INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Desarrollo de Sistemas Distribuidos

Práctica 6 ‘Servicios Web’

Docente:

Carreto Arellano Chadwick

Grupo:

7CM6

Alumnos:

Rodríguez Hernández Marlene Guadalupe

Fecha: jueves 16, octubre 2025

Antecedentes

Los servicios web surgieron ante una necesidad de estandarizar la comunicación entre distintas plataformas y lenguajes de programación

Al principio se crearon los estándares DCOM y CORBA, los cuales no tuvieron mucho éxito, no es sino en 1999 cuando comenzó a plantearse un nuevo estándar, el cual usa hoy en día XML, SOAP, WSDL y UDDI.

Con base en estas definiciones, podemos entender que por la World Wide Web Consortium (W3C) un servicio web es un sistema de software designado para soportar la interacción de interoperabilidad de una máquina a otra mediante una red.

Así mismo podemos definirla como un servicio implementado específicamente en una representación como un conjunto de clientes.

Hoy en día se derivan distintos servidores como:

- Servidor iterativo, el servidor que manipula la petición, y en caso de ser necesario devuelve una respuesta a la petición del cliente.
- Servidor concurrente, este no puede manipular la petición por sí mismo, pero la pasa mediante un hilo separado a otro proceso.
    - Servidor multihilos

Sin embargo, estos servidores tienen algo en común, los clientes hacen contexto con el servidor enviando peticiones a un punto final, el cual también es llamado puerto, localizado en el servidor, entonces cada servidor atiende un puerto específico

Planteamiento del problema

En los sistemas distribuidos actuales, es común que los sistemas se comuniquen entre ellos y sea posible compartir datos en tiempo real, cosas que damos por hecho como enviar un mensaje de texto compartir un reel, sin embargo, establecer esta comunicación toma su tiempo y su manera tiene que ser en forma estándar para que los datos viajen entre aplicaciones de manera segura y sin errores, llegando así todos los datos de la misma forma en que han sido enviados en un principio y no lleguen a la mitad o más incompletos incluso

Propuesta de solución

Esta práctica surge para poder comprender la interacción entre un sistema y un servidor, por lo que desarrollamos una aplicación y un cliente (o en este caso seguimos manejando a las tortugas), para demostrar que hemos aprendido a crear un servicio web que permite enviar datos desde el servidor y que estos lleguen de la mejor forma

al cliente. La solución consiste en desarrollar un servicio web RESTful más Spring Boot, utilizando HTTP para la comunicación, mientras los clientes (tortugas) hacen peticiones HTTP (POST, GET) al servidor (la carrera).

Desarrollo de la solución

En la siguiente tabla presento una breve estructura de los archivos y su funcionamiento dentro del proyecto

SERVICIOS-WEB/	
├── backend/	REST con Spring Boot
│ ├── src/main/java/com/ejemplo/carrera/	
│ │ ├── controller/CarreraController.java	Capa de presentación (API REST)
│ │ ├── model/Tortuga.java	Entidad o
│ │ ├── service/CarreraService.java	Logica de la carrera
│ │ └── CarreraTortugasApplication.java	Clase principal de Spring Boot
│ └── pom.xml	Configuracion Maven
└── cliente/	API Java que consume el servicio
├── ClienteTortuga.java	Programa principal del cliente
├── CarreraAPI.java	Clase que hace las peticiones
└── Tortuga.java	Modelo usado en el cliente

- Backend (Servidor de la carrera con Spring Boot)
    - CarreraService.java
        - aqui esta la lógica principal de la carrera, declaramos que la meta son cien pasos, con el Map tortugas almacenamos las tortugas registradas (su id y su posición) y el AtomicReference, lo utilizamos con el idGanador para guardar el id de la tortuga ganadora una vez que alguna llega a la meta, este es seguro de usar para múltiples hilos (threads)

```
public class CarreraService {
    private static final int META = 100;
    private final Map<String, Tortuga> tortugas = new ConcurrentHashMap<>();
    private final AtomicReference<String> idGanador = new AtomicReference<>(
```

- el método registrarTortuga()
  - crea un ID único (UUID), y una tortuga con ese id, la agrega al mapa tortugas y devuelve el id para que el cliente Tortuga lo use

```
// Registra una nueva tortuga y devuelve su ID
public synchronized String registrarTortuga() {
    String id = UUID.randomUUID().toString();
    tortugas.put(id, new Tortuga(id));
    System.out.println("Registrada tortuga con ID: " + id);
    return id;
}
```

- el método avanzar(String id)
  - busca a la tortuga por su id, y sino existe devuelve error

```
if (idGanador.get() != null) {
    respuesta.put("estado", "terminado");
    respuesta.put("ganador", idGanador.get());
    respuesta.put("posicion", t.getPosicion());
    return respuesta;
}
```

  - si ya hay un ganador, se avisa que la carrera terminó y se indica al ganador

```
t.avanzar();
respuesta.put("posicion", t.getPosicion());
```

  - las tortugas se mueven usando el método avanzar(), y se guarda la nueva posición en la respuesta

```
if (t.getPosicion() >= META && idGanador.compareAndSet(null, id))
    respuesta.put("estado", "ganador");
    System.out.println("La tortuga " + id + " ha ganado la carrer
} else {
    respuesta.put("estado", "en_carrera");
}
```

- finalmente, si alguna tortuga llega o pasa la meta (los cien pasos) se marca como ganadora, sino su estado siendo ‘en carrera’

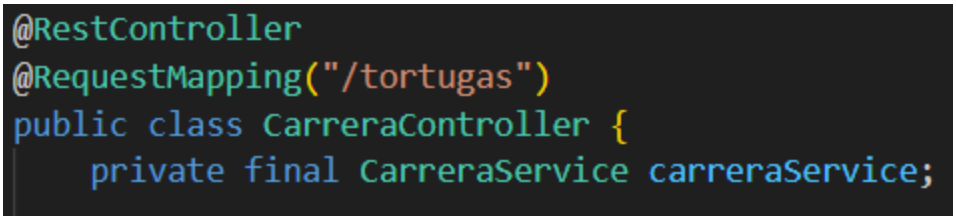
- el método getEstadoCarrera()
  - devuelve todas las posiciones actuales

```
// Devuelve el estado completo de la carrera (ID de tortuga y posic
public Map<String, Integer> getEstadoCarrera() {
    Map<String, Integer> estado = new LinkedHashMap<>();
    for (Tortuga t : tortugas.values()) {
        estado.put(t.getId(), t.getPosicion());
    }
    return estado;
}
```

- el método reiniciarCarrera()
  - limpia todo para iniciar una nueva carrera

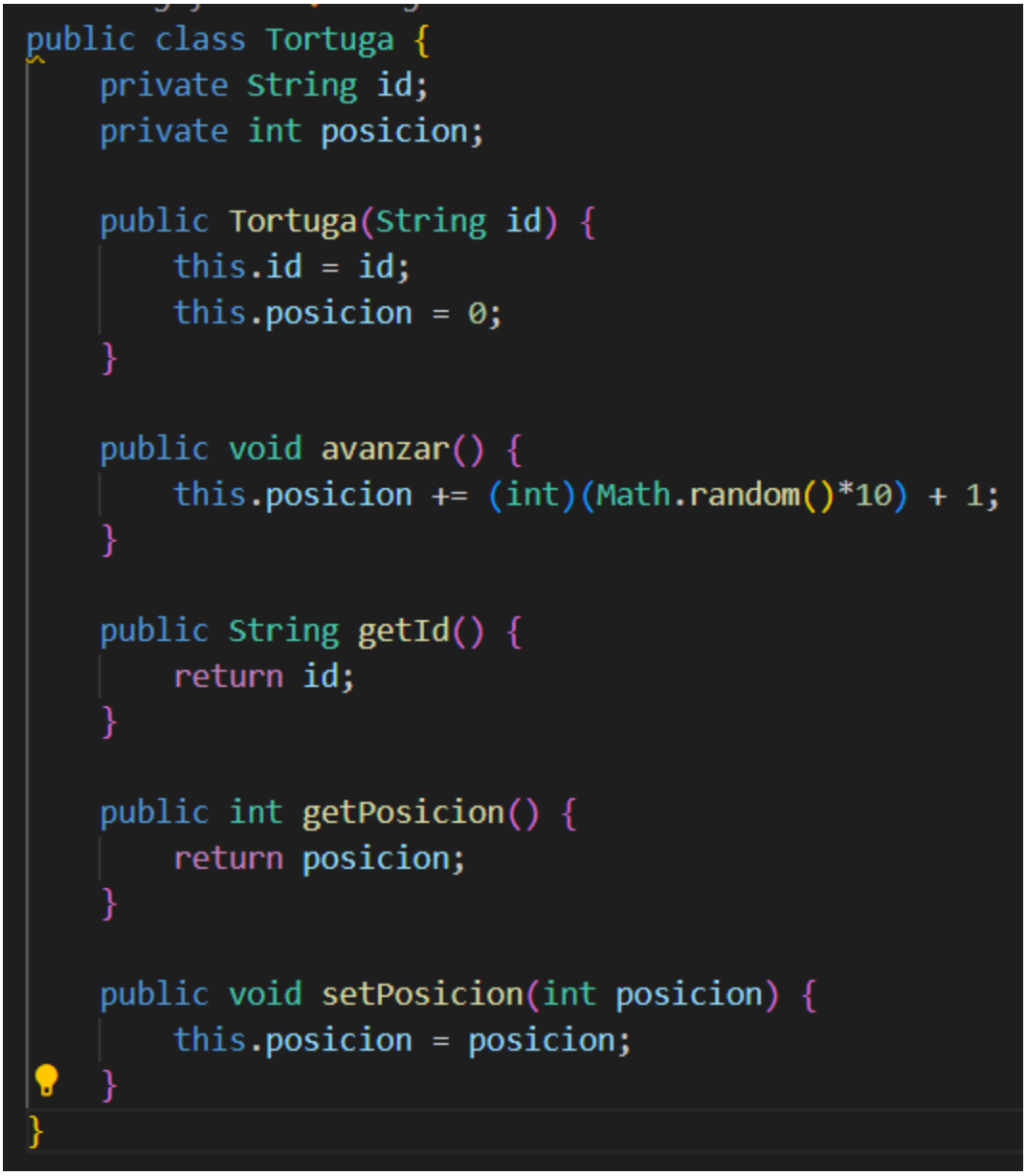
```
// Reinicia la carrera
public void reiniciarCarrera() {
    tortugas.clear();
    idGanador.set(null);
    System.out.println("Carrera reiniciada!");
}
```

- CarreraController.java
  - aquí defini los endpoints HTTP del servicio



- cada método del controlador responde a una URL
  - Registrar tortuga (metodo POST), con la ruta /tortugas -develve un id nuevo-
  - Avanzar (método POST), con la ruta /tortugas/(id)/avanzar -avanza la tortuga y devuelve JSON-
  - Estado carrera (método GET), con la ruta /tortugas/estado -devuelve las posiciones de todas las tortugas-
  - Ganador (método GET), con la ruta /tortugas/ganador -devuelve el id de la tortuga ganadora-
  - Reiniciar (método POST), con la ruta /tortugas/reiniciar -limpia la carrera-

- Tortuga.java (modelo)
  - aqui ponemos los datos de cada tortuga



```java
public class Tortuga {
    private String id;
    private int posicion;

    public Tortuga(String id) {
        this.id = id;
        this.posicion = 0;
    }

    public void avanzar() {
        this.posicion += (int)(Math.random()*10) + 1;
    }

    public String getId() {
        return id;
    }

    public int getPosicion() {
        return posicion;
    }

    public void setPosicion(int posicion) {
        this.posicion = posicion;
    }
}
```

        - así cada vez que una tortuga avanza, su posición aumenta un numero aleatorio entre uno y diez (lo que hemos estado haciendo desde la primer práctica)
- Cliente (Tortuga, es quien consume el servicio)
    - CarreraAPI.java
        - en este archivo se consume el servicio REST, usamos el HttpURLConnection para enviar peticiones y Jackson (ObjectMapper) para interpretar las respuestas JSON

```java
public static Map<String, Object> avanzarTortuga(String id) throws Exception {
    URL url = new URL(BASE_URL + "/" + id + "/avanzar");
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setRequestMethod(method:"POST");
    conn.setDoOutput(dooutput:true);
    conn.setRequestProperty(key:"Content-Type", value:"application/json");

    BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
    String json = in.readLine();
    in.close();

    // Convertir JSON a Map<String,Object>
    Map<String, Object> resultado = gson.fromJson(json, new TypeToken<Map<String, Object>>() {}.get
    return resultado;
}
```

- ClienteTortuga.java
  - este es el programa principal que simula una tortuga real participando en la carrera, hace:
    - registra a la tortuga (POST /tortugas)
    - guarda su id
    - cada segundo llama a /tortugas/{id}/avanzar
    - muestra la posición y el estado (en_carrera, ganador, terminado)
    - se detiene cuando alguien gana

La comunicación entre el cliente <-> servidor funciona de la siguiente forma:

1. el cliente envía peticiones HTTP (POST o GET) al backend
2. el backend responde con datos JSON
3. el cliente interpreta ese JSON usando ObjectMapper y lo muestra por consola

Finalmente para ver el funcionamiento de este proyecto, primero levantamos el servidor, con Spring Boot (mvn spring-boot:run)

```
PS C:\Users\kirit\Downloads\ESCOM\DSD-R\Servicios-Web\backend> mvn spring-boot:run
```

en el cual tenemos rutas como:

- POST /registrar -> registrar tortuga
- POST /avanzar/{id} -> avanzar una tortuga
- GET /estado -> obtener posiciones de todas las tortugas
- GET /ganador -> obtener al ganador
- POST /tortugas/reiniciar -> reinicia la carrera

mientras que del lado del cliente compilamos (javac *.java) y ejecutamos (java ClienteTortuga) y el cliente procesa mostrando en consola el movimiento de la tortuga que está corriendo en esa terminal

```
C:\Users\kirit\Downloads\ESCOM\DSD-R\Servicios-Web\cliente>java -cp ".;libs/*" Client
eTortuga
Tortuga registrada con ID: ed10390a-4bb8-4a4c-b0e8-09e226c7bca5
Esperando iniciar la carrera...
Tortuga avanza a posicion: 5 (en_carrera)
Tortuga avanza a posicion: 10 (en_carrera)
Tortuga avanza a posicion: 13 (en_carrera)
Tortuga avanza a posicion: 17 (en_carrera)
Tortuga avanza a posicion: 21 (en_carrera)
Tortuga avanza a posicion: 25 (en_carrera)
Tortuga avanza a posicion: 34 (en_carrera)
Tortuga avanza a posicion: 41 (en_carrera)
Tortuga avanza a posicion: 49 (en_carrera)
Tortuga avanza a posicion: 53 (en_carrera)
Tortuga avanza a posicion: 62 (en_carrera)
Tortuga avanza a posicion: 69 (en_carrera)
Tortuga avanza a posicion: 74 (en_carrera)
Tortuga avanza a posicion: 79 (en_carrera)
Tortuga avanza a posicion: 88 (en_carrera)
Tortuga avanza a posicion: 90 (en_carrera)
Tortuga avanza a posicion: 100 (ganador)
Has ganado la carrera!
Carrera finalizada para esta tortuga.
```

el cual se conecta al backend, se registra una tortuga con un id específico y comienza a avanzar, así mismo consulta con el backend para verificar si ya hubo algún ganador y detenerse al finalizar la carrera ya sea si gana o pierde.

```
:\Users\kirit\Downloads\ESCOM\DSD-R\Servicios-Web\backend\target\classes started by ImNayeon in C:\Users\kirit\Downloads\ESCOM\DSD-R\Servicios-Web\backend)
at.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2025-10-16T13:55:15.842-06:00  INFO 16924 --- [           main] c.e.carrera.CarreraTortugasApplication   : Started CarreraTortugasApplication in 1.955 seconds (process running fo
r 2.384)
2025-10-16T13:55:24.420-06:00  INFO 16924 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-10-16T13:55:24.422-06:00  INFO 16924 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2025-10-16T13:55:24.423-06:00  INFO 16924 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
Registrada tortuga con ID: ed10390a-4bb8-4a4c-b0e8-09e226c7bca5
2025-10-16T13:55:24.501-06:00  WARN 16924 --- [nio-8080-exec-2] ocalVariableTableParameterNameDiscoverer : Using deprecated '-debug' fallback for parameter name resolution. Compi
le the affected code with '-parameters' instead or avoid its introspection: com.ejemplo.carrera.controller.CarreraController
La tortuga ed10390a-4bb8-4a4c-b0e8-09e226c7bca5 ha ganado la carrera!
```

Resultados / Conclusiones

Durante el desarrollo de esta práctica, tomando en cuenta lo que ya se ha trabajado con prácticas anteriores e implementando la mejora con los servicios web, hemos aprendido las bases fundamentales de un Web Server, como fueron sus inicios, su funcionamiento, el cual he comprendido mejor gracias a esta práctica, en la cual al conectarse o unirse las tortugas a la carrera estas van avanzando al igual que al

principio en un hilo distinto pero ahora su comunicación ha cambiado y a su vez se sigue manteniendo el manejo de concurrencia básica en la simulación de la carrera, esto a través de los procesos distribuidos con HTTP y el consumo de servicios web desde el cliente java usando el HTTpURLConnection lo cual fue más fácil de implementar haciendo uso del spring boot para construir APIs RESTful como un ejemplo simple comprendiendo mejor todos estos conceptos.

Creo que lo que mas me impresiono es como se tiene que ir acomodando todo en carpetas, esto para que tenga una mejor estructura, se sepa dividir la información y se pueda interpretar de una mejor manera, aplicando así los principios básicos tanto de la estructura de cada proyecto como de tener buenas prácticas y código limpio.

Enlace al repositorio de GitHub donde está la práctica

https://github.com/Marlene0807/Sistemas-Distribuidos.git

Referencias

Historia de los Web Services