

# Documentación Pokemón Bytes



# **Documentación – Fase V**

## *Índice*

### **1. Introducción y Objetivos**

- Automatización mediante Data Seeding.

### **2. Arquitectura del Cliente Reactivo**

- 2.1. Uso de WebClient vs RestTemplate.
- 2.2. Estrategia ETL (Extract, Transform, Load).

### **3. Implementación del Data Seeding (DataLoader)**

- 3.1. Orquestación Reactiva (Project Reactor y Mono.zip).
- 3.2. Manejo de Buffers y Lotes.

### **4. Trazabilidad del Código**

- 4.1. Configuración de Buffer en Poke ApiService.
- 4.2. Lógica de Carga Masiva en DataLoader.

### **5. Diagramas y Evidencia**

- 5.1. Diagrama de Secuencia Asíncrono.
- 5.2 Logs del Sistema

## 1. Introducción y Objetivos

En un RPG de la escala de Pokémon, gestionar manualmente la información de cientos de criaturas y movimientos es inviable y propenso a errores humanos. El objetivo de esta fase ha sido automatizar la población de la base de datos (**Data Seeding**) conectando el backend con la **PokéAPI** (la fuente de datos pública oficial).

El sistema implementa un cliente HTTP reactivo que, al iniciar el servidor, verifica la integridad de los datos locales y, si es necesario, descarga, procesa y almacena la información de la Generación II (251 Pokémon y Movimientos) de manera autónoma.

Fuente oficial API: <https://pokeapi.co/docs/v2#games-section>

## 2. Arquitectura del Cliente Reactivo

Para esta tarea no se utilizó el cliente tradicional (RestTemplate, bloqueante), sino **WebClient** (parte del stack **Spring WebFlux**).

### 2.1. Justificación

- **Asincronía y Paralelismo:** La carga inicial requiere realizar más de **500 peticiones HTTP** (251 Pokémon + 251 Especies + Movimientos). Hacerlo de forma secuencial llevaría varios minutos de arranque. Con WebClient y Flux, las peticiones se lanzan en paralelo de forma no bloqueante.
- **Gestión de Memoria:** Se configuró un ExchangeStrategies personalizado para aumentar el buffer de memoria a **16MB**, necesario para procesar los grandes JSONs de respuesta de la API sin desbordamientos de pila.

### 2.2. Flujo de Datos (ETL)

El proceso sigue un esquema **ETL (Extract, Transform, Load)**:

1. **Extract:** Descarga de JSONs crudos desde pokeapi.co.
2. **Transform:** Mapeo de campos anidados (JSON) a la estructura plana de la entidad JPA (PokedexMaestra). Fusión de datos de "Detalles" (Stats) y "Especies" (Ratio de captura).
3. **Load:** Persistencia en MySQL mediante saveAll() en lotes para optimizar el rendimiento de la base de datos.

## 3. Implementación Data Seeding (DataLoader)

La clase DataLoader.java implementa la interfaz CommandLineRunner, lo que garantiza su ejecución automática justo después de que el contexto de Spring Boot se haya levantado.

### 3.1. Orquestación Reactiva

El código utiliza operadores de Reactor (Flux, Mono.zip, flatMap) para combinar múltiples fuentes de datos.

#### Lógica Implementada:

- **Verificación de Estado:** if (pokedexRepository.count() < POKEMON\_LIMIT) evita descargas innecesarias si los datos ya existen.

- **Fusión de Peticiones (Mono.zip):** Para cada Pokémon, se necesitan dos endpoints distintos: /pokemon/{id} (para stats y tipos) y /pokemon-species/{id} (para el ratio de captura). El sistema lanza ambas peticiones simultáneamente y combina los resultados en una tupla cuando ambas han finalizado.

### 3.2. Configuración del Cliente (Poke ApiService)

El servicio Poke ApiService encapsula la configuración de bajo nivel del cliente HTTP.

## 4. Trazabilidad del Código

A continuación, los fragmentos clave que demuestran la implementación .

### 4.1. Configuración de WebClient (Poke ApiService.java)

```
@Service
public class Poke ApiService {
    private final WebClient webClient;

    public Poke ApiService(WebClient.Builder webClientBuilder) {
        // Aumento del límite de buffer a 16MB para soportar JSONs grandes
        final int maxBufferSize = 16 * 1024 * 1024;
        final ExchangeStrategies strategies = ExchangeStrategies.builder()
            .codecs(codecs ->
codecs.defaultCodecs().maxInMemorySize(maxBufferSize))
            .build();

        this.webClient = webClientBuilder
            .baseUrl("https://pokeapi.co/api/v2/")
            .exchangeStrategies(strategies)
            .build();
    }

    // Petición Asíncrona que devuelve un Mono
    public Mono<Map<String, Object>> getPokemonDetails(String name) {
        return webClient.get()
            .uri("pokemon/{name}", name)
            .retrieve()
            .bodyToMono(new ParameterizedTypeReference<Map<String,
Object>>() {});
    }
}
```

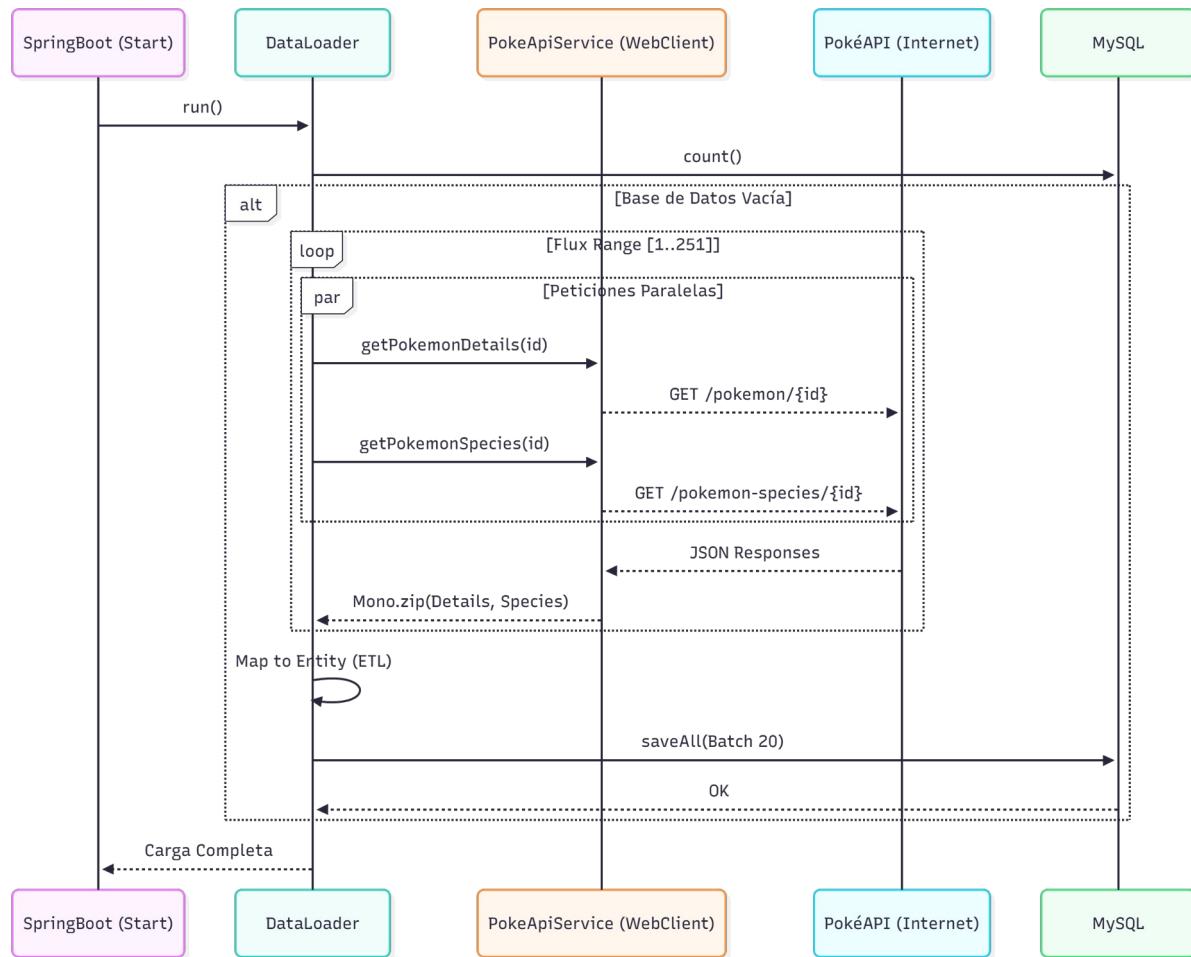
```
    }  
}
```

#### 4.2. Lógica de Carga Masiva (DataLoader.java)

```
@Component  
public class DataLoader implements CommandLineRunner {  
  
    // Constantes de Gen II  
    private static final int POKEMON_LIMIT = 251;  
  
    @Override  
    public void run(String... args) {  
        if (pokedexRepository.count() < POKEMON_LIMIT) {  
  
            // Flujo Reactivo: Itera del 1 al 251  
            Flux.range(1, POKEMON_LIMIT)  
                .flatMap(id ->  
                    // Fusión de dos llamadas API en paralelo  
                    Mono.zip(  
                        apiService.getPokemonDetails(String.valueOf(id)), // T1:  
                        Stats  
                        apiService.getPokemonSpecies(String.valueOf(id)) // T2:  
                        Ratio Captura  
                    )  
                    , 5) // Concurrencia controlada (5 hilos)  
  
                .map(tuple -> mapCombinedDataToEntity(tuple.getT1(),  
                    tuple.getT2()))  
                .buffer(20) // Agrupa en listas de 20 para inserción  
                .doOnNext(pokedexRepository::saveAll) // Persistencia en MySQL  
                .blockLast(); // Espera a que termine el proceso antes de liberar  
                el hilo  
        }  
    }  
}
```

## 5. Diagramas y Evidencia

### 5.1. Diagrama de Secuencia Asíncrono.



## 5.2 Logs del Sistema

```
PS C:\Users\jesus\OneDrive\Escritorio\Pokémon Bytes> & 'C:\Users\jesus\.vscode\extensions\redhat.java-1.50.0-win32-x64\jre\21.0.9-win32-x86_64\bin\java.exe' '@C:\Users\jesus\AppData\Local\Temp\cp_240\14687\9psocwlcn@cbnsd.angfile' 'com.proyecto.pokemon_backend.PokemonBackendApplication'

:: Spring Boot ::          (v3.5.8)

2025-12-08T18:35:18.775+01:00  INFO 14288 --- [pokemon-backend] [main] c.p.p.PokemonBackendApplication      : Starting PokemonBackendApplication using Java 21.0.9 with PID 14288 (C:\Users\jesus\OneDrive\Escritorio\Pokémon Bytes\pokemon-backend\target\classes started by jesus in C:\Users\jesus\OneDrive\Escritorio\Pokémon Bytes)
2025-12-08T18:35:18.776+01:00  INFO 14288 --- [pokemon-backend] [main] c.p.p.PokemonBackendApplication      : No active profile set, falling back to 1 default profile: "default"
2025-12-08T18:35:19.970+01:00  INFO 14288 --- [pokemon-backend] [main] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2025-12-08T18:35:20.107+01:00  INFO 14288 --- [pokemon-backend] [main] s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 127 ms. Found 7 JPA repository interfaces.
2025-12-08T18:35:20.625+01:00  INFO 14288 --- [pokemon-backend] [main] o.s.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8881 (http)
2025-12-08T18:35:20.657+01:00  INFO 14288 --- [pokemon-backend] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-12-08T18:35:20.657+01:00  INFO 14288 --- [pokemon-backend] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.49]
2025-12-08T18:35:20.715+01:00  INFO 14288 --- [pokemon-backend] [main] o.a.c.c.C.[Tomcat].[localhost].[]   : Initializing Spring embedded WebApplicationContext
2025-12-08T18:35:20.715+01:00  INFO 14288 --- [pokemon-backend] [main] w.s.c.ServletContextListenerApplicationEvent : Root WebApplicationContext: initialization completed in 1890 ms
2025-12-08T18:35:20.930+01:00  INFO 14288 --- [pokemon-backend] [main] o.hibernate.jpa.internal.util.LogHelper : HHH0003204: Processing PersistenceUnitInfo [name: default]
2025-12-08T18:35:21.002+01:00  INFO 14288 --- [pokemon-backend] [main] org.hibernate.Version       : HHH000412: Hibernate ORM core version 6.6.36.Final
2025-12-08T18:35:21.040+01:00  INFO 14288 --- [pokemon-backend] [main] o.h.c.internal.RegionFactoryInitiator : HHH000026: Second-level cache disabled
2025-12-08T18:35:21.324+01:00  INFO 14288 --- [pokemon-backend] [main] o.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: ignoring JPA class transformer
2025-12-08T18:35:21.354+01:00  INFO 14288 --- [pokemon-backend] [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2025-12-08T18:35:21.393+01:00  INFO 14288 --- [pokemon-backend] [main] com.zaxxer.hikari.pool.HikariPool  : HikariPool-1 - Added connection com.mysql.cj.jdbc.ConnectionImpl@663622b1
2025-12-08T18:35:21.505+01:00  INFO 14288 --- [pokemon-backend] [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2025-12-08T18:35:21.595+01:00  INFO 14288 --- [pokemon-backend] [main] org.hibernate.orm.connections.Pooling : HHH01001005: Database info:
Database JDBC URL [Connecting through datasource 'HikariDataSource (HikariPool-1)']
Database driver: undefined/unknown
Database version: 8.0.40
Autocommit mode: undefined/unknown
Isolation level: undefined/unknown
Minimum pool size: undefined/known
Maximum pool size: undefined/known
2025-12-08T18:35:22.355+01:00  INFO 14288 --- [pokemon-backend] [main] o.h.e.t.j.p.i.JtaPlatformInitiator      : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
2025-12-08T18:35:22.395+01:00  INFO 14288 --- [pokemon-backend] [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2025-12-08T18:35:23.079+01:00  INFO 14288 --- [pokemon-backend] [main] e.AuthenticationProviderManagerConfigurer : Global AuthenticationManager configured with AuthenticationProvider bean with name authenticationProvider
2025-12-08T18:35:23.080+01:00  WARN 14288 --- [pokemon-backend] [main] r$InitializeUserDetailsServiceManagerConfigurer : Global AuthenticationManager configured with an AuthenticationProvider bean. UserDetailsService beans will not be used by Spring Security for automatically configuring username/password login. Consider removing the AuthenticationProvider bean. Alternatively, consider using the UserDetailsService in a manually instantiated DaoAuthenticationProvider. If the current configuration is intentional, to turn off this warning, increase the logging level of 'org.springframework.security.config.annotation.authentication.configuration.InitializeUserDetailsServiceManagerConfigurer' to ERROR
2025-12-08T18:35:23.175+01:00  WARN 14288 --- [pokemon-backend] [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2025-12-08T18:35:23.814+01:00  INFO 14288 --- [pokemon-backend] [main] o.s.b.a.e.web.EndpointLinksResolver    : Exposing 1 endpoint beneath base path '/actuator'
2025-12-08T18:35:23.825+01:00  INFO 14288 --- [pokemon-backend] [main] c.p.p.PokemonBackendApplication      : Started PokemonBackendApplication in 5.475 seconds (process running for 6.031)
Hibernate:
  select
    count(*)
  from
    tipos t1_0
  hibernate:
  select
    count(*)
  from
    items ii_0
  hibernate:
  select
    count(*)
  from
    pokedex_maestra pm1_0
  hibernate:
  select
    count(*)
  from
    ataques a1_0
2025-12-08T18:35:24.694+01:00  INFO 14288 --- [pokemon-backend] [on(2)-127.0.0.1] o.a.c.c.C.[Tomcat].[localhost].[]   : Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-12-08T18:35:24.694+01:00  INFO 14288 --- [pokemon-backend] [on(2)-127.0.0.1] o.s.web.servlet.DispatcherServlet      : Initializing DispatcherServlet 'dispatcherServlet'
2025-12-08T18:35:24.696+01:00  INFO 14288 --- [pokemon-backend] [on(2)-127.0.0.1] o.s.web.servlet.DispatcherServlet      : Completed initialization in 2 ms
```

Captura de la terminal durante el despliegue de PokemonBackendApplication. Se observa inicialización exitosa del Tomcat en el puerto 8081, la conexión a la base de datos MySQL mediante el pool HikariCP y la ejecución de las consultas de verificación de integridad (Hibernate Select Count) de los Seeders.

## Análisis de Logs

- *Tomcat initialized with port 8081*: Confirma que la configuración en application.properties (server.port=8081) funciona y no choca con el puerto 8080 estándar.
- *HikariPool-1 - Start completed*: La conexión con MySQL es exitosa. El usuario/contraseña son correctos.
- *Hibernate: select count(\*) ...*: Estas líneas finales demuestran que los componentes DataLoader, ItemSeeder y TipInitializer se están ejecutando.
  - Como solo salen los select count(\*), significa que la base de datos ya tiene los datos cargados, por lo que el sistema inteligentemente se salta la descarga de la PokeAPI (evitando perder tiempo).
- *Started PokemonBackendApplication in 5.475 seconds*: El tiempo de arranque es bueno (menos de 6 segundos).

