JORGE AGUILERA

API-First con Micronaut

API-First con Micronaut

Desarrollo orientado al Contrato

Jorge Aguilera

Este libro está a la venta en https://leanpub.com/api-first-micronaut

Esta versión se publicó en 2026-08-18

A ella, que me vio nacer, esté donde quiera estar

Aprende Micronaut desde cero

Índice general

Bienvenido

Bienvenido a esta guía práctica de **Api-First** con **Micronaut**. El objetivo de este libro es ir más allá de la teoría del diseño de APIs y construir, paso a paso, una aplicación simple pero funcional utilizando el ecosistema de Micronaut.

> Para aprovechar al máximo este libro no necesitas ser un experto en Micronaut ni en OpenAPI, pero sí se asume cierta familiaridad con el ecosistema Java/JVM, la construcción de APIs REST y el uso de herramientas de línea de comandos así como algún IDE.

Requisitos previos

El principal requisito para poder seguir las explicaciones y ejecutar los ejemplos que se proporcionan es tener un JDK instalado. En este libro usaré Java 25 por ser una versión moderna, pero versiones anteriores son también perfectamente válidas.

La idea es que el código a utilizar sea sencillo y no utilice características propias de una versión específica de Java, aunque un mínimo de conocimiento de Java es necesario.

Así mismo usaré Linux como sistema operativo de referencia para aquellos comandos que haya que ejecutar en la terminal, como crear directorios o ejecutar un programa, pero tanto las explicaciones como ejemplos son igualmente válidos en Windows como MacOS. Simplemente, tendrás que usar el correspondiente en tu sistema.

Ejemplos

Todo el código fuente del libro es 100% funcional y está alojado en GitHub. Para evitar problemas con el historial de Git o la complejidad de gestionar ramas interactivas, el repositorio utiliza una estructura basada en directorios independientes.

Cada capítulo relevante o fase del proyecto corresponde a una carpeta raíz en el repositorio:

```
petstore-apifirst
|   README.md
|
└──01-petstore-codefirst
|   |   README.md
|   |   build.gradle
|   |
|   └──src
|       |   ...
|       |   Application.java
|
└──02-petstore-codefirst
|   |   README.md
|   |   build.gradle
|   |
|   └──src
|       |   ...
|       |   Application.java
```

Si dispones de **git** puedes clonar todos los ejemplos directamente ejecutando en un directorio de tu máquina:

```
git clone git@github.com:jagedn/petstore-apifirst.git
```

O bien bajarte con un navegador el zip y descomprimirlo en una carpeta de tu elección.

Convenciones del libro

A lo largo de los capítulos encontrarás diferentes tipos de contenido:

- Bloques de código: Mostrarán fragmentos de Java, especificaciones OpenAPI en YAML o configuraciones Gradle por ejemplo.
- Comandos de terminal: Indicados con el prefijo `$`. Puedes copiarlos y ejecutarlos directamente en tu consola (quitando el prefijo `$`)
- Llamadas a endpoints: Ejemplos con curl preparados para copiar y pegar contra la aplicación en ejecución local.

ApiFirst

El concepto de API, o application programming interfaces, tiene mucha historia ya en el mundo de la computación moderna.

Su funcion principal es permitir que diferentes piezas de software se comuniquen, incluso siendo servicios corriendo en máquinas remotas, estando hoy en día presentes en prácticamente todos los proyectos.

Podemos decir, de forma muy simplificada, que existen dos formas de desarrollar una API: definiéndolo al "principio" (Api-first) del proyecto o al "final" (Code-first)

Code-first, plantea que el API será generado a partir del propio código, bien mediante anotación o por inferencia del mismo, permitiendo al programador centrarse en el negocio y generar la especificación del API a partir del mismo.

Por su parte, la metodología API-first prioriza el diseño del API desde el inicio del proyecto, incluso antes de escribir ninguna línea de código. Esto permite a los equipos construir aplicaciones pensando desde el principio en la interconectividad

Podemos decir que con API-first el equipo multidisciplinar (backend, frontend, product…) comienza diseñando, discutiendo y formalizando la especificación de la API —habitualmente usando la especificación OpenAPI 3.x— antes de escribir una sola línea de lógica de negocio o infraestructura.

Pilares

Los 3 pilares en los que se basa API-first son:

- La API es el contrato de la verdad, es decir, la única fuente oficial de lo que el sistema puede hacer (y lo que no).
- Diseño colaborativo. Promueve que un equipo multidisciplinar participe en esta definición, puesto que todos son partes interesadas, a diferencia de Code-first donde usualmente es el backend el que define el contrato y los consumidores deben adaptarse
- Desacoplamiento del código, donde el framework/servidor se ajustan a la especificacion y no al reves

API-first NO implica que la API diseñada sea inmutable y no se puede cambiar. De hecho es una buena práctica que el equipo multidisciplinar tenga reuniones de revision frecuentes sobre todo al principio cuando se está en fase de descubrimiento.

Esto permite que todas las partes implicadas en el desarrollo (tanto productores como consumidores) puedan avanzar sin "esperar" a que la otra parte tenga completada y desplegada la suya como suele ocurrir con CodeFirst

Herramientas

Aunque en un plano agnóstico API-first es independiente de las herramientas, en la práctica es recomendable que el framework elegido las proporcione de una forma integrada y de calidad.

Hoy en día existen multitud de herramientas que a partir de la especificación generan MockServers, por ejemplo, de tal forma que el consumidor puede ir avanzando su parte contra este y reemplazarlo con la versión generada por el productor cuando esté lista

De la misma forma existen herramientas que a partir de las especificaciones pueden generar diferentes clases de tests (unitarios, de integración, ...) para que el productor pueda asegurar que su proyecto cumple con las especificaciones acordadas.

En este libro nos vamos a centrar en la parte de productor, y usaremos cualquier IDE capaz de interpretar proyectos Java con Gradle como herramienta de desarrollo.

Micronaut

Durante más de dos décadas, los frameworks en el ecosistema Java han confiado fuertemente en la reflexión en tiempo de ejecución (runtime reflection), la carga dinámica de clases y la generación de proxies en memoria para implementar Inyección de Dependencias (DI) y Programación Orientada a Aspectos (AOP) siendo Spring el framework de referencia en el desarrollo de aplicaciones.

Esta arquitectura funcionó bien durante la era de los servidores de aplicaciones monolíticos que funcionaban de forma ininterrumpida. Sin embargo, en el paradigma actual de contenedores (Docker/Kubernetes), microservicios y Serverless, este modelo tradicional presenta dos problemas graves: tiempos de arranque elevados y un alto consumo de memoria RAM.

En 2018 Graeme Rocher, el principal desarrollador de Grails (un framework basado en Spring y Groovy), presentó en el Greach de Madrid un proyecto en el que había empezado a trabajar Youtube

En esta presentación Graeme mostró las bases de lo que sería Micronaut

¿Qué es Micronaut?

Micronaut es un framework JVM moderno y modular diseñado específicamente para la construcción de microservicios y aplicaciones cloud-native. Fue creado por el equipo detrás de Grails (Object Computing, Inc.) con una premisa fundamental: hacer en **tiempo de compilación** todo lo que tradicionalmente se hacía en tiempo de ejecución.

Características

- **Compilación Ahead-Of-Time** (AOT): Micronaut utiliza procesadores de anotaciones (JSR-269 en Java, KSP en Kotlin o AST Transformations en Groovy) para calcular el grafo de dependencias, validar anotaciones y generar los metadatos necesarios durante la fase de compilación del proyecto
- **Cero reflexión en runtime**: Al no depender de la reflexión para la inyección de dependencias, el tiempo de arranque de la aplicación es prácticamente independiente de la cantidad de beans o controladores que tenga el proyecto

- **Consumo mínimo de memoria**: La eliminación de caches de reflexión internas permite que la aplicación arranque consumiendo una fracción de la memoria RAM requerida por otros frameworks tradicionales.

Maven o Gradle

Micronaut es compatible con Maven y Gradle, simplemente al crear el proyecto decides cúal quieres usar.

Personalmente, prefiero Gradle y es lo que usaremos en el libro, pero si quieres usar Maven simplemente debes indicárselo al comando `mn create-app` con la opción `--build maven`

Java, Kotlin o Groovy

Micronaut, a diferencia de otros frameworks JVM como Quarkus por ejemplo, puede trabajar con Java, Kotlin o Groovy sin problemas.

Yo soy un fan de Groovy, pero en los ejemplos del libro usaremos Java por ser el lenguaje más común en el ecosistema JVM.

Versiones

Al tiempo de escribir este libro Micronaut se encuentra en su versión 5.0.x y es la que usaremos, aunque versiones anteriores son también perfectamente válidas.

Si bien Micronaut demuestra ser un framework muy vivo, con releases frecuentes, la transición entre ellas es muy cuidada y fácil.

> En muchos casos basta con cambiar el número de versión y todo el proyecto sigue siendo válido. En mi experiencia personal la mayor dificultad se presentó al pasar de las versiones 2.x a 3.x, pero no he encontrado mayores problemas al pasar de 3.x a 4.x o recientemente de 4.x a 5.x, aunque siempre dependerá de las dependencias y características de tu propio proyecto

Singleton e Inject

A lo largo del libro iremos usando diferentes funcionalidades de Micronaut, casi todas implementadas por el framework mediante anotaciones, pero antes de irlas descubriendo nos pararemos un segundo a explicar las dos que más se usan a lo largo del libro.

Si has trabajado con Spring las anotaciones **@Bean** , **@Service**, **@Prototype**, etc. te sonarán familiares.

Con estas anotaciones indicamos al framework que estas clases son especiales y que están destinadas a ser inyectadas en otras clases, bien mediante métodos *setXXX* o, preferiblemente, en el constructor de las clases dependientes de ellas.

De forma similar Micronaut no solo proporciona las suyas propias, sino que utiliza las definidas en los JSR como por ejemplo **@Singleton**

Con esta anotación indicamos al framework que nuestra aplicación requiere de una sóla instancia de esta clase, la cual deberá ser proporcionada en aquellos puntos en las que se indique.

Un ejemplo sencillo:

```
@Singleton
public class TestService{

    void sayHello(){
        System.out.println("Hello");
    }

}

@Singleton
public class UseCase1{

   @Inject
   TestService testService;

   void doIt(){
    testService.sayHello();
   }
}

@Singleton
public class UseCase2{

   private final TestService testService;
   public UseCase2(TestService testService){
```

```
26      this.testService = testService;
27     }
28
29     void doIt(){
30      testService.sayHello();
31     }
32 }
```

En este ejemplo vemos cómo le indicamos a Micronaut que nuestra aplicación consta de tres clases y que queremos una sóla de cada una (al anotarlas como @Singleton)

Además, le decimos a Micronaut que en tiempo de compilación, antes de crear *UseCase1* y *UseCase2* tiene que crear *TestService*.

Así mismo una vez que se haya creado *UseCase1* tiene que asignarle a la variable *testService* la instancia creada anteriormente (al anotarla con **@Inject**).

De la misma forma, cuando se vaya a llamar al constructor de *UseCase2* se tiene que proporcionar la misma instancia de *TestService* que se ha usado en *UseCase1*

A lo largo del libro veremos alguna otra anotación, como **@Controller**, **@Transactional** pero son más fáciles de entender y casi autoexplicativas

El enfoque de Micronaut, **a diferencia del de Spring**, es que todo esto puede hacerlo en **tiempo de compilación**, es decir, Micronaut genera e inyecta código que resuelven todos estos detalles y los incorpora a nuestro "jar" como si los hubiéramos escrito nosotros mismos. Por el contrario, Spring utiliza las anotaciones para, en **tiempo de ejecución** descubrir estos detalles y resolver las dependencias entre los Singletons

Quarkus

Cuando uno habla de Micronaut es muy normal que alguien pregunte por Quarkus.

Quarkus es un framework que comparte muchas características e incluso historia con Micronaut, ya que fue creado casi al mismo tiempo y con una visión parecida.

La diferencia principal radica en la filosofía: Quarkus adopta el estándar MicroProfile/Jakarta EE y está muy optimizado para el ecosistema Kubernetes/Red Hat, mientras que Micronaut mantiene una sintaxis extremadamente cercana a Spring, facilitando la transición a desarrolladores acostumbrados al paradigma de Spring.

Aunque no lo he usado tanto como me gustaría Quarkus es otro framework muy maduro y con funcionalidades muy interesantes. Particularmente no lo he usado tanto porque no se puede usar Groovy, lenguaje que me apasiona, pero por lo demás es un framework tan bueno, en mi experiencia, como Micronaut.

ApiFirst con Micronaut

En una estrategia API-First, la especificación OpenAPI actúa como la fuente de verdad. Para llevar esto a producción de forma eficiente, necesitamos que la traducción del contrato OpenAPI a interfaces y controladores de Java no suponga una penalización de rendimiento en runtime.

El procesador de anotaciones de Micronaut no solo genera el servidor HTTP, sino que analiza la especificación OpenAPI durante la compilación para:

- Validar la consistencia de los tipos de datos antes de ejecutar la aplicación.
- Generar clases DTO y clientes HTTP sin añadir metadatos en runtime.
- Proporcionar documentación Swagger/OpenAPI estática.

En los próximos capítulos veremos la diferencia en la práctica, comparando la implementación directa basada en código con la generación automática mediante contratos.

Petstore Lite

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Persistencia

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Entity

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Repository

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Inicio

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Java y Sdkman

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Micronaut

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

01-petstore-codefirst

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Code First

Una vez validado que contamos con el entorno de trabajo adecuado, y que la aplicación arranca correctamente, pasaremos a crear nuestra primera versión de Petstore Lite usando CodeFirst

El código de este capítulo se encuentra en la carpeta **02-petstore-codefirst** del repositorio de github

Partiendo del proyecto creado anteriormente, y como queremos proporcionar al equipo de FrontEnd una API REST válida cuanto antes, vamos a trabajar lo primero en la capa http definiendo los Controllers y los DTOs de entrada y salida.

Como el Controller creado automáticamente por Micronaut no nos interesa podemos/debemos eliminarlo.

DTOs

Creamos un DTO en el package **dtos** para representar el Pet que queremos crear.

```
1  package com.incsteps.dtos;
2
3  import io.micronaut.serde.annotation.Serdeable;
4  import jakarta.validation.constraints.NotNull;
5
6  @Serdeable
7  public record CreatePetRequest(
8          @NotNull String name,
9          int age
10 ) {
11 }
```

La anotación `@Serdeable` nos permite serializar y deserializar el DTO utilizando el formato JSON facilitando su uso en las llamadas HTTP.

Así mismo, crearemos un DTO para representar el Pet creado.

```java
package com.incsteps.dtos;

import io.micronaut.serde.annotation.Serdeable;

@Serdeable
public record Pet(
        Long id,
        String name,
        int age
) {
}
```

Como podemos ver, el DTO de salida es muy similar al DTO de entrada, incluyendo el ID del Pet creado.

Con estos DTOs ya podemos crear el Controller para crear un nuevo Pet.

```java
package com.incsteps.http;

import com.incsteps.dtos.CreatePetRequest;
import com.incsteps.dtos.Pet;
// resto de imports eliminados por brevedad

@Controller("/pets")
@Tag(name = "Pets", description = "Operaciones sobre el catálogo de mascotas")
public class PetstoreController {

}
```

Get Pet by ID

Por ahora vamos a definir solo el interface http para obtener un Pet por su ID pero devolviendo siempre un 404.

```
    @Get("/{id}")
    @Produces(MediaType.APPLICATION_JSON)
    @Operation(
            summary = "Obtener mascota por ID",
            description = "Devuelve los detalles de una mascota concreta dada su clave primaria."
    )
    @ApiResponse(
            responseCode = "200",
            description = "Mascota encontrada",
            content = @Content(schema = @Schema(implementation = Pet.class))
    )
    @ApiResponse(
            responseCode = "400",
            description = "ID suministrado no válido",
            content = @Content
    )
    @ApiResponse(
            responseCode = "404",
            description = "Mascota no encontrada",
            content = @Content
    )
    public HttpResponse<Pet> get(
            @Parameter(description = "Identificador único de la mascota", required = true)
            @PathVariable("id")
            @NotNull
            @Positive Long id
    ) {
        return HttpResponse.notFound();
    }
```

Post Pet

```
    @Post
    @Produces(MediaType.APPLICATION_JSON)
    @Operation(
            summary = "Registrar una nueva mascota",
            description = "Crea una nueva mascota en el catálogo con los datos proporcionados."
    )
    @ApiResponse(
            responseCode = "201",
            description = "Mascota creada con éxito",
            content = @Content(schema = @Schema(implementation = Pet.class))
    )
    @ApiResponse(
```

```
            responseCode = "400",
            description = "Petición no válida (datos faltantes o erróneos)",
            content = @Content
    )
    public HttpResponse<Pet> create(
            @Parameter(description = "Datos de la mascota a crear", required = true)
            @Body @NotNull @Valid CreatePetRequest request
    ) {
        return HttpResponse.badRequest();
    }
```

Al igual que en el caso de obtener un Pet por su ID, por ahora vamos a devolver un bad request.

Delete Pet

De la misma forma que en el caso de obtener un Pet por su ID o crear un nuevo Pet, definiremos un método DELETE para eliminar un Pet.

Puedes ver el método completo en el archivo **PetstoreController.java** en la carpeta **02-petstore-codefirst** del repositorio de github.

Desplegando

Una vez que tenemos definido los endpoints (sin lógica de negocio implementada) procederíamos a desplegarlo. Por ahora nos bastará con ejecutarlo en local y comprobar que Micronaut genere correctamente el contrato OpenAPI a partir de los DTOs y los endpoints definidos.

```
$ ./gradlew run
```

En un navegador abrimos la URL http://localhost:8080/swagger/petstore-0.0.yml y veremos el contrato OpenAPI generado.

```yaml
openapi: 3.0.1
info:
  title: petstore
  version: "0.0"
tags:
- name: Pets
  description: Operaciones sobre el catálogo de mascotas
paths:
  /pets:
    post:
      tags:
      - Pets
      summary: Registrar una nueva mascota
      description: Crea una nueva mascota en el catálogo con los datos proporcionados.
      operationId: create
      requestBody:
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/CreatePetRequest"
 (continua)
```

Iterando Code First

Como puedes ver, **Micronaut ha generado a partir del código un recurso OpenAPI en formato YAML que representa el contrato de la API**.

Esto nos permite documentar la API y generar clientes de cliente para consumirla. Es decir, una vez desplegada la aplicación podremos crear un MockServer para simular la API en nuestro entorno de desarrollo mientras desarrollamos la funcionalidad de negocio por separado.

UseCases

En el repositorio de ejemplo encontrarás una de las tantas posibles implementaciones. Unas implementarían arquitectura hexagonal, otras una arquitectura simple Controller-Service-Repository.

Cúal elegir (y cómo testearlas) queda fuera del alcance de este libro. Por no complicar los ejemplos y tener un proyecto completo, he optado por crear unos casos de uso como SingletonS, que puedes encontrar en la carpeta **02-petstore-codefirst/src/main/java/com/incsteps/usecases** los cuales usa el Controller.

OpenAPI

Hasta ahora hemos delegado en Micronaut la tarea de generar el OpenAPI a partir de nuestras anotaciones (Code-First). Pero en Api-First somos nosotros quienes definimos la API para que, por un lado, el framework genere el código y por otro, los clientes puedan consumirlo.

OpenAPI es el estándar abierto de facto en la industria para describir APIs RESTful de forma independiente del lenguaje de programación. Mantenido por la OpenAPI Initiative (bajo el paraguas de la Linux Foundation), permite definir endpoints, formatos de entrada/salida, mecanismos de autenticación y esquemas de datos en un documento estructurado en formato YAML o JSON.

En este capítulo se presenta el formato OpenAPI brevemente, pero puedes consultar toda la especificación en OpenAPI

Estructura básica

A continuación se presenta un extracto de cómo sería un OpenAPi en formato YAML:

```yaml
openapi: 3.0.3
info:
  title: Petstore Lite API
  version: 1.0.0
  description: Contrato oficial para el servicio de gestión de mascotas.

servers:
  - url: http://localhost:8080/api/v1
    description: Servidor local de desarrollo

paths:
  # Definición de rutas y operaciones HTTP
  /pets/{id}:
    get:
      summary: Obtener mascota por ID
      operationId: getPetById
      parameters:
        - name: id
```

```yaml
          in: path
          required: true
          schema:
            type: integer
            format: int64
      responses:
        '200':
          description: Mascota encontrada
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Pet'
      ...

components:
  # Elementos reutilizables (Schemas, Respuestas, Parámetros)
  schemas:
    Pet:
      type: object
      ...
```

En el ejemplo anterior se ha utilizado el formato YAML, escrito a "mano" pero existen editores o plugins que facilitan la definición del OpenAPI de forma visual. Por ejemplo Swagger Editor Online es el editor online de referencia.

- openapi: Especifica la versión del estándar utilizada. La serie 3.0.x y 3.1.x son las normas actuales en producción.
- info: Metadatos de la API (título, versión semántica, términos de servicio, contacto, licencia).
- servers: Array con las URLs base donde la API está desplegada o estará disponible.
- paths: La espina dorsal del documento. Define los endpoints disponibles, los métodos HTTP admitidos (get, post, delete, etc.), parámetros de entrada y respuestas esperadas.
- components: El almacén de componentes reutilizables. Aquí se definen los Schemas (los DTOs), respuestas comunes o esquemas de seguridad para evitar duplicar código YAML.

Paths

En **paths** asociamos rutas HTTP con parámetros de entradas y las respuestas esperadas

En cada **path** indicamos el verbo HTTP (get, post, delete, etc.) así como los parámetros que pueden venir en la ruta y/o en el body

Así mismo definimos las respuestas esperadas por el endpoint (cuantas más definamos mejor explicado quedará la API)

El campo **operationId** es fundamental al trabajar con herramientas de generación de código como las de Micronaut. Determinará directamente el nombre del método Java que se creará en la interfaz resultante, lo cual nos ayudará en la implementación.

Schemas

Mediante los **schemas** definimos la estructura de los DTOS que se van a utilizar en la API.

En Api-First definiremos los campos y formatos que se van a utilizar en la API pensando en la integración con los clientes, no en la implementación. Es decir, prestaremos cuidado en no definir campos que serían propios de la capa de persistencia por ejemplo

Micronaut (y otras herramientas orientadas a la generación del código) usarán los **schemas** para generar las clases Java que corresponden junto con los métodos de acceso a sus propiedades.

En esta fase podremos indicar si un campo es obligatorio o no, su tipo y su formato, si es un enumerado, etc. Por ejemplo:

```yaml
components:
  schemas:
    Pet:
      type: object
      required:
        - id
        - name
      properties:
        id:
          type: integer
          format: int64
          example: 1
        name:
          type: string
          example: "Tiramisu"
        age:
```

```
17            type: integer
18            example: 10
19          status:
20            type: string
21            enum: [AVAILABLE, ADOPTED]
```

Atributos como required, enum, minimum o pattern son leídos por Micronaut durante la fase de compilación para aplicar automáticamente las anotaciones de Jakarta Validation (@NotNull, @Pattern, @Min) sobre las clases Java generadas.

Api First con Micronaut

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Crear proyecto

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Plugin

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Construir el contrato

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Build

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Run

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Swagger-UI

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Implementando Api First

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Iteración 1

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Model

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Enum

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Validaciones

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Controller

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Implementación

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Iteración 2

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Testing con API-First

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Estrategias de testing

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Tests de contrato

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Tests de integración

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Testing con Micronaut

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Protección y seguridad

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Micronaut Security

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Security Schema y Micronaut

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Seguridad global

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Validaciones de seguridad

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Swagger UI

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Tests

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

OpenAPI Client

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

PokeApi

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Micronaut Client

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Download PokeApi Spec

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Build

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Client config

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Consumiendo el cliente

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

HttpClient

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Test

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Conclusion

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Micronaut

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Buenas prácticas

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

El ecosistema Contract-First

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Otras APIs

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Referencias

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

Agradecimientos

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

A mi entorno

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

A la comunidad Micronaut

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

A los revisores

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.

A tí

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en https://leanpub.com/api-first-micronaut.