

# Prueba Técnica

Martin Caffarena

Marzo 2025

Repo: [Github](#)

## Tecnologías Utilizadas:

### Backend:

- **Java 21:** Lenguaje de programación.
- **Spring Boot 3:** Framework para desarrollo de la API REST.

### Base de datos:

- **MySQL:** Base de datos relacional.
- **Spring Data JPA:** ORM para interactuar con MySQL.

### Testing:

- **JUnit 5:** Pruebas unitarias.
- **Mockito:** Simulación de dependencias en pruebas.
- **Github Action:** Automatización CI/CD

### Despliegue:

- **Docker:** Contenerización del proyecto.
- **AWS Elastic Beanstalk:** Despliegue de la aplicación en la nube.
- **AWS RDS (MySQL):** Base de datos en la nube.

## Arquitectura:

La solución implementa una arquitectura monolítica en capas, separando las responsabilidades en:

- **Controladores (Controller):** Exponen los endpoints REST y gestionan las solicitudes HTTP.
- **Lógica de Negocio (Business):** Procesa la lógica de la aplicación y orquesta la comunicación con la capa de datos.
- **Acceso a Datos (Data Access):** Maneja la interacción con la base de datos MySQL a través de JPA.

Se han aplicado principios de responsabilidad única (Single Responsibility), buenas prácticas de desarrollo y Clean Code, asegurando código modular, mantenible y eficiente.

## Funcionalidades:

### N-ésimo número de Fibonacci:

Permite obtener el n-ésimo valor de la serie numérica de Fibonacci

Endpoint:

GET /api/fibonacci/{n}

Funcionamiento:

- Si el número ya ha sido calculado previamente, se recupera desde la base de datos para mejorar el rendimiento (simulación de caché)
- En caso contrario se calcula el resultado, se almacena en la base de datos para futuras consultas y se devuelve la respuesta

### Estadísticas números consultados:

Devuelve el número de veces que fue consultado cada número

Endpoint:

GET /api/fibonacci/stats

Funcionamiento:

- Se obtiene una lista de todos los números consultados con la cantidad de veces que han sido solicitados

## Collection de Postman:

Ubicada en la raíz del proyecto dentro de la carpeta Documentación.

## Tests

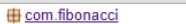

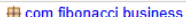

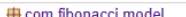

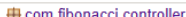

Se realizaron pruebas unitarias en las tres capas del proyecto:

- Controladores (Controllers)
- Lógica de Negocio (Business Logic)
- Acceso a Datos (Data Access)

Se logró una cobertura de código del 97%

Reporte de herramienta de cobertura Jacoco:

### fibonacci-api

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Ctxty	Missed Lines	Missed Methods	Missed Classes
com.fibonacci		37 %		n/a	1 2	2 3	1 2	0 1
com.fibonacci.business		100 %		100 %	0 11	0 28	0 6	0 2
com.fibonacci.model		100 %		n/a	0 13	0 24	0 13	0 2
com.fibonacci.controller		100 %		100 %	0 4	0 11	0 3	0 1
Total	5 of 216	97 %	0 of 12	100 %	1 30	2 66	1 24	0 6

Este reporte podemos verlo al ejecutar dentro del proyecto los siguientes comandos

```
mvn clean test
```

```
mvn jacoco:report
```

Esto generará un archivo `index.html` en la ruta:

```
...\LaBancaPrueba\fibonacci-api\target\site\jacoco\index.html
```

A su vez se implementó un flujo de trabajo para que los tests y un build sea ejecutado con cada commit y PR en el repositorio de Github

## Ejecución del proyecto:

Descripción completa para la ejecución del proyecto se encuentra en el archivo `README.md` ubicado en la ruta del proyecto