

Universidad de Las Américas
Facultad de Ingenierías y Ciencias Agropecuarias
Ingeniería de Software
Informe de laboratorio

1. DATOS DEL ALUMNO: Camila Cabrera

2. TEMA DE LA PRÁCTICA: Guía práctica de Caching Data con Spring

3. OBJETIVO DE LA PRÁCTICA

- Explorar el Framework Spring a través de la implementación de código en base a la guía práctica de data en cache.

4. OBJETIVOS ESPECÍFICOS

- Instalar el JDK 17
- Inicializar de Visual Studio Code para proyectos de java con mave
- Crear proyecto con Spring Initializr
- Implementar la documentación para el proyecto de data en cache

5. MATERIALES Y MÉTODOS

Materiales:

- Visual studio code
- Maven 3.5+
- JDK 17
- Guía práctica de Spring

Métodos:

1. Instalación del JDK 17.
2. Configuración de Visual Studio Code para proyectos de Java con Maven.
3. Creación del proyecto Spring utilizando Spring Initializr, seleccionando las dependencias necesarias para la implementación del proyecto.

4. Implementación de la guía práctica de Spring para la configuración y uso de caché de datos.
5. Registro de hallazgos durante el proceso de implementación de la documentación de Spring.

6. DESARROLLO DE LA PRÁCTICA Y RESULTADOS

Marco Teórico

Un sistema de caché sirve para almacenar datos, recursos o elementos para ser accedidos posteriormente con mayor facilidad. En el caso de los sitios web esto es importante, dado que en este tipo de programa se busca mejorar siempre mejorar el rendimiento y la velocidad de esto.

Implementar un sistema de cache permite en un sitio web, enviar los datos al usuario a mayor velocidad, dado que se guarda contenido en memoria RAM, mejorando el tiempo de envío. En cache se tiende a guardar solicitudes recurrentes a base de datos e incluso imágenes o estilos. Hay dos tipos principales de sistemas de cache, los que corren a nivel del servidor y los que funcionan como parte de un CDN (servicio externo al servidor que busca mejorar la velocidad de carga del sitio)

Redis vs Memcached

“Redis es un almacén de estructura de datos en memoria de código abierto que permite a los desarrolladores crear aplicaciones de alto rendimiento con persistencia de datos” (IACARO, 2023). Se puede almacenar datos como cadenas. Hashes, listas o conjuntos. Se pueden implementar clústers permitiendo la escalabilidad de capas en almacenamiento de cachés sin intervención de los servidores o aplicativos. Es compatible con tecnologías web comunes (Python, PHP, Java).

“Memcached es un sistema de almacenamiento en caché de memoria distribuida que funciona almacenando datos de uso frecuente en la memoria para una recuperación rápida” (IACARO, 2023). Permite almacenar datos básicos

provenientes de una base, en memoria ocasionando que los tiempos de acceso sean más rápidos. Reduce la latencia y la carga de los servidores de aplicaciones.

Tabla #1 : Ventajas y desventajas Redis vs Memcached

	Redis	Memcached
Ventajas	<ul style="list-style-type: none"> - Persistencia de datos configurable, se puede almacenar datos en disco. - Alta disponibilidad mediante replicación de datos en varios servidores - Procesamiento en tiempo real de grandes conjuntos de datos 	<ul style="list-style-type: none"> - Fácil instalación en servidores dedicados - Bajo uso de memoria y gran rendimiento - Almacenamiento de pares clave-valor - Almacenamiento de datos simples
Desventajas	<ul style="list-style-type: none"> - Puede requerir una cantidad sustancial de recursos del servidor - Escalabilidad limitada en conjunto de datos muy grandes 	<ul style="list-style-type: none"> - Almacenada los datos en forma de texto - No replica datos en varios nodos

Fuente: Elaboración propia

Imagen #2: Características Redis vs Memcached

	Memcached	Redis
Latencia inferior a un milisegundo	Sí	Sí
Facilidad de uso para los desarrolladores	Sí	Sí
Partición de datos	Sí	Sí
Compatibilidad con un amplio conjunto de lenguajes de programación	Sí	Sí
Estructuras de datos avanzadas	-	Sí
Arquitectura de multiproceso	Sí	-
Instantáneas	-	Sí
Replicación	-	Sí
Transacciones	-	Sí
Publicación/suscripción	-	Sí
Scripting de Lua	-	Sí
Soporte geoespacial	-	Sí

Fuente: (AWS, s.f.)

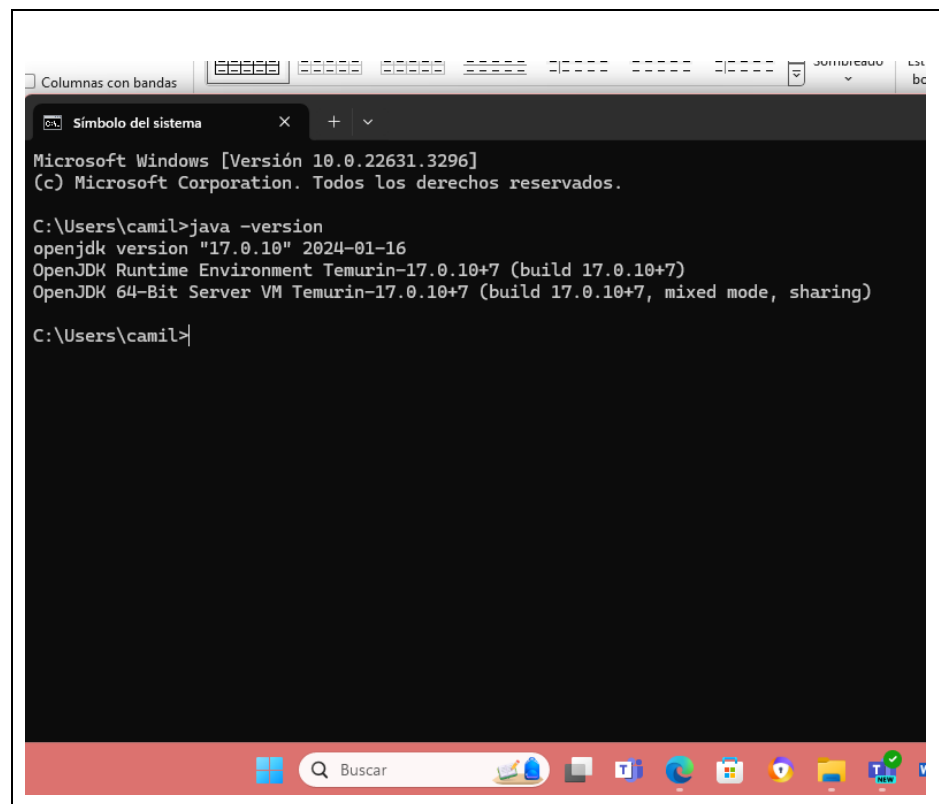
En términos de latencia ambos admiten tiempos de respuesta menores a un milisegundo, siendo fáciles de usar desde el punto de vista sintáctico y requieren una cantidad mínima de código. Cuando la demanda aumenta se puede escalar horizontalmente distribuyendo los datos en varios nodos. Ambos, son compatibles en amplios lenguajes de programación. Sin embargo, únicamente Redis admite listas, conjuntos, conjuntos ordenados, hashes, matrices y conjunto de datos más avanzados. Respecto a la arquitectura de multiprocesos solo Memcached admite varios procesos, pudiendo utilizar varios núcleos. Una característica importante de Redis es que se puede configurar para mantener datos en el disco como seguridad de datos.

En general para decidir cual sistema utilizar, es necesario saber los requisitos específicos del proyecto. Memcached destaca en simplicidad y velocidad mientras que Redis brinda funcionalidades avanzadas y flexibilidad.

Desarrollo de la práctica

- **Instalación del JDK 17:**
 - Con la ayuda del IDE Visual code se instaló el jdk 17.
 - Para verificar la instalación correcta se comprueba con el comando `java -version` en la terminal del computador.

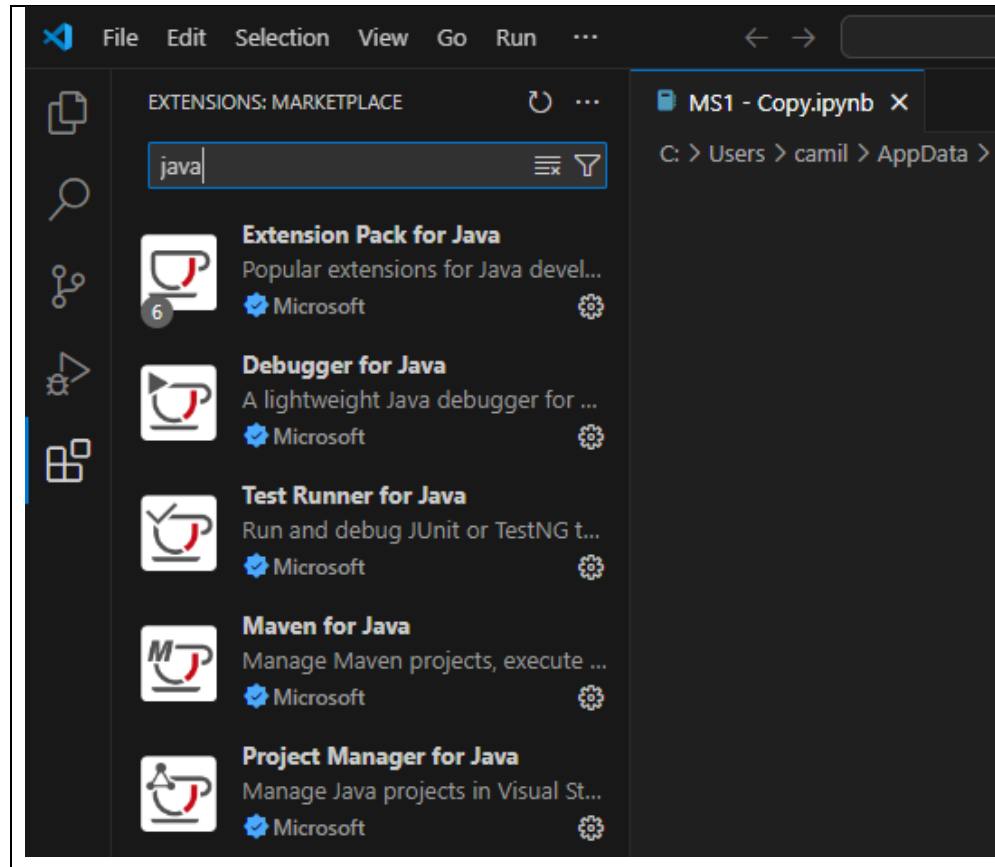
Imagen #3: Validación versión JDK



Fuente: Elaboración propia

- **Inicialización de Visual Studio Code para proyectos de Java con Maven:**
 - En el IDE visual studio code se instalaron la dependencia para Java (Imagen 3) y la extensión para trabajar con Maven.

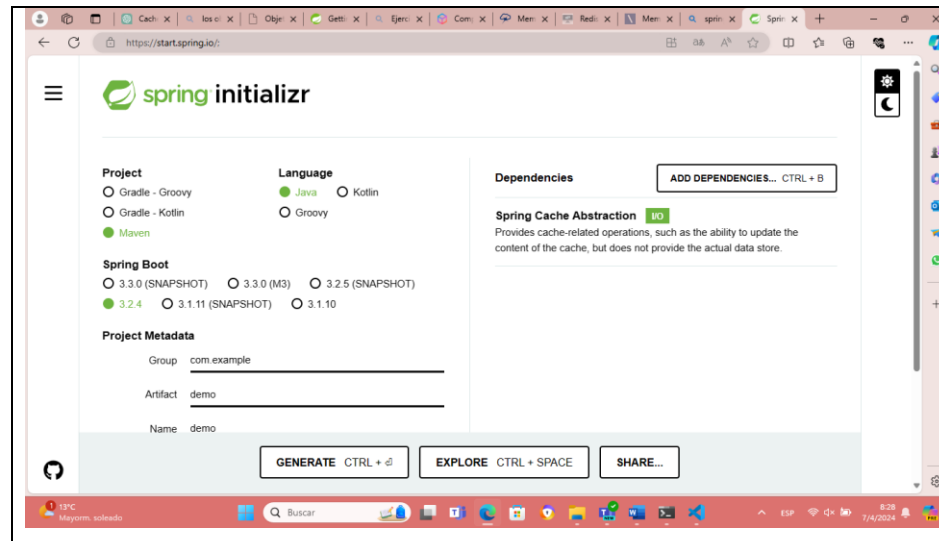
Imagen #3: Dependencia de Java



Fuente: Elaboración propia

- **Creación del Proyecto con Spring Initializr:**
 - Se accedió a Spring Initializr desde el navegador web, seleccionando las configuraciones para el proyecto
 - Se descarga el proyecto generado y se importa en Visual Studio Code.

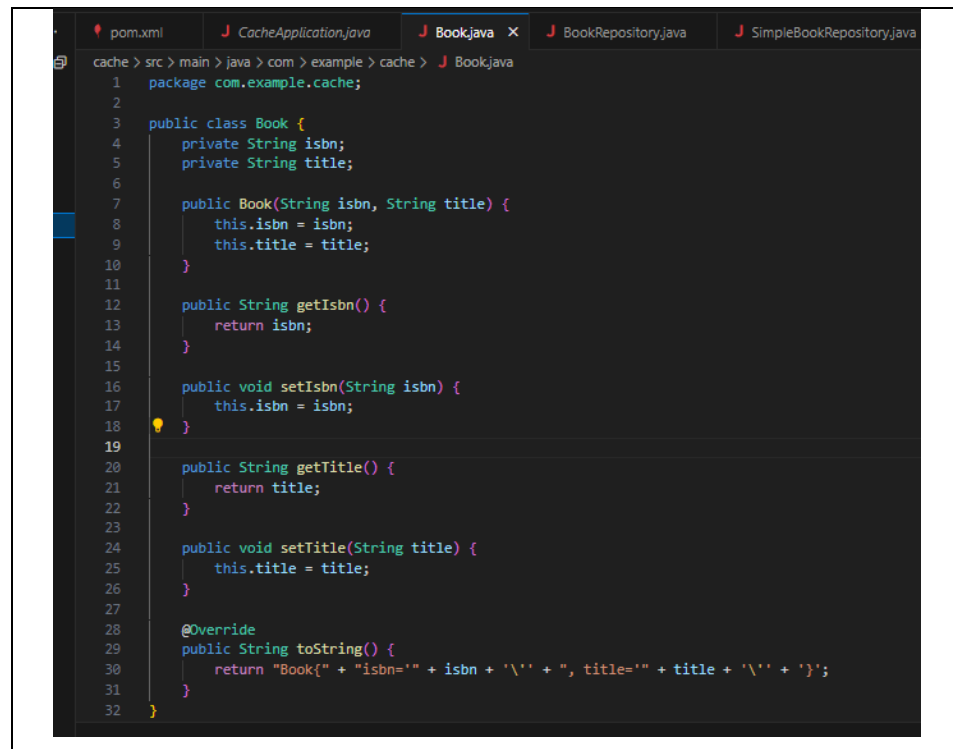
Imagen #4: Spring Initializr



Fuente: Elaboración propia

- **Implementación de la Documentación para el Proyecto de Data en Caché:**
 - Se siguió la guía práctica de Spring obtenida en el siguiente link: [Getting Started | Caching Data with Spring](#) para la configuración y uso del caché de datos.
 - Se agregaron las notaciones spring necesarias para las clases y métodos para habilitar la caché y definir los datos que se almacenarán en ella.
 - Primero se creó una clase llamada Book, siendo el modelo de dato que se va a manejar. Con dos atributos el isbn y el título.

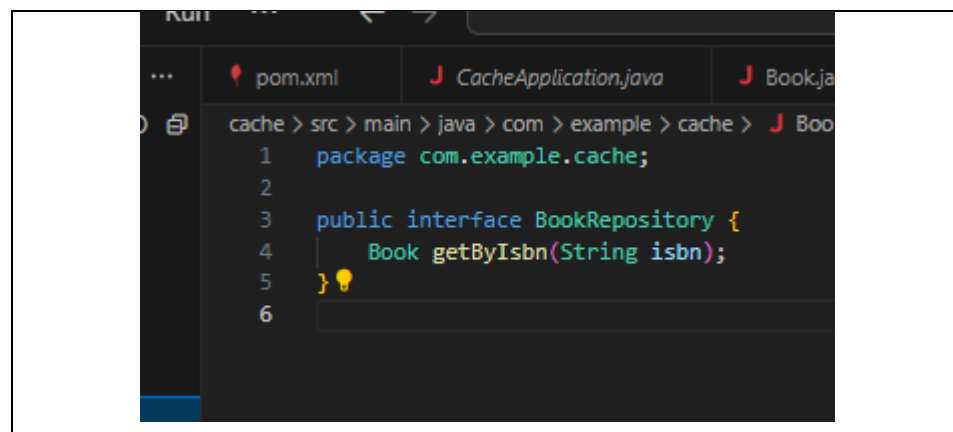
Imagen #5: Clase Book



```
1 package com.example.cache;
2
3 public class Book {
4     private String isbn;
5     private String title;
6
7     public Book(String isbn, String title) {
8         this.isbn = isbn;
9         this.title = title;
10    }
11
12    public String getIsbn() {
13        return isbn;
14    }
15
16    public void setIsbn(String isbn) {
17        this.isbn = isbn;
18    }
19
20    public String getTitle() {
21        return title;
22    }
23
24    public void setTitle(String title) {
25        this.title = title;
26    }
27
28    @Override
29    public String toString() {
30        return "Book{" + "isbn='" + isbn + '\'' + ", title='" + title + '\'' + '}';
31    }
32 }
```

- Luego una interfaz repositorio donde se declaro un método para obtener un libro dado un isbn

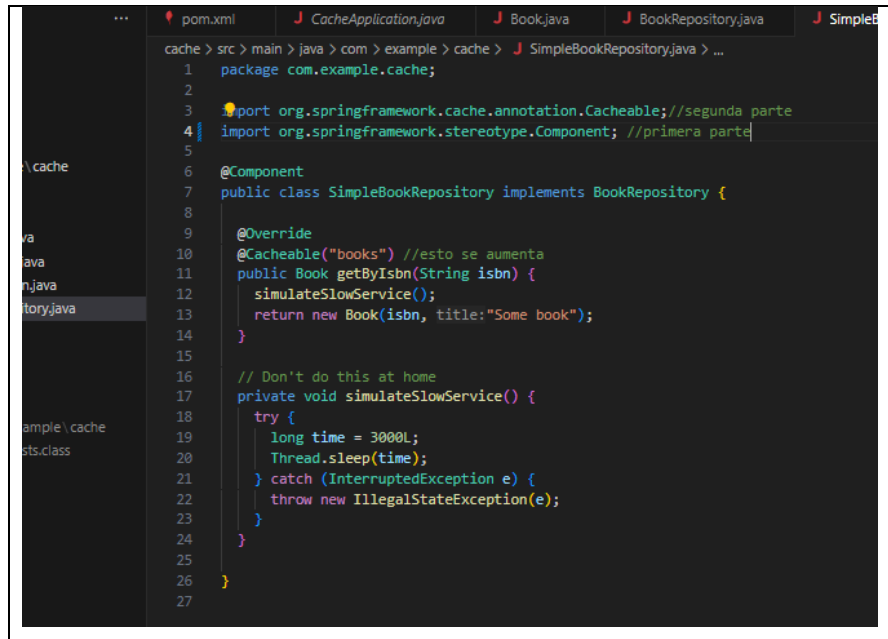
Imagen #6: Interfaz



```
1 package com.example.cache;
2
3 public interface BookRepository {
4     Book getByIsbn(String isbn);
5 }
6
```


- En una clase llamada SimpleBookRepository se implementó la interfaz, en la cual se codificó el método para simular la latencia de recibir desde una fuente externa, un libro a partir de un ISBN.

Imagen #6: Clase SimpleBookRepository



```
cache > src > main > java > com > example > cache > SimpleBookRepository.java > ...
1 package com.example.cache;
2
3 import org.springframework.cache.annotation.Cacheable; //segunda parte
4 import org.springframework.stereotype.Component; //primera parte
5
6 @Component
7 public class SimpleBookRepository implements BookRepository {
8
9     @Override
10    @Cacheable("books") //esto se aumenta
11    public Book getByIsbn(String isbn) {
12        simulateSlowService();
13        return new Book(isbn, title:"Some book");
14    }
15
16    // Don't do this at home
17    private void simulateSlowService() {
18        try {
19            long time = 3000L;
20            Thread.sleep(time);
21        } catch (InterruptedException e) {
22            throw new IllegalStateException(e);
23        }
24    }
25
26 }
27
```

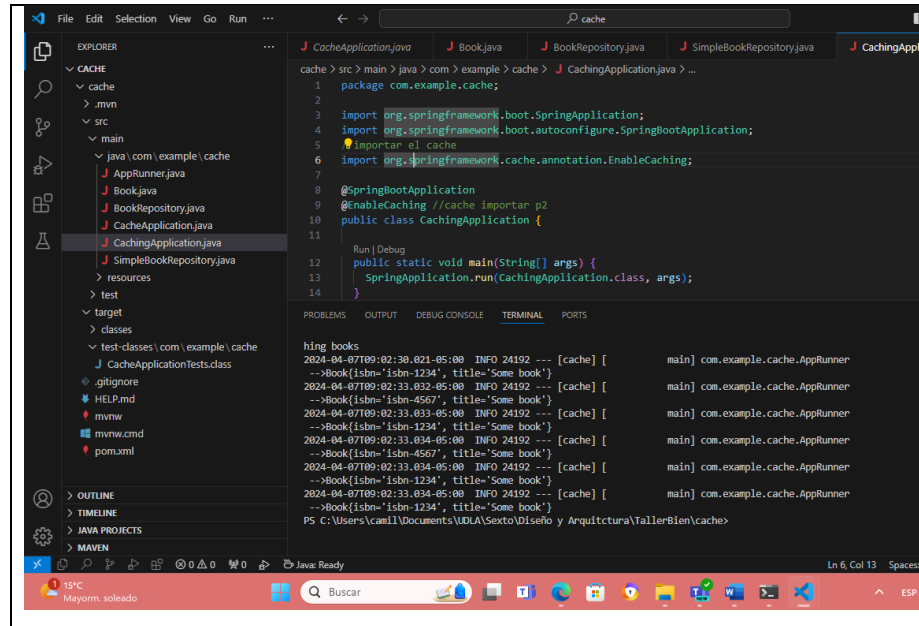
- Luego en la clase que alberga el método main CachingSpplication. se usa la notación @SpringBootApplication para añadir los siguientes tags. @Configuration, @EnableAutoConfiguration y @ComponentScan. También se usa dentro del método main() SpringApplication.run() para inicializar la aplicación sin necesidad de incluir ninguna linea xml

Imagen #7: Clase CachingApplication

```
cache > src > main > java > com > example > cache > CachingApplication.java > ...
1  package com.example.cache;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  //importar el cache
6  import org.springframework.cache.annotation.EnableCaching;
7
8  @SpringBootApplication
9  @EnableCaching //cache importar p2
10 public class CachingApplication {
11
12     Run | Debug
13     public static void main(String[] args) {
14         SpringApplication.run(CachingApplication.class, args);
15     }
16 }
17
```

- Luego se crea la clase AppRuner donde se establece la inyección del repositorio del libreo y se lo llama varias veces con diferentes argumentos
- Cuando se prueba hasta este punto la aplicación, la obtención de resultados es muy lenta, Cumpliendo con el delay de segundos puesto.
- Ahora se habilita el cache
- En la clase SimpleBookRepository se coloca la notación @Cacheable de tipo book
- Luego en la clase CachingApplication se habilita la notación @EnableCaching
- Cuando se vuelve a probar la aplicación esta va mucho más rápido y es efectiva al introducir un isbn ya consultado dado que se guarda en cache.

Imagen #4: Ejecución del programa



Fuente: Elaboración propia

• Pruebas y Evaluación de los Resultados:

- Como resultado se pudo cumplir con la guía de la práctica y observar cómo mejoro la calidad de visualización de los datos al habilitar el cache mediante las notaciones Spring, sin necesidad de colocar código extenso. Al correr la primera vez el sistema se observó como cada búsqueda del libro tomaba los tres segundos de latencia, mientras que al aplicar el cache solo las que eran únicas se demoraban mientras que las que ya habían sido consultadas aparecían con mayor rapidez.

7. OPINIÓN PERSONAL

- Encontré muy interesante que el framework Spring proporcioné una interfaz gráfica “Spring Initializr” para la creación de proyectos. Esto hizo que fuera muy sencillo y conveniente. La interfaz me permitió seleccionar las dependencias necesarias de manera rápida y sin complicaciones. Esto es algo que no había visto antes ya que hasta este momento de mi experiencia en programación nunca había creado un proyecto con características específicas desde la web. Además, puedo destacar una vez más las grandes capacidades de visual studio code ya que a través de la instalación de extensiones pude trabajar con java y Maven sin ningún inconveniente.
- En mi opinión es fundamental para que haya más miembros de la comunidad en un sistema o framework la utilización de guías prácticas y métodos de enseñanza. En este caso la guía práctica de Spring proporcionada fue muy útil para comprender los conceptos y pasos necesarios para habilitar el caché de datos, es fácil e intuitiva y permite la implementación rápida. Esto me pareció un gran logro ya que en ocasiones las guías son muy complejas y largas de implementar.
- Respecto al resultado de la práctica. Pude observar el impacto positivo en el rendimiento al habilitar el caché de datos. Las consultas de datos recurrentes son procesadas mucho más rápido, lo que en un ambiente de producción considero que aumentara la calidad en la experiencia del usuario. En el caso de un sistema web la UX es fundamental, por ende, considero que implementar cache es una fortaleza al momento de programarlo.
- Referente al framework Spring pude ver como se basa en la simplicidad y la productividad, siendo lo que más me impactó. A pesar de ser un framework completo y robusto, encontré que la curva de aprendizaje era bastante manejable, especialmente con la ayuda de la documentación oficial y otros

recursos en línea. La estructura fue clara y coherente, además que pude entender específicamente sus características y funcionalidades.

8. ANEXOS

Link Github del proyecto: [CamilaACT/CacheSpring \(github.com\)](https://github.com/CamilaACT/CacheSpring)

9. BIBLIOGRAFÍA

AWS. (s.f.). AWS. Obtenido de Comparación de Redis y Memcached:

<https://aws.amazon.com/es/elasticache/redis-vs-memcached/>

Borges, S. (6 de Diciembre de 2023). *infranetworking*. Obtenido de Memcached vs Redis: ¿cuál es

mejor?: <https://blog.infranetworking.com/memcached-vs-redis-cual-es-mejor/>

Digitla Guide. (31 de Marzo de 2021). *Digital Guide IONOS*. Obtenido de Memcached vs. Redis: comparativa de las bases de datos en memoria:

<https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/memcached-vs-redis/>

IACARO. (4 de Febrero de 2023). *IACARO.COM*. Obtenido de Redis o Memcached ¿Cuál es mejor?:

<https://iacaro.com/redis-o-memcached-cual-es-mejor/>