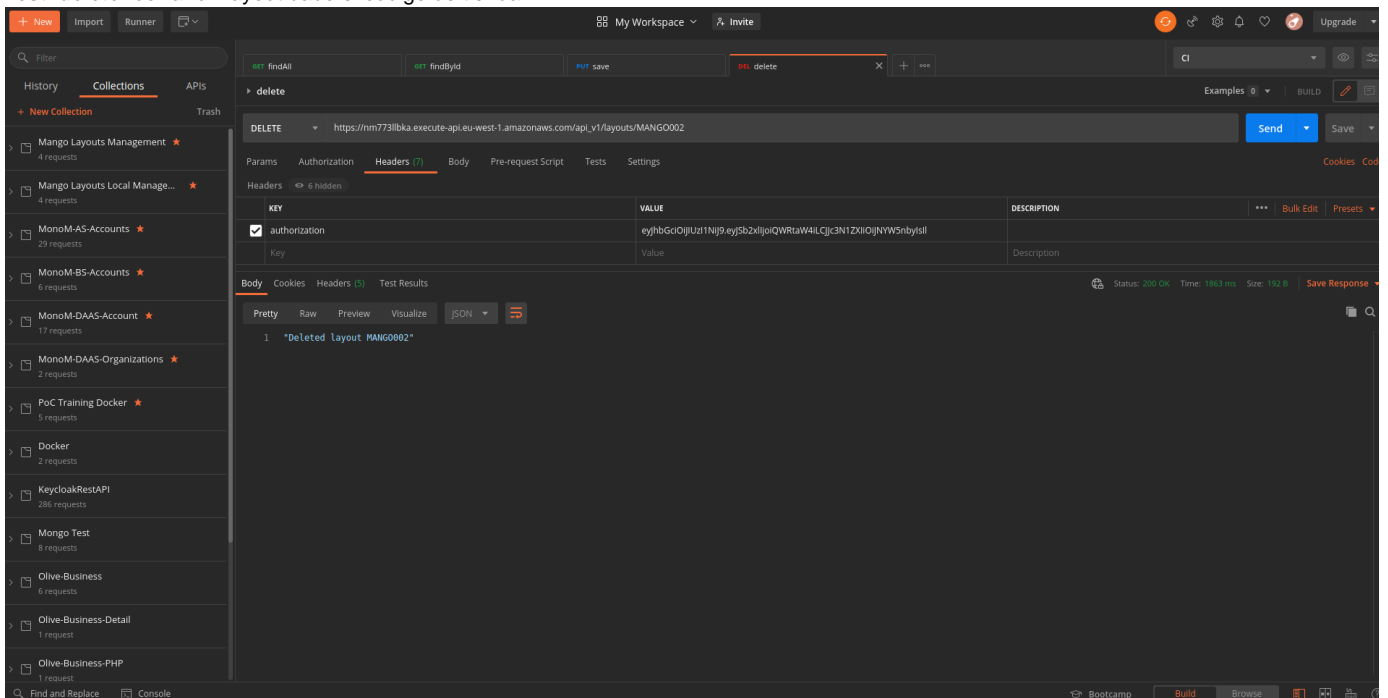


Test save: crea o actualizar un layout dato un objeto tienda .Este objeto debe contener:

- **id [string]:** string indicando el identificador único de tienda.
- **store_name [string]:** descripción de la tienda.
- **floors [array JSONs]:** colección de layouts asociados a la tienda.



Test delete: borrar un layout dado el código de tienda



Desde este link se puede importar los endpoints de Postman del CRUD



Mango Layouts ..._collection.json