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**: functiones implementado el CRUD de layouts de tienda
- **API Gateway**: diseño de endpoints y rutas acceso a la lambdas de forma securizada.

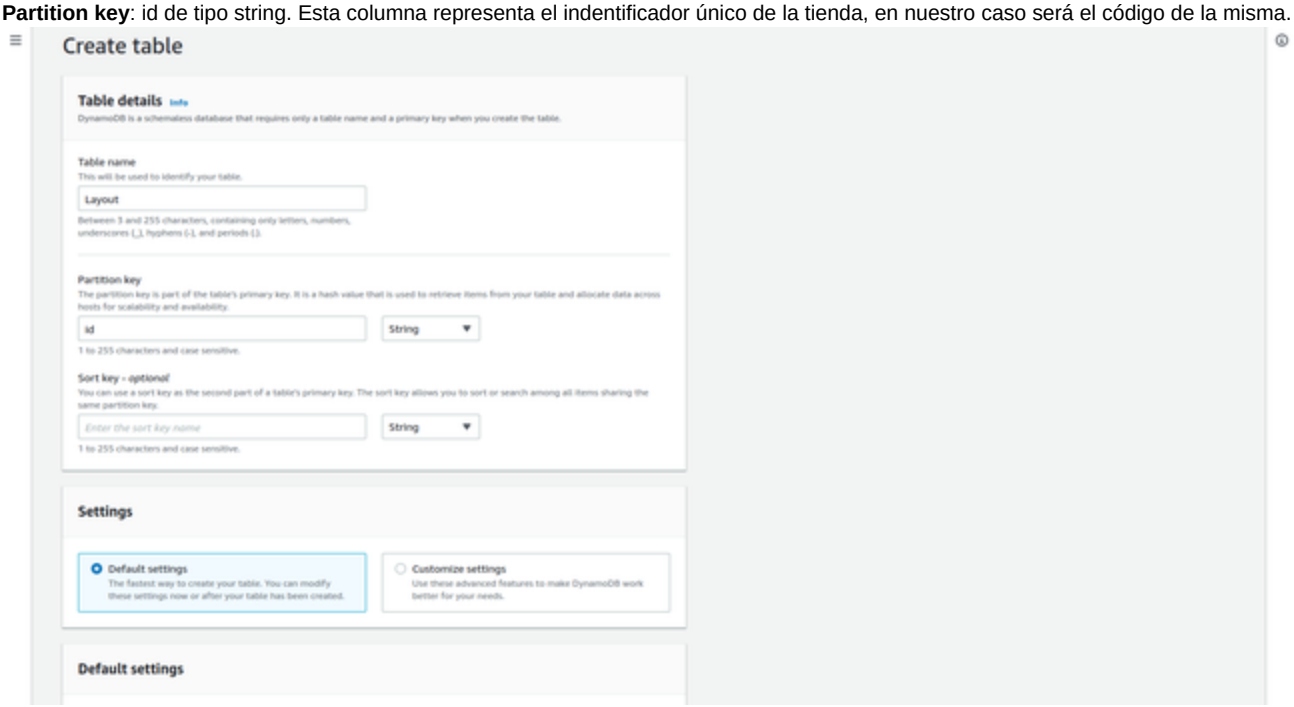
Estos son los pasos a seguir a la hora de aprovisionar la infraestructura antes diseñada:

STEP 01: Creamos una tabla de DynamoDB

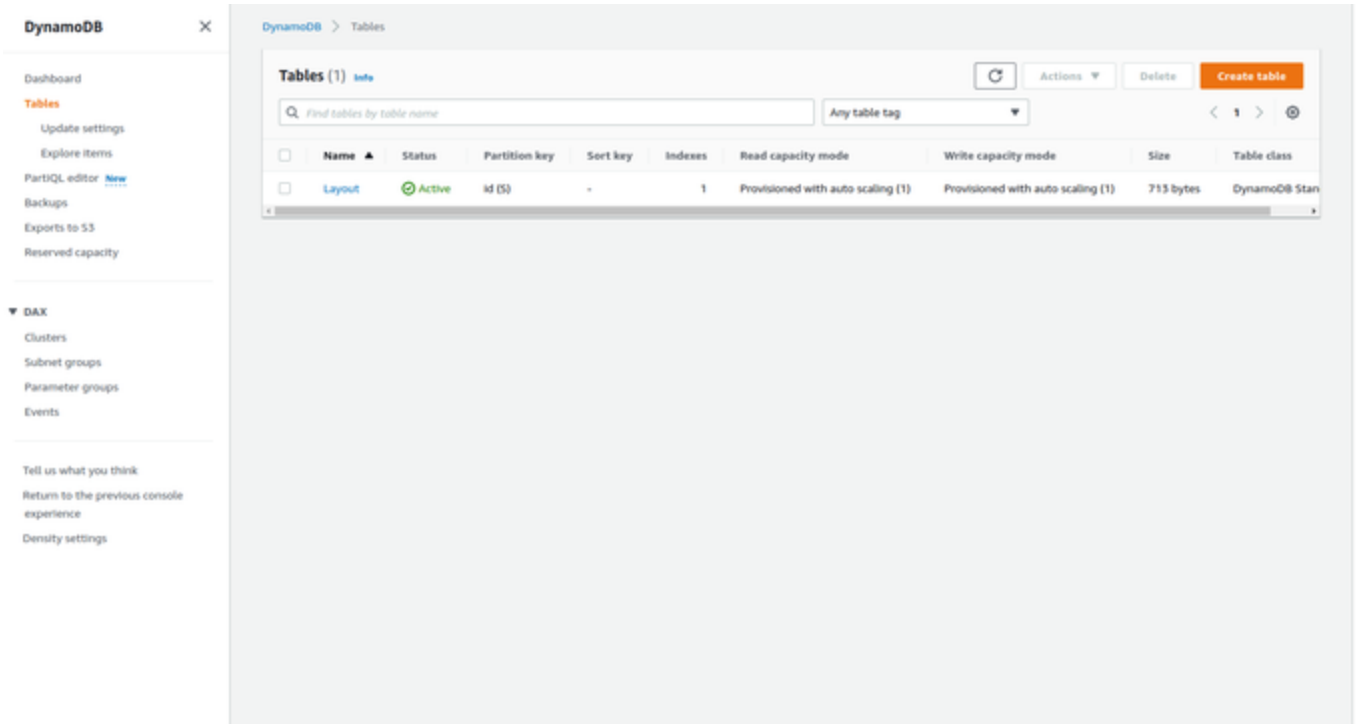
Primero se crea la tabla que va a guardar el estado de todos los layouts de las tiendas através de este link de AWS

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.



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**

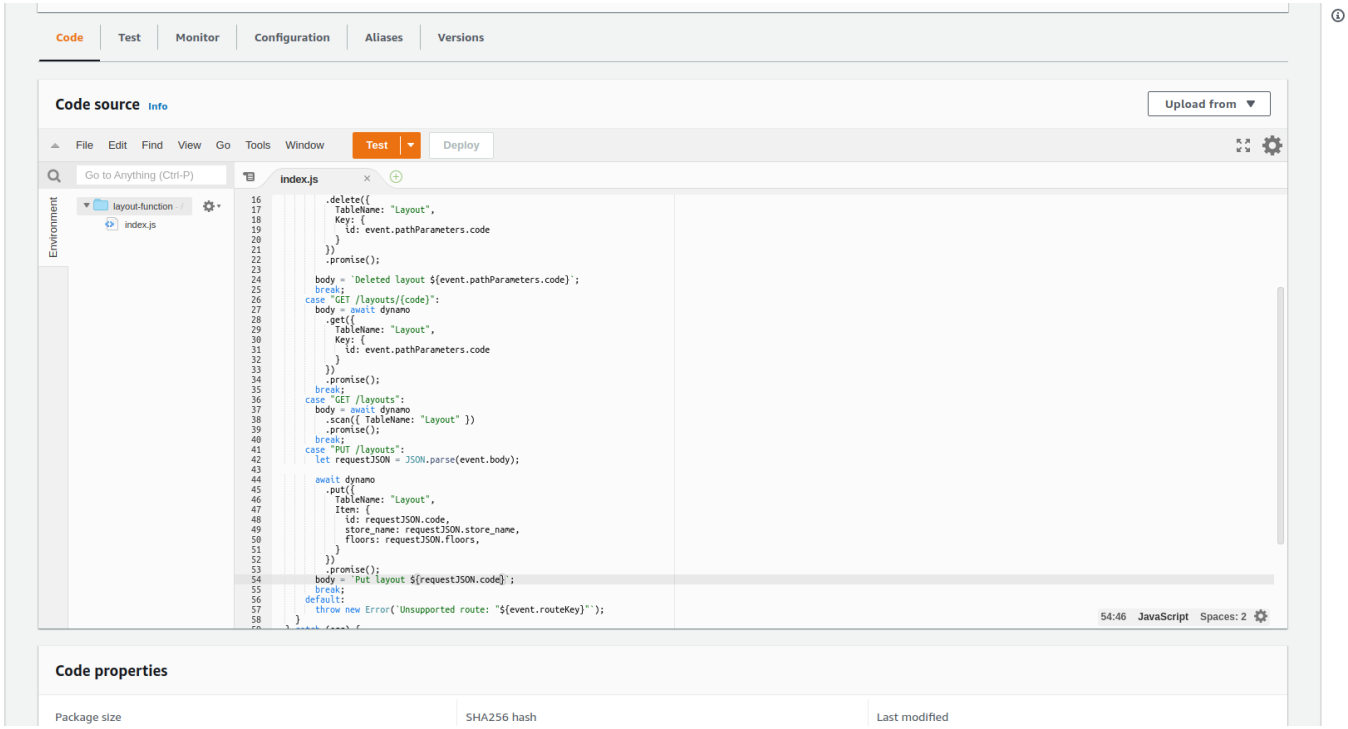
STEP 03: Implementamos el lambda

Implementamos el lambda siguiendo esta tabla de endpoints:

Nombre	Endpoint	Scope	URI Params	Body	Descripción
findAll	<LAMBDA_URI> /layouts	GET			Recuperar todos los layouts
findById	<LAMBDA_URI> /layouts/{id}	GET	`MANGO 002`		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": "Area1",
				            "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	MANGO002		Borra un layout dado su identificador único (código de tienda)

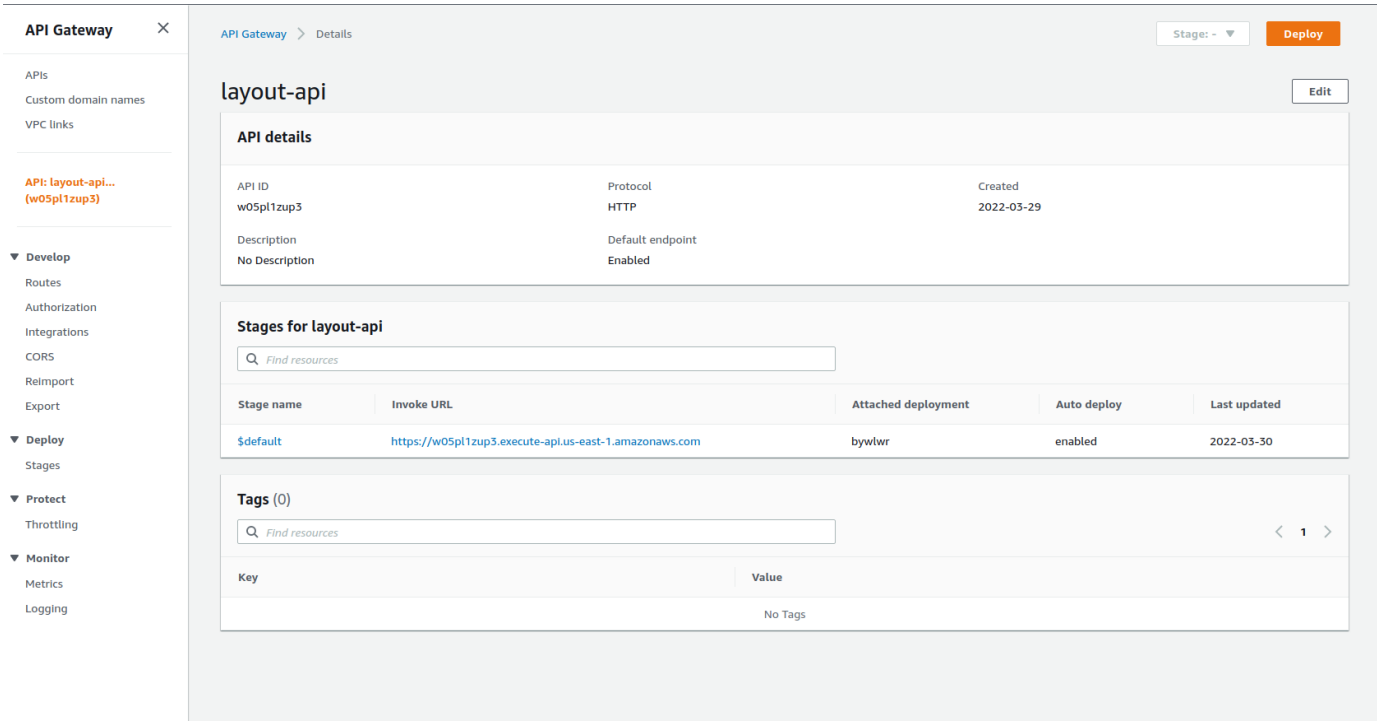


STEP 04: Creamos un API

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 **rutas** para ser configurar posteriormente



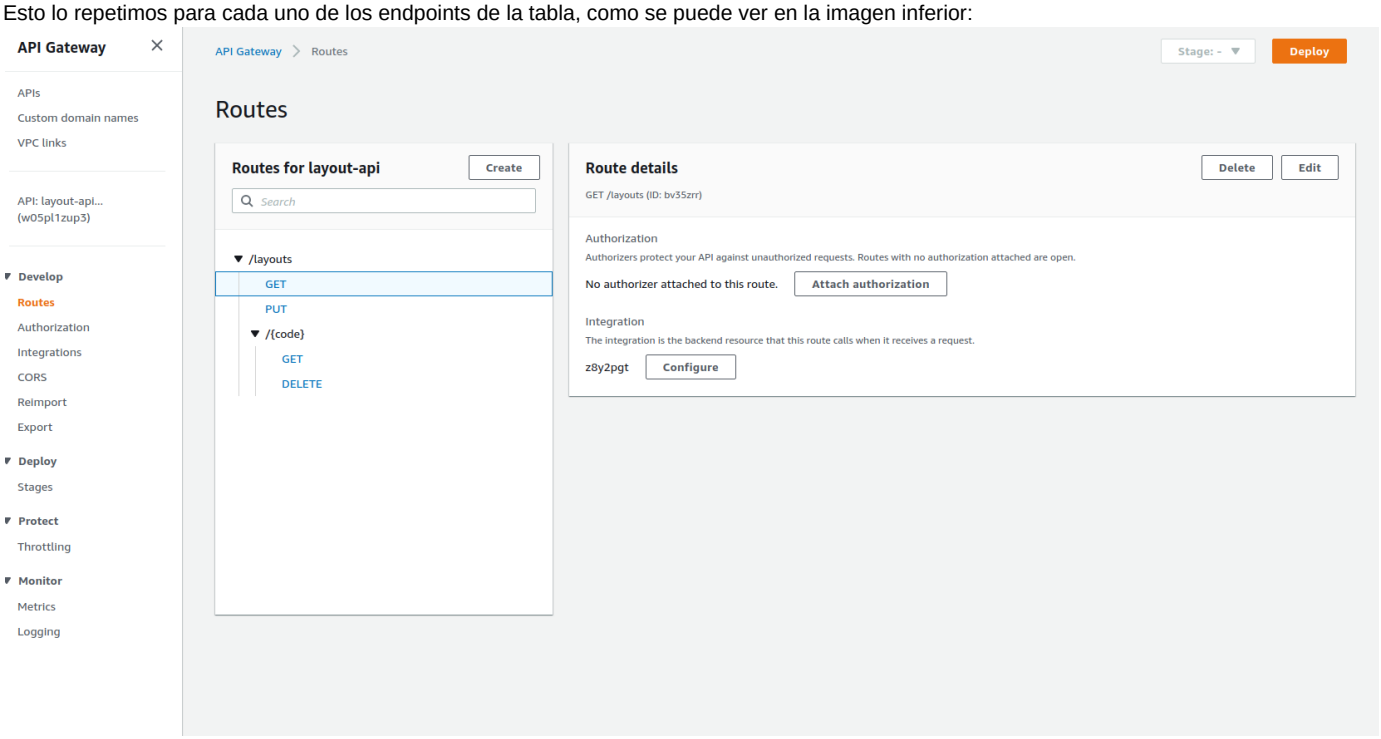
STEP 05: Creamos las rutas del API

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

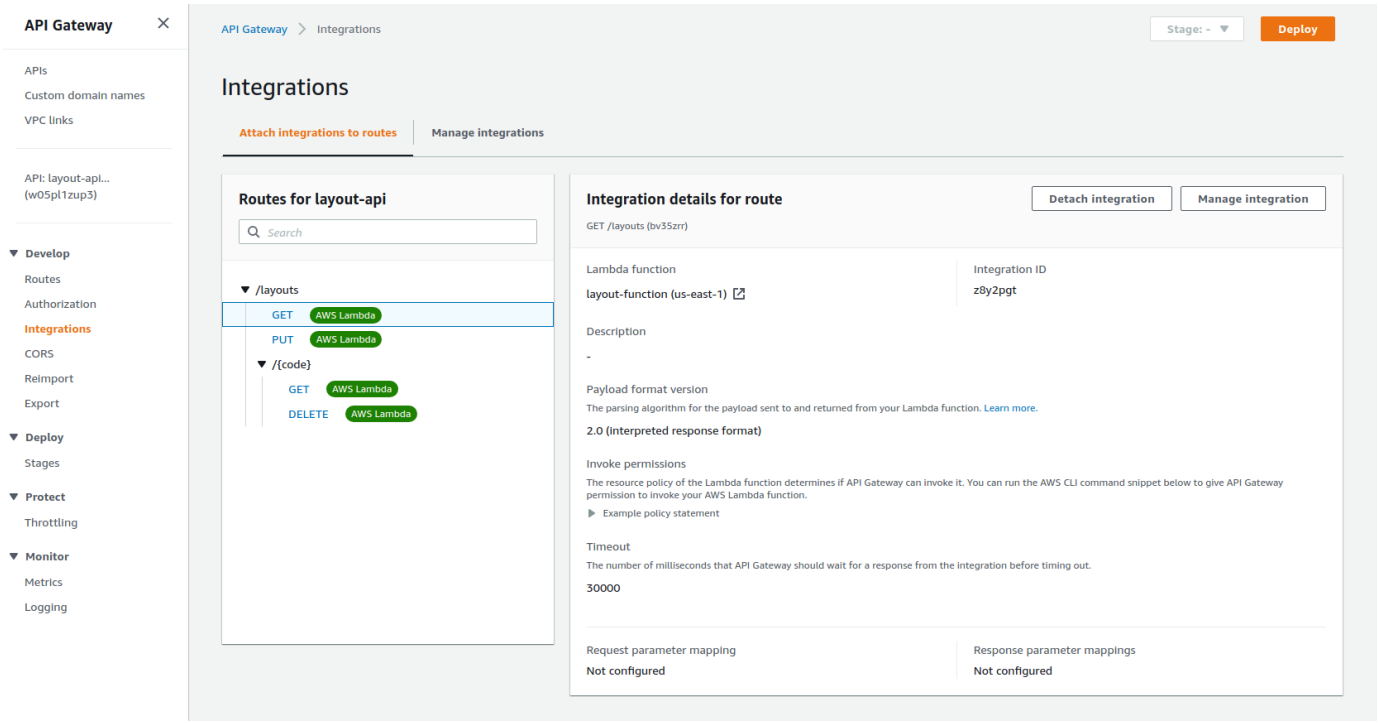
Ahora que ya tenemos nuestra función lambda implementado el CRUD, vamos a crear una integración que utilice esta función lambda.

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

STEP 07: Conectamos la integración a las rutas

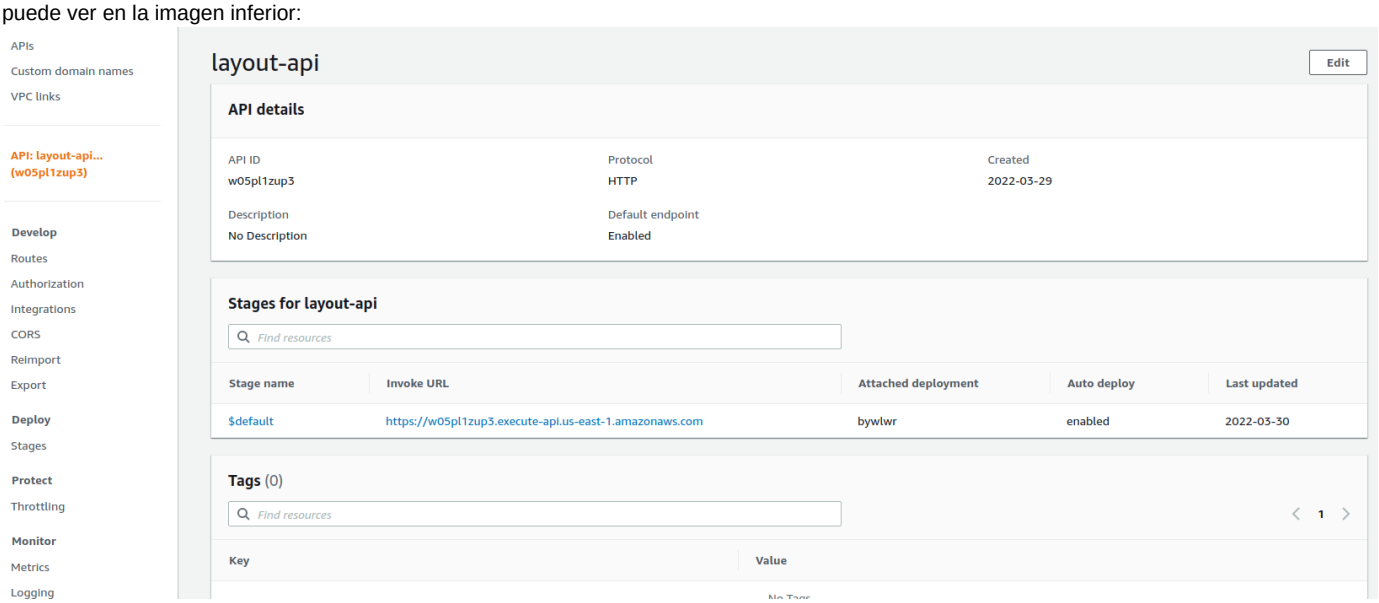
Ahora con la integración creada la conectamos a cada una de las rutas antes creadas:

- 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:



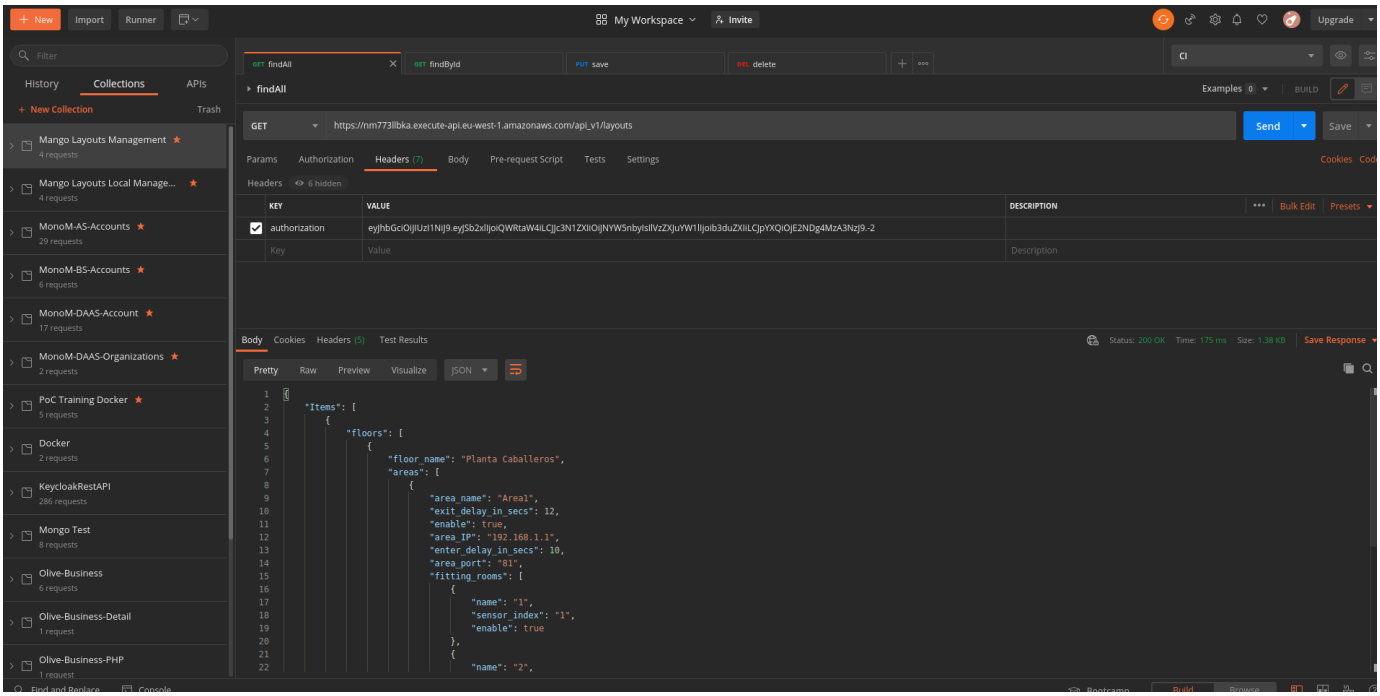
STEP 08: Probar la API

Ahora que ya tenemos implementada la integración con DynamoDB con una función lambda en Node.js y creadas las rutas con todos los metódos 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 recien creado bajo el título **Invoke URL,** como se puede ver en la imagen inferior:

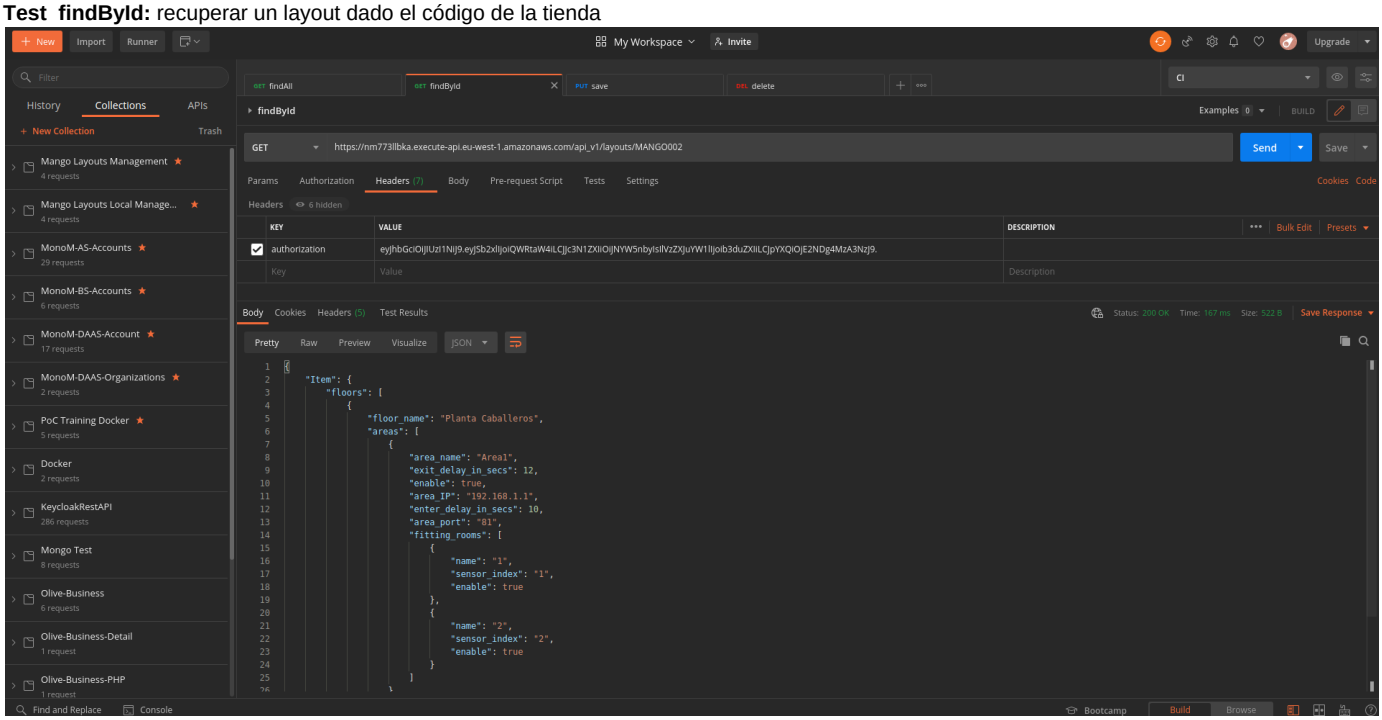


Usaremos Postman para probarlo e implementaremos todos los métodos ofrecidos por el API HTTP configurado previamente:

Test findAll: recuperar todas las tiendas configuradas

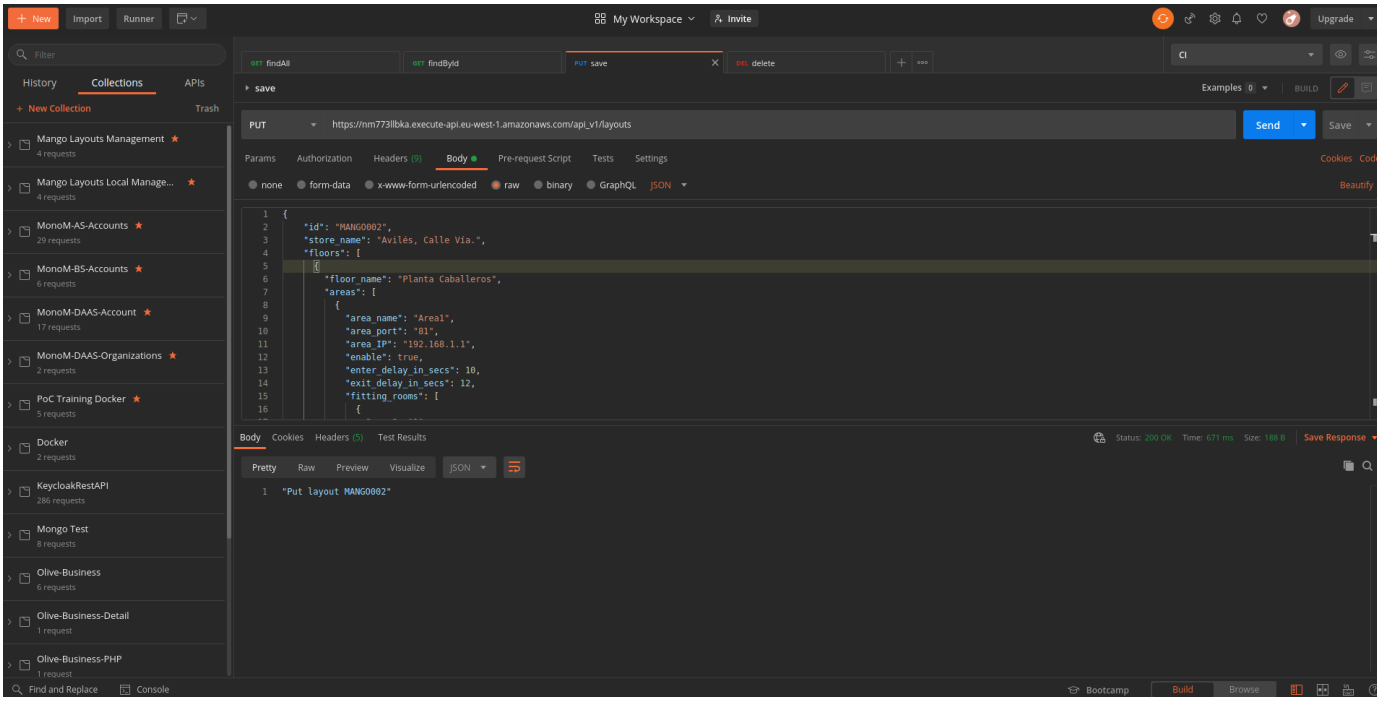


Test findById: recuperar un layout dado el código de la tienda

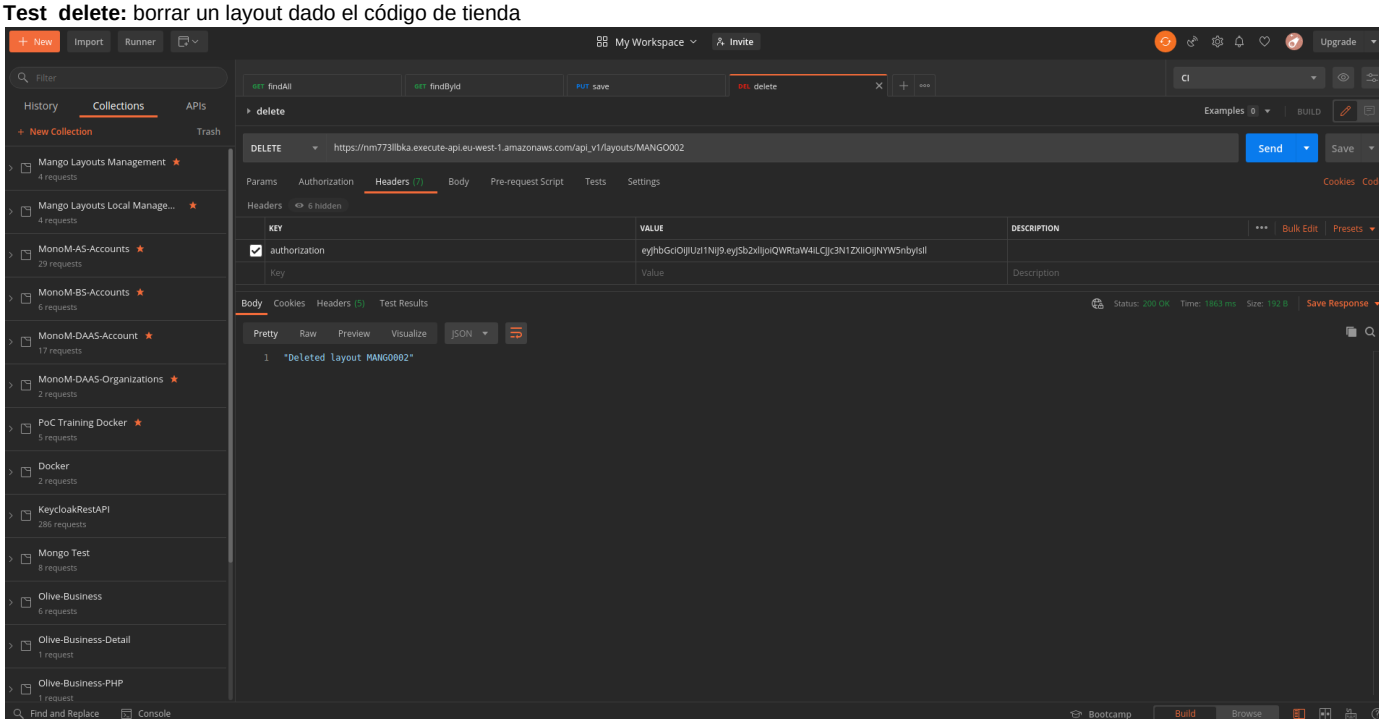


Test save: crea o actualizar un layout dato un objecto tienda .Este objecto debe contener:

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