



**mercado  
libre**

# **MELI CHALLENGE**

**GUÍA DE INSTALACIÓN Y DOCUMENTACIÓN**

Autor: Osneider Manuel Acevedo Naranjo

## RESUMEN

El proyecto lo desarrolle bajo la creación de un servicio de backend para entregar el detalle de un producto, siguiendo un enfoque de **microservicios** con principios de **resiliencia, escalabilidad, disponibilidad, monitoreo y seguridad**.

El flujo de trabajo se dividió en varias etapas clave:

### 1. Análisis y Diseño:

- Se investigó el detalle de un producto en la página oficial de Mercado Libre para definir el **modelo de datos** de la API (le pase un link con el detalle de un producto a Gemini para que interpretara que datos necesito para el detalle de un producto).
- Se diseñó una **arquitectura de microservicios** completa, utilizando conceptos como **ApiGateway**, balanceo de carga, Circuit Breaker y registro de servicios (**registry-server**).
- Se implementó un servicio de seguridad con **Auth-server** (OAuth 2.0) y un **config-server** para la configuración centralizada.
- Decidí usar **Java con Spring Boot y Spring Cloud** para agilizar el desarrollo y la implementación de estos componentes.

### 2. Tecnología y Herramientas:

- **Stack Principal:** Java (17), Spring Boot, Spring Cloud.
- **IDE de Desarrollo:** Se utilizó **Visual Studio Code** para la construcción del proyecto.
- **Componentes:** Los componentes desarrollados son:
  - **gateway:** se encargará de centralizar los consumo a los microservicios, garantizando la seguridad Auth2.0, la escalabilidad mediante el consumo por balanceador y consumos de contingencia en caso de que el servicios principal de consulta de detalles de productos falle.
  - **auth-server:** se encargará de aprovisionar la autenticación mediante Auth2.0.
  - **config-server:** Se encargará de aprovisionar los parámetros de conexiones, credenciales a base de datos y parámetros secretos para la generación de JWT en auth2.0.
  - **registry-server:** este será nuestro Servidor de Registro y Descubrimiento de Servicios (balanceador).
  - **product-detail:** este será el servicio de consulta de detalles de productos (servicio principal)
  - **product-detail-fallback:** este será el servicio de consulta de detalles de productos (servicio de respaldo o contingencia)
- **Bases de Datos:** Se utilizó **H2** como base de datos en memoria. La construcción de los modelos relacionales y el precargue inicial de datos desde un archivo CSV se gestionó con **Liquibase**.

- **Herramientas de IA:** Se utilizó **Gemini** para el análisis inicial y **GitHub Copilot** junto con **Claude Sonnet 4** para la construcción del código, optimizando el tiempo de desarrollo.

### 3. Implementación y Pruebas:

- Se crearon los microservicios mencionados, con un servicio principal (Product-Detail) y su respectivo fallback para asegurar la resiliencia del sistema.
- Se implementaron soluciones de **monitoreo** con **Actuator**, **Prometheus**, **Zipkin** y **Grafana** para garantizar la **trazabilidad y la observación** del sistema.
- El proyecto se configuró para una ejecución sencilla en un entorno local, listo para pruebas sin necesidad de configuraciones adicionales complejas.

### 4. Entrega y Documentación:

- Se entregó todo el código en un **repositorio de GitHub**.
- Se elaboró un **archivo README.md** para cada servicio, tal como lo solicitaba la prueba.
- Se preparó una **guía de instalación completa**, que incluyó un documento detallado y un **video tutorial**, explicando el proceso de montaje y pruebas.
- Los insumos se compartieron a través de GitHub, un archivo ZIP y un enlace de Google Drive, asegurando una entrega completa y accesible.

En resumen, la actividad demostró la capacidad de diseñar, desarrollar y documentar una solución de backend robusta y escalable, utilizando las mejores prácticas de la industria y herramientas de vanguardia para la eficiencia y calidad del resultado.

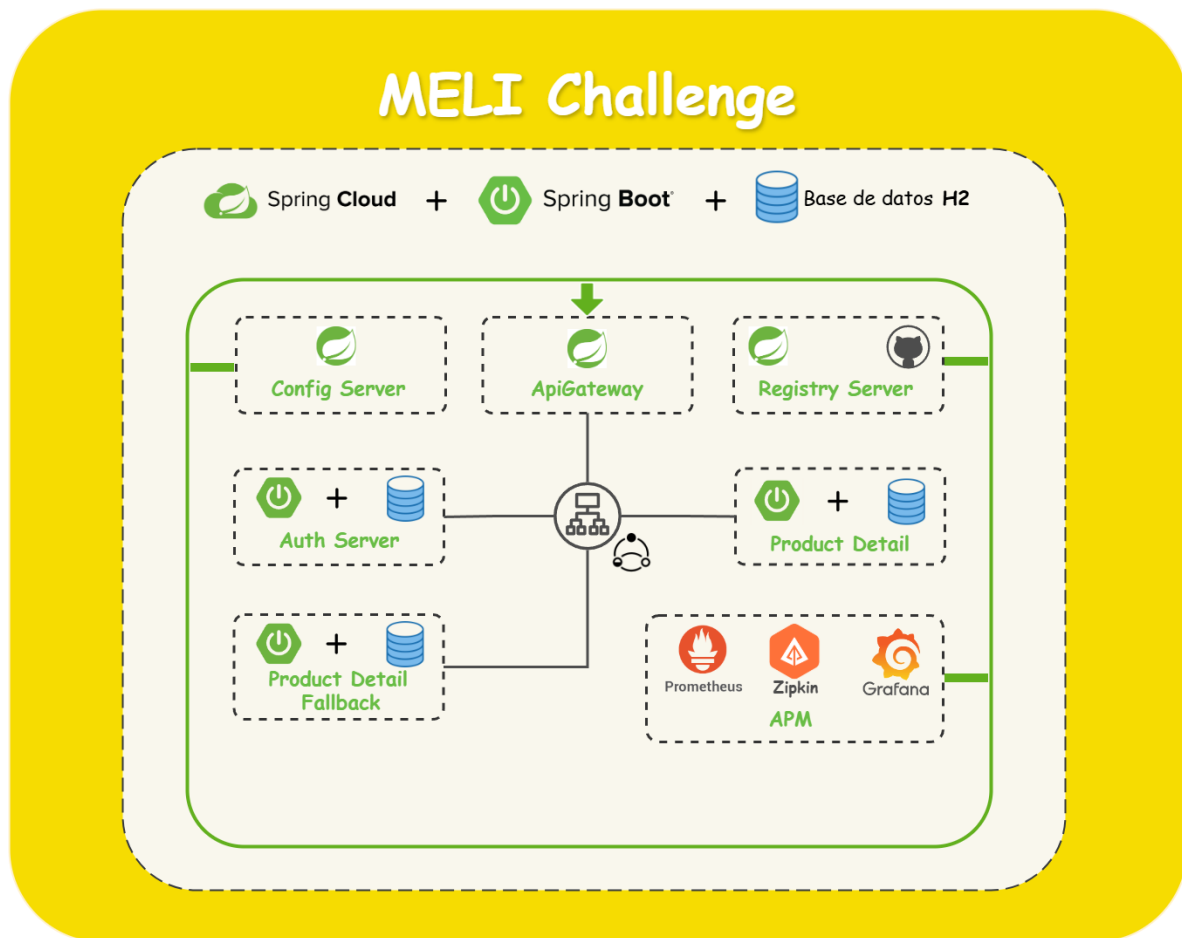
## PUNTOS CLAVE DE LA ENTREGA

Me voy a centrar en el microservicio ProductDetail que es el encargado de retornar el detalle de un producto y el centro de toda la prueba de MELI Challenge.

- 1- **Excepciones personalizadas:** Le implemente un sistema de excepciones personalizado llamado MeliException que nos permite estandarizar los errores para no entregar información del servidor a los clientes.
- 2- **Información relevante para soporte en caso de fallos:** En caso de fallos sobre el servicio el responde nos entrega un trace y un código sobre el mensaje de respuesta, esto nos permite identificar dentro del código que excepción fallo y a nivel de zipkin validar el detalle del error.

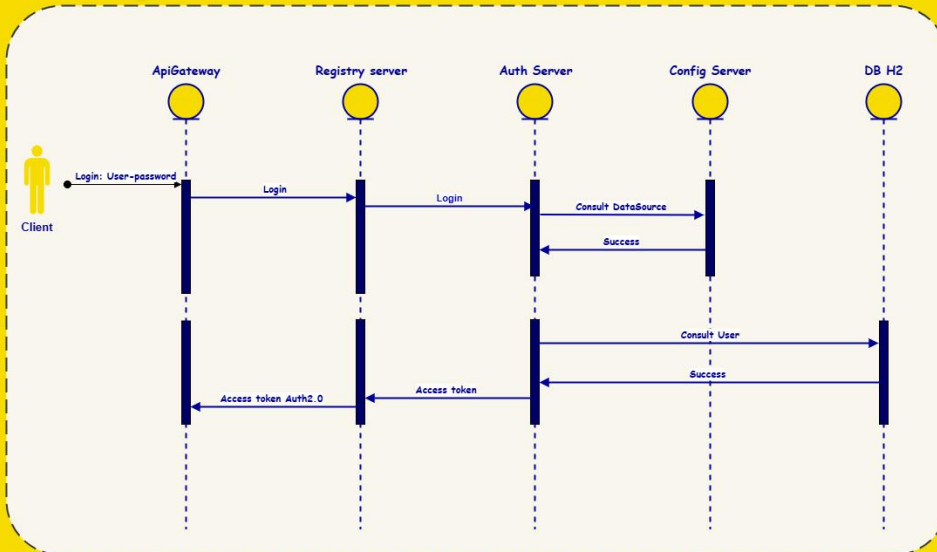
- 3- **Modelo relación compatible con cualquier base de datos:** al implementar liquibase este se encarga de generar y construir los modelos relacionales y no relaciones según la estructura de sus archivos de configuración, permitiendo generar compatibilidad sobre cualquier base de datos, adicionalmente permite generar un control de versiones sobre los cambios en DB.
- 4- **Cargue inicial de datos:** al implementar liquibase puedo generar un cargue inicial de datos desde archivos .sql, csv, etc.
- 5- **Generación de Swagger automatico:** Al implementar springdoc-openapi permito documentar en tiempo real todas las APIs que expone mi microservicio.
- 6- **Monitoreo de Endpoint:** Al implementar actuator y prometheus puedo generar métricas de consumo sobre los servicios, permitiendo monitoreo sobre fallos, tiempos de espera altos, consumos sobre endpoint no existentes, entre otros.
- 7- **Monitoreo Zipkin:** Al implementar Zipkin puedo determinar los tiempos de espera entre los consumos de las diferentes clases para poder determinar si existe tiempos de esperas prologados sobre un componente y poder así tomar acciones de optimización, así mismo me permite generar span para identificar el error sobre los try catch de la aplicación.
- 8- **Anotaciones personalizadas:** Al utilizar el concepto de programación orientado a aspecto, cree anotaciones para mejorar el aspecto visual del código, permitiendo no saturar a los desarrolladores con la documentación de Swagger.
- 9- **Patrones de arquitectura:**
  - Microservicio
  - MVC (Model-View-Controller)
  - Repository
  - Service
  - DTO
  - Exception Handling
- 10- **Principios SOLID y buenas prácticas:**
  - Single Responsibility
  - Open/Closed
  - Dependency Injection
  - Separation of Concerns
  - DRY (Don't Repeat Yourself)
  - Encapsulamiento

## DIAGRAMA DE CONTEXTO DE LA SOLUCION

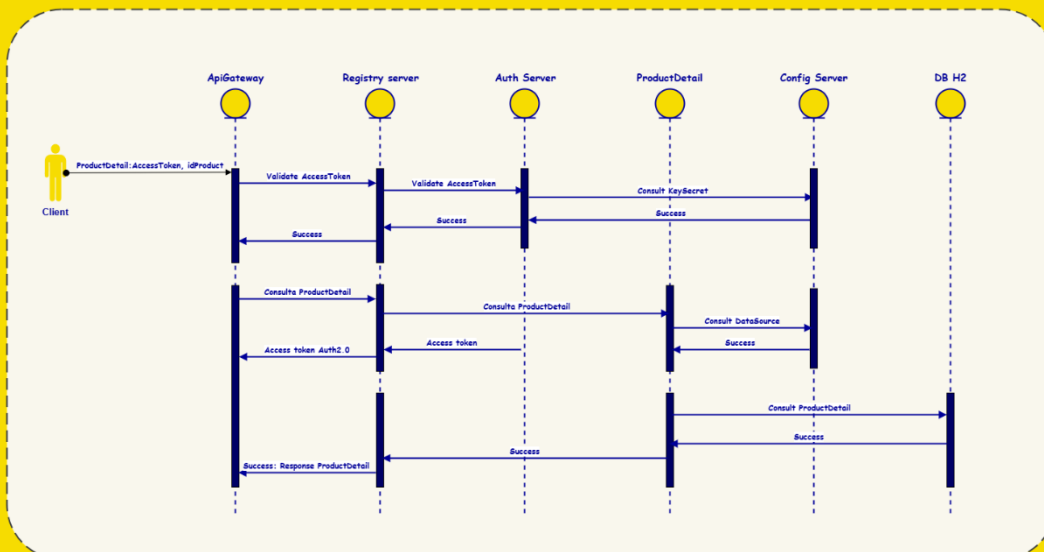


## DIAGRAMA DE SECUENCIA

### Flujo de Autenticación Auth2.0



### Flujo Consulta de detalle producto



## TABLA DE IPs

PROYECTO	HOST	PUERTO
registry-server	http://localhost	8761
config-server	http://localhost	7777
product-detail	http://localhost	8080
product-detail-fallback	http://localhost	8181
auth-server	http://localhost	3030
gateway	http://localhost	4040

## GUIA DE EJECUCIÓN

**Nota:** por favor, en lo posible revisar el video ya que la prueba la lleve a cabo bajo una arquitectura mas robusta y es complejo plasmas la configuración en un documento por que se aria muy extenso.

Video guía y demostración completa de la prueba:

**Todo el material, instaladores y proyectos:**

<https://drive.google.com/drive/folders/1ndne98VMTfcicorl9chrzgRKxe6UxY-Y?usp=sharing>

**Video duración:** 37min

**El archivo pesa debido a que el video esta en HD:** solo el video pesa 2.7GB

GitHub:

**Todo el proyecto:** <https://github.com/lko-developer-software/MELI-CHALLENGE#>

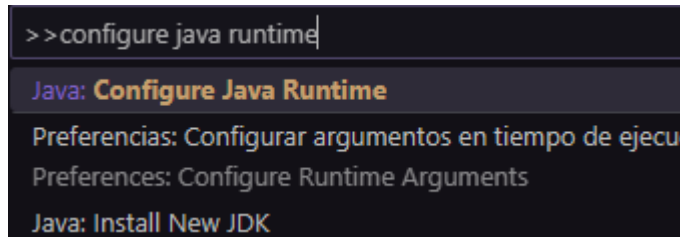
**GitHub de parámetros para Config Server:** <https://github.com/lko-developer-software/MELI>

## CONFIGURACION MANUAL

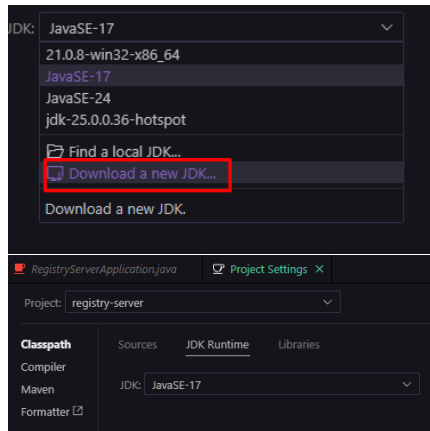
**Requisitos previos con visual studio code:**

Validar que se tenga configurado Java 17 y Maven.

- 1- Sobre visual studio code presiona: ctrl+chif+p
- 2- Busca la opción: configure java runtime

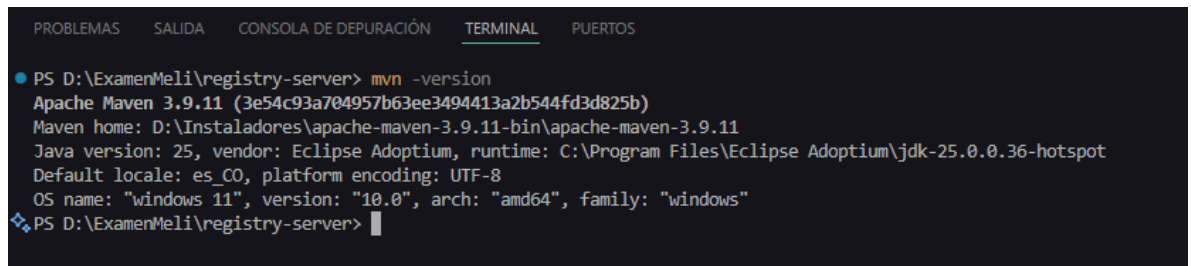


- 3- Instale la versión de java 17 de cualquiera de los proveedores:



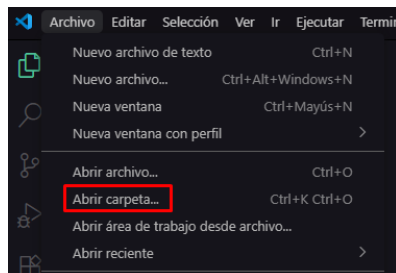
- 4- Validación de Maven:

Sobre visual studio code presiona `ctrl+chif+ñ` y ejecuta el comando `mvn -version` y utilice la versión 3.9.11



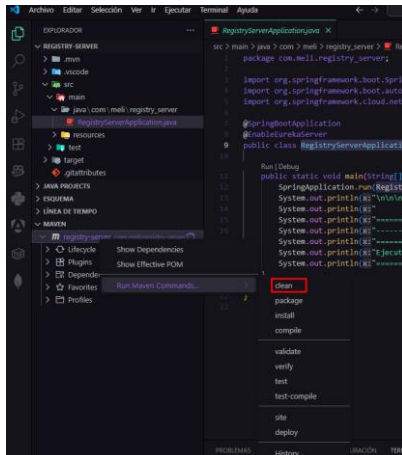
## PASO 1: Iniciar registry-server en visual studio code

-Descomprimos el archivo examenMELI.zip y abrimos la carpeta registry-server.

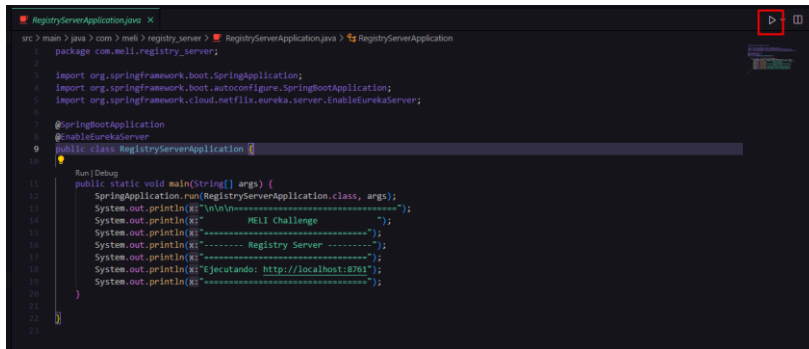


-Ejecutamos un Maven clean y luego un Maven install:

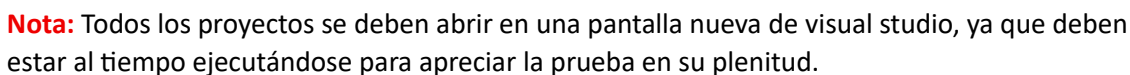




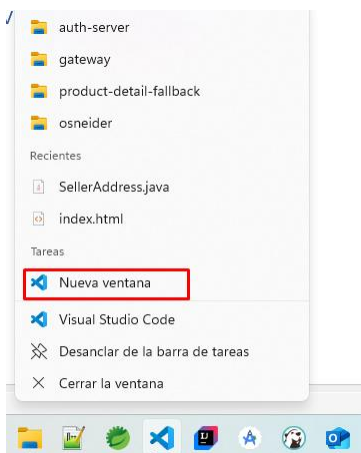
Posteriormente ejecutamos el proyecto:



Verificamos que este ejecutando:



-Abrir proyecto con visual studio code:



# Empresa: Mercado Libre - Colombia

## Prueba: Challenge - Backend Developer AI

### Versión Formato: 1.0



-Ejecutar

Maven clean, Maven install tal cual se indico en el paso anterior y ejecutar proyecto estando ubicados en el archivo ConfigServerApplication.java

```
src > main > java > com > meli > config_server > ConfigServerApplication.java > ConfigServerApplicati...
package com.meli.config_server;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.config.server.EnableConfigServer;

@SpringBootApplication
@EnableConfigServer
@EnableDiscoveryClient
public class ConfigServerApplication {

    Run | Debug
    public static void main(String[] args) {
        SpringApplication.run(ConfigServerApplication.class, args);
        System.out.println("\n\n-----");
        System.out.println(x"      MELI Challenge      ");
        System.out.println(x"----- Config Server -----");
        19 System.out.println(x"-----");
        System.out.println(x"Ejecutando: http://localhost:7777");
        System.out.println(x"-----");
    }
}
```

PS D:\ExamenMeli\config-server> & "C:\Program Files\Java\jdk-17\bin\java.exe" @"C:\Users\osnal\AppData\Local\Temp\cp\_91bo7pfbu6pwzjz5450wk8y3.argFile" "com.meli.config\_...server\ConfigServerApplication"

MELI Challenge

----- Config Server -----

Ejecutando: http://localhost:7777

### PASO 3: Iniciar product-detail en visual studio code

-Ejecutar Maven clean, Maven install tal cual se indico en el paso anterior y ejecutar proyecto estando ubicados en el archivo ProductDetailApplication.java

```
src > main > java > com > meli > product_detail > ProductDetailApplication.java > ProductDetailAppi...
18 public class ProductDetailApplication {
19
20     /**
21      * Punto de entrada principal de la aplicación.
22      *
23      * @param args argumentos de línea de comandos
24      */
25     Run | Debug
26     public static void main(String[] args) {
27         SpringApplication.run(ProductDetailApplication.class, args);
28
29         System.out.println("\n\n-----");
30         System.out.println(x"      MELI Challenge      ");
31         System.out.println(x"----- Product detail -----");
32         System.out.println(x"-----");
33         System.out.println(x"Ejecutando: http://localhost:8080");
34         System.out.println(x"-----");
35     }
36 }
37
38 }
```

2025-09-30T13:31:59.595-05:00 INFO [product-detail,,] 10964 --- [product-detail] [foRepliator-Md] [ com.netflix.discov

2025-09-30T13:31:59.598-05:00 INFO [product-detail,,] 10964 --- [product-detail] [foRepliator-Md] [ o.s.b.w.embedded.t

2025-09-30T13:31:59.599-05:00 INFO [product-detail,,] 10964 --- [product-detail] [foRepliator-Md] [ .s.c.n.e.s.EurekaA

2025-09-30T13:31:59.600-05:00 INFO [product-detail,,] 10964 --- [product-detail] [foRepliator-Md] [ com.netflix.discov

2025-09-30T13:31:59.781-05:00 INFO [product-detail,,] 10964 --- [product-detail] [foRepliator-Md] [ c.m.p.ProductDetail

2025-09-30T13:31:59.783-05:00 INFO [product-detail,,] 10964 --- [product-detail] [foRepliator-Md] [ .ConditionEvaluati

onDeltaLoggingListener : Condition evaluation unchanged

2025-09-30T13:32:00.594-05:00 WARN [product-detail,,] 10964 --- [product-detail] [foRepliator-Md] [ z.r.i.AsyncReporte

rBoundedSyncReporter : Spans were dropped due to exceptions. All subsequent errors will be logged at FINE level.

MELI Challenge

----- Product detail -----

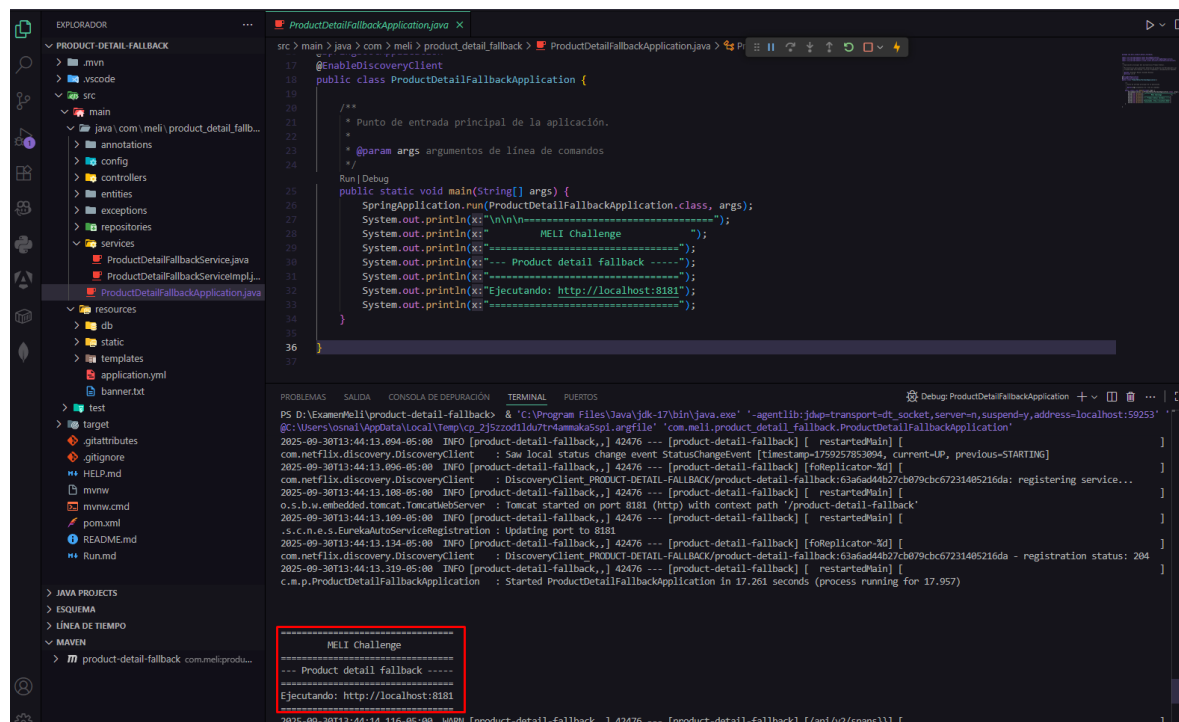
Ejecutando: http://localhost:8080

**Nota:** Si les

sale un error posterior a levantar el servicio no se alarmen, el servicio cuenta con monitoreo en zipkin.

## PASO 4: Iniciar product-detail-fallback en visual studio code

Ejecutar Maven clean, Maven install tal cual se indicó en el paso anterior y ejecutar proyecto estando ubicados en el archivo ProductDetailFallbackApplication.java



The screenshot shows the Visual Studio Code interface with the file explorer on the left, the editor in the center, and the terminal at the bottom. The editor displays the `ProductDetailFallbackApplication.java` file, which contains the following code:

```
src > main > java > com > meli > product_detail_fallback > ProductDetailFallbackApplication.java > P
17 @enableDiscoveryClient
18 public class ProductDetailFallbackApplication {
19
20     /**
21      * Punto de entrada principal de la aplicación.
22      *
23      * @param args argumentos de línea de comandos
24      */
25     Run [Debug]
26     public static void main(String[] args) {
27         SpringApplication.run(ProductDetailFallbackApplication.class, args);
28         System.out.println(x2"\\n\\n-----");
29         System.out.println(x2"          MELI Challenge          ");
30         System.out.println(x2"----- Product detail fallback -----");
31         System.out.println(x2"Ejecutando: http://localhost:8181");
32         System.out.println(x2"-----");
33     }
34 }
35
36
37
```

The terminal output shows the following logs:

```
PS D:\ExamenMeli\product-detail-fallback> & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=59253'
@C:\Users\Vosnal\AppData\Local\Temp\cp_2J5z2od1d07r4damaka5p1.argfile' 'com.meli.product_detail_fallback.ProductDetailFallbackApplication'
2025-09-30T13:44:13.094-05:00 INFO [product-detail-fallback,,] 42476 --- [product-detail-fallback] [ restarted@main] [
com.netflix.discovery.DiscoveryClient : Saw local status change event StatusChangeEvent [timestamp=1759257853094, current=UP, previous=STARTING]
2025-09-30T13:44:13.096-05:00 INFO [product-detail-fallback,,] 42476 --- [product-detail-fallback] [fokeplicator-3d] [
com.netflix.discovery.DiscoveryClient : DiscoveryClient PRODUCT-DETAIL-FALLBACK/product-detail-fallback:63a6ad44b27cb079cbc67231405216da: registering service...
2025-09-30T13:44:13.108-05:00 INFO [product-detail-fallback,,] 42476 --- [product-detail-fallback] [ restarted@main] [
o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8181 (http) with context path '/product-detail-fallback'
2025-09-30T13:44:13.109-05:00 INFO [product-detail-fallback,,] 42476 --- [product-detail-fallback] [ restarted@main] [
.s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8181
2025-09-30T13:44:13.114-05:00 INFO [product-detail-fallback,,] 42476 --- [product-detail-fallback] [fokeplicator-3d] [
com.netflix.discovery.DiscoveryClient : DiscoveryClient PRODUCT-DETAIL-FALLBACK/product-detail-fallback:63a6ad44b27cb079cbc67231405216da - registration status: 204
2025-09-30T13:44:13.319-05:00 INFO [product-detail-fallback,,] 42476 --- [product-detail-fallback] [ restarted@main] [
c.m.p.ProductDetailFallbackApplication : Started ProductDetailFallbackApplication in 17.261 seconds (process running for 17.957)

-----
MELI Challenge
-----
Product detail fallback
-----
Ejecutando: http://localhost:8181
-----
```

**Nota:** Si les sale un error posterior a levantar el servicio no se alarmen, el servicio cuenta con monitoreo en zipkin.

## PASO 5: Iniciar auth-server en visual studio code

Ejecutar Maven clean, Maven install tal cual se indicó en el paso anterior y ejecutar proyecto estando ubicados en el archivo ProductDetailFallbackApplication.java

# Empresa: Mercado Libre - Colombia

## Prueba: Challenge - Backend Developer AI

### Versión Formato: 1.0



The screenshot shows the Visual Studio Code interface with the `AuthServerApplication.java` file open. The file is located in the `src/main/java/com/meli/auth_server` directory. The code is a Spring Boot application that prints out the "MELI Challenge" and the URL `http://localhost:3030`. The terminal output shows the application running successfully on port 3030.

```
src > main > java > com > meli > auth_server > AuthServerApplication.java > AuthServerApplication
5 import org.springframework.boot.autoconfigure.SpringBootApplication;
6
7
8 @SpringBootApplication
9 public class AuthServerApplication {
10
11
12     Run [Debug]
13     public static void main(String[] args) {
14         SpringApplication.run(AuthServerApplication.class, args);
15     }
16
17     System.out.println("\n\n\n-----");
18     System.out.println("\n\n\nMELI Challenge\n\n\n");
19     System.out.println("\n\n\n----- auth-server -----");
20     System.out.println("\n\n\n-----");
21     System.out.println("\n\n\nEjecutando: http://localhost:3030");
22     System.out.println("\n\n\n-----");
23 }
24
25
26
```

Terminal Output:

```
PS D:\Exam\Heli\auth_server> & 'C:\Program Files\Java\jdk-17\bin\java.exe' ^-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:51042 '@C:\Users\osna1\
nal\AppData\Local\Temp\cp_2n0kcdog94nm07562gckfz..argfile' 'com.meli.auth_server.AuthServerApplication'
2025-09-30T16:59:42.845-05:00 INFO 43456 --- [auth-server] [ restartedMain] o.s.c.n.e.s.EurekaServiceRegistry : Registering application AUTH-SERVER with eureka w
ith status UP
2025-09-30T16:59:42.846-05:00 INFO 43456 --- [auth-server] [ restartedMain] com.netflix.discovery.DiscoveryClient : Saw local status change event StatusChangeEvent [
timestamp=1759269582846, current-UP, previous=STARTING]
2025-09-30T16:59:42.847-05:00 INFO 43456 --- [auth-server] [foReplicator-3d] com.netflix.discovery.DiscoveryClient : DiscoveryClient_AUTH-SERVER/auth-server:5e09d3848
a0eb73ca75530877d2172a9: registering service...
2025-09-30T16:59:42.869-05:00 INFO 43456 --- [auth-server] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 3030 (http) with context p
ath '/auth-server'
2025-09-30T16:59:42.870-05:00 INFO 43456 --- [auth-server] [ restartedMain] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 3030
2025-09-30T16:59:42.889-05:00 INFO 43456 --- [auth-server] [foReplicator-3d] com.netflix.discovery.DiscoveryClient : DiscoveryClient_AUTH-SERVER/auth-server:5e09d3848
a0eb73ca75530877d2172a9 - registration status: 204
2025-09-30T16:59:43.051-05:00 INFO 43456 --- [auth-server] [ restartedMain] c.m.auth_server.AuthServerApplication : Started AuthServerApplication in 9.337 seconds (p
rocess running for 10.018)

-----
MELI Challenge
-----
auth-server -----
Ejecutando: http://localhost:3030
```

## PASO 6: Iniciar Gateway en visual studio code

Ejecutar Maven clean, Maven install tal cual se indicó en el paso anterior y ejecutar proyecto estando ubicados en el archivo `ProductDetailFallbackApplication.java`

The screenshot shows the Visual Studio Code interface with the `GatewayApplication.java` file open. The file is located in the `src/main/java/com/meli/gateway` directory. The code is a Spring Boot application that prints out the "MELI Challenge" and the URL `http://localhost:4040`. The terminal output shows the application running successfully on port 4040.

```
src > main > java > com > meli > gateway > GatewayApplication.java > GatewayApplication
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
6
7 @SpringBootApplication
8 @EnableDiscoveryClient
9 public class GatewayApplication {
10
11
12     Run [Debug]
13     public static void main(String[] args) {
14         SpringApplication.run(GatewayApplication.class, args);
15     }
16
17     System.out.println("\n\n\n-----");
18     System.out.println("\n\n\nMELI Challenge\n\n\n");
19     System.out.println("\n\n\n----- Gateway -----");
20     System.out.println("\n\n\n-----");
21     System.out.println("\n\n\nEjecutando: http://localhost:4040");
22     System.out.println("\n\n\n-----");
23 }
24
```

Terminal Output:

```
PS D:\Exam\Heli\gateway> & 'C:\Program Files\Java\jdk-17\bin\java.exe' ^-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:51084 '@C:\Users\osna1\
AppData\Local\Temp\cp_2n0kcdog94nm07562gckfz..argfile' 'com.meli.gateway.GatewayApplication'
stamp=1759270361691, current-UP, previous=STARTING]
2025-09-30T17:12:41.693-05:00 INFO 15884 --- [gateway] [foReplicator-3d] com.netflix.discovery.DiscoveryClient : DiscoveryClient_GATEWAY/gateway:a05ce0fcf54a8c35509a3
3712e44ec: registering service...
2025-09-30T17:12:41.724-05:00 INFO 15884 --- [gateway] [foReplicator-3d] com.netflix.discovery.DiscoveryClient : DiscoveryClient_GATEWAY/gateway:a05ce0fcf54a8c35509a3
3712e44ec - registration status: 204
2025-09-30T17:12:41.919-05:00 INFO 15884 --- [gateway] [ main] o.s.b.w.embedded.netty.NettyWebServer : Netty started on port 4040 (http)
2025-09-30T17:12:41.920-05:00 INFO 15884 --- [gateway] [ main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 4040
2025-09-30T17:12:42.288-05:00 INFO 15884 --- [gateway] [ main] com.meli.gateway.GatewayApplication : Started GatewayApplication in 6.111 seconds (process
running for 6.669)

-----
MELI Challenge
-----
Gateway -----
Ejecutando: http://localhost:4040
```

## PASO 7: Pruebas en Postman

Dentro de la documentación se encuentra archivo: ChallengeMeli.postman\_collection.json

Importar proyecto en postman he iniciar pruebas.



Se debe consumir el servicio Product Detail desde el Gateway

Paso #1 Generar Access token

Usuarios:

adminDev :admin123

vendorDev :vendor123

clientDev :client123

Paso #2 Consumir servicio de productDetail pasando como autorizador el acceso token por la opción de Beare Token

