**FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO**

|  |  |
| --- | --- |
| PROGRAMA DE FORMACIÓN | ADSO |

|  |  |  |  |
| --- | --- | --- | --- |
| COMPETENCIA | 220501096. Desarrollar la solución de *software* de acuerdo con el diseño y metodologías de desarrollo. | RESULTADOS DE APRENDIZAJE | 20501096-04. Codificar el *software* de acuerdo con el diseño establecido. |

|  |  |
| --- | --- |
| NÚMERO DEL COMPONENTE FORMATIVO | 20 |
| NOMBRE DEL COMPONENTE FORMATIVO | Desarrollo de aplicaciones web con Java |
| BREVE DESCRIPCIÓN | El componente proporciona una guía sobre tecnologías y *frameworks* en el ecosistema Java. Cubre Java con JDBC para la conectividad de bases de datos, Servlets y JSP para el desarrollo web, React JS para interfaces de usuario, Hibernate para ORM, y Spring Framework para aplicaciones empresariales, destacando la inyección de dependencias y la configuración simplificada con Spring Boot. |
| PALABRAS CLAVE | JDBC, Servlets, Hibernate, Spring, React JS |

|  |  |
| --- | --- |
| ÁREA OCUPACIONAL | 2 - CIENCIAS NATURALES, APLICADAS Y RELACIONADAS |
| IDIOMA | Español |

1. **TABLA DE CONTENIDOS:**

1. Java con JDBC

1.1. Clases e interfaces Java

1.2. Conexión a base de datos desde aplicación Java

2. *Servlets* y JSP

2.1. Introducción a *servlets*

2.2. Formulario HTML con *Servlets*

2.3. HTTP *Session*

2.4. Elementos de JSP

2.5. MVC

3. Fundamentos de React JS

3.1. Instalando React JS

3.2. Crear un proyecto

3.3. Propiedades y estados de los componentes

4. Hibernate

4.1. Operaciones básicas

4.2. Asociaciones

5. Spring

5.1. Introducción

5.2. Spring Data

5.3. Spring Boot y MySQL

5.4. Validaciones

1. **INTRODUCCIÓN**

En el ámbito del desarrollo de *software,* Java se ha consolidado como uno de los lenguajes de programación más robustos y versátiles. Este documento proporciona una guía comprensiva sobre algunas de las tecnologías y *frameworks* más importantes utilizados en el ecosistema Java, con un enfoque particular en la conectividad de bases de datos, el desarrollo web y la gestión de la persistencia de datos.

El contenido de este documento se estructura en cinco secciones principales:

1. **DESARROLLO DE CONTENIDOS:**
2. **Java con JDBC**

Cuando se habla de JDBC (*Java Database Connectivity),* se hace referencia a un *driver* que encapsula un conjunto de clases e interfaces escritas en Java, creadas para establecer conexiones, enviar sentencias SQL y procesar los datos resultantes sobre bases de datos relacionales.

Java con JDBC

aplicaciones java y jdbc (equipo del quiente) = Sistema gestior de bases de datos relacional (equipo servidor de bd)


En términos más generales, JDBC es uno de los puentes que puede ser utilizado en Java para comunicar una aplicación Java con un servidor de bases de datos relacionales. Trabajar con conexiones a bases de datos mediante JDBC implica realizar cuidadosamente las siguientes acciones:

La siguiente tabla muestra las direcciones oficiales para la descarga del *driver* JDBC para las bases de datos relacionales más comúnmente utilizadas en la industria del *software.*

**Tabla 1.** *Drivers* JDBC para SGBD

|  |  |
| --- | --- |
| Base de Datos | Página oficial para descarga de JDBC |
| **Mysql** | <https://dev.mysql.com/downloads/connector/j/> |
| **SQL Server** | <https://docs.microsoft.com/en-us/sql/connect/jdbc/download-microsoft-jdbc-driver-for-sql-server?view=sql-server-ver15> |
| **Oracle** | <https://www.oracle.com/database/technologies/jdbc-upc-downloads.html> |
| **PostgreSQL** | <https://jdbc.postgresql.org/download.html> |
| **DB2** | <https://www.ibm.com/support/pages/db2-jdbc-driver-versions-and-downloads> |

* 1. **Clases e interfaces Java**

Además de disponer del *driver* JDBC correspondiente para la base de datos que se desea conectar, se requiere un conjunto adicional de clases e interfaces Java para hacer uso del *driver.* Para ello, es necesario importar los paquetes java.sql o javax.sql. A continuación, se describen algunas de las clases más importantes dentro de estas librerías, y otros elementos, para gestionar correctamente una base de datos por medio de JDBC.

|  |
| --- |
| Acordeón  CF020\_1.1\_Clases e interfaces Java |

|  |  |
| --- | --- |
|  | **Clases e interfaces Java**  Para profundizar y conocer los elementos más importantes de las clases e interfaces Java más usadas, se debe visitar el documento **Clases e interfaces Java**, disponible en el siguiente enlace. |

* 1. **Conexión a base de datos desde aplicación Java**

Garantizar la conexión a una base de datos requiere la aplicación concreta de los pasos ya explicados.

Los pasos que deben realizarse para garantizar una conexión a base de datos desde una aplicación Java, utilizando JDBC, son:

Para afianzar los conocimientos logrados hasta este punto, se recomienda revisar el video propuesto a continuación. En este, se presenta un ejemplo sencillo y se describen las prácticas de cada paso descrito.

1. Conexión JDBC

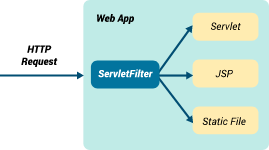
A screenshot of a computer

Description automatically generated

1. ***Servlets* y JSP**

Los *Servlets* y los JSP (*Java Server Pages*), son tecnologías que se utilizan para realizar ejecuciones en servidores Java, permitiendo extender los servicios de un servidor web. (Berzal, 2007)

Java con JDBC



Un ejemplo sencillo de lo que se puede hacer con JSP y *Servlet* es que, por medio de una página JSP, se puede crear un formulario que contenga una serie de datos como: primer nombre, segundo nombre, edad y correo. Luego, a través de un *servlet,* se realiza el envío de estos registros para que sean almacenados en una base de datos y, finalmente, el usuario pueda acceder a esta información.

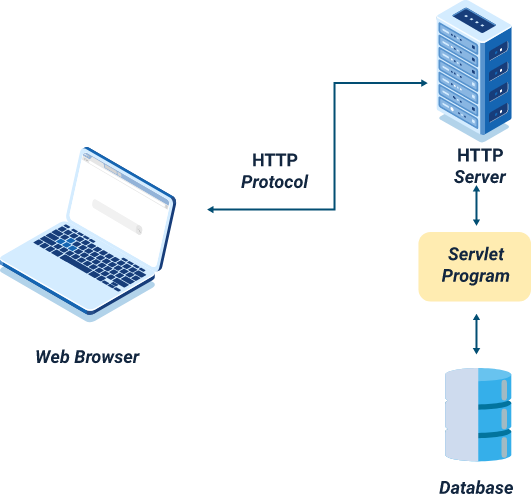
Conozca, a continuación, los procesos principales de estas tecnologías y algunos aspectos generales del uso de JSP y *Servlet:*

|  |
| --- |
| SLIDES  CF020­\_2­\_Servlets y JSP (1) |

* 1. **Introducción a *servlets***

*Servlet* es un programa que se lleva a cabo en una aplicación o servidor web; se emplea como una capa que intercede entre la base de datos y el servidor de peticiones HTTP, por medio de un navegador web o cualquier otro cliente HTTP (Gómez, 2017).

Arquitectura de un *servlet*



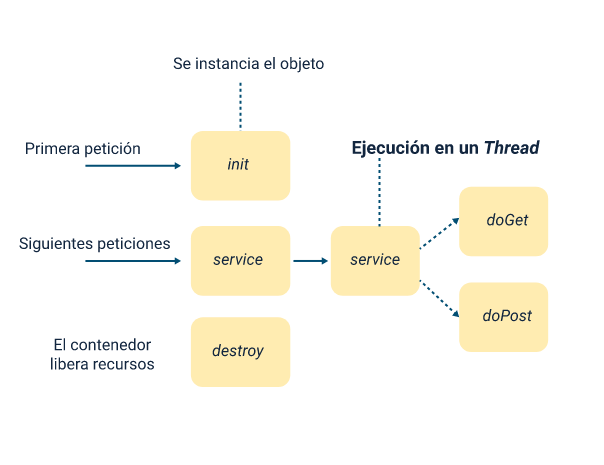
A continuación, se presentan algunas ventajas de utilizar *servlets:*

Las tareas principales de un *servlet* son:

**Ciclo de vida de un *Servlet***

A diferencia de las clases en Java, que inician con un método main(), en los *servlets* se invocan ciertos métodos al recibir peticiones. A esta metodología se le llama ciclo de vida de un *servlet* y se compone de tres métodos: *init, service* y *destroy.*

Ciclo de vida de un *Servlets*



Conocer el ciclo de vida de un *Servlet;* procurar llevar registro, en la libreta personal de apuntes, de los aspectos más importantes y destacados.

|  |
| --- |
| TARJETAS  CF020­\_ 2.1\_Introducción a servlets |

**Ejemplo de *Servlet***

El siguiente código presenta un "Hola Mundo" en un *servlet,* con sus métodos de ciclo de vida.

|  |
| --- |
| 1. public class HolaMundo extends HttpServlet {  2.  private String mensaje;  **3.** public void init () throws ServletException  4. {  // inicia la asignación de mensaje hola mundo  5.    mensaje = “Hola Mundo”;  }  6.       public void doGet (HttpServletRequest request,HttpServletResponse response)       throws ServletException, IOException                {   // Se especifica el formato de la respuesta utilizando método getWriter  7.                      salida = response.getWriter ();                    // Genera respuesta de la petición  8.         salida.println("<h1>" + mensaje + "</ h1>");}  9.public void destroy ()  10.            {.} |

En el código anterior, se hace lo siguiente:

Explore el vídeo que se propone para ahondar en los aspectos más importantes de Servet.

**Video.** *Servet*

A screenshot of a computer

Description automatically generated

* 1. **Formularios HTML con *Servlets***

Las páginas web contienen formularios HTML para la solicitud de información al usuario. El proceso de recolección de datos y respuesta de este formulario se debe realizar por medio de un programa. Esto lo hace un *servlet* que recoge las peticiones de un formulario y las procesa.

La declaración de un formulario HTML se hace por medio de la etiqueta **<form>**, donde es primordial que se especifique el método por el que se realizará el envío de los datos, los cuales pueden ser **get** o **post.**

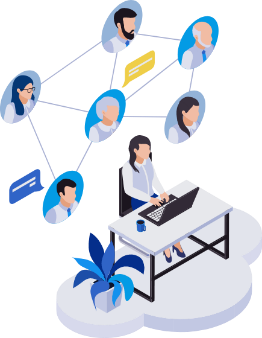
A continuación, se presentan los aspectos más relevantes en lo relativo a formularios HTML con *servlets:*

**Video.** Formulario *Servlet*

A screenshot of a computer

Description automatically generated

**2.3. HTTP *Session***

Las sesiones se crean para hacer un seguimiento y memorizar las acciones de un usuario en un sitio web. Las sesiones en Java permiten guardar información entre las diversas peticiones HTTP, utilizando una clase que realiza el seguimiento de las sesiones; estos objetos también se almacenan del lado del servidor.

Para poder hacer uso de este objeto, es necesario obtener un objeto de la sesión para poder escribir en él, eliminarlo o leerlo, incluso después de que la sesión haya sido cerrada.

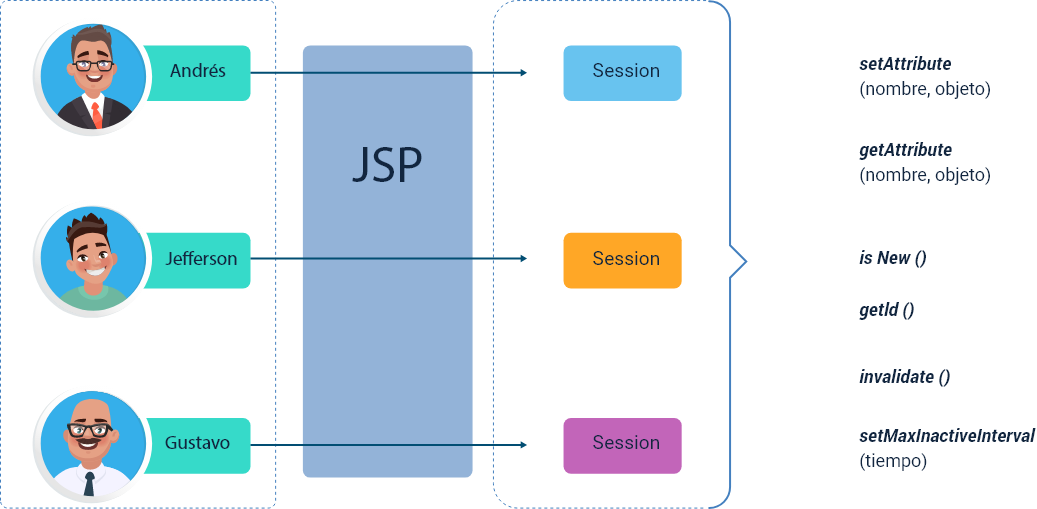
Los métodos más utilizados de HttpSession son:

|  |
| --- |
| Pestañas  CF020­\_2.3\_HTTP Session |

|  |  |
| --- | --- |
| **Tenga en cuenta:**  **A yellow circle with a black exclamation mark  Description automatically generated** | El método utilizado para obtener la sesión de un usuario es **getSession ()** este método regresa una interfaz de tipo **HttpSession**, Esta interfaz está dentro del paquete javax. **servlet.http** y es usada por el contenedor de páginas JSP que permite la creación de una sesión entre el cliente y el servidor.  <%**HttpSession** variable =**request.getSession();** %> |

A continuación, se puede conocer el funcionamiento básico de las sesiones:

HTTP Session



* 1. **Elementos de JSP**

En JSP se encuentran tres tipos de elementos que pueden ser insertados en una página web: el código, las directivas y las acciones. Cada uno de ellos favorece las distintas ejecuciones que se requieren en la construcción de aplicaciones con Java.

|  |
| --- |
| ACORDEÓN  CF020\_2.4.\_Elementos de JSP |

Para afianzar sus conocimientos en elementos JSP, le invitamos a revisar el siguiente vídeo.

**Video.** Tipos De Codigo JSP



<https://www.youtube.com/watch?v=1TNloswji-4>

* 1. **MVC**

El MVC (*Modelo-Vista-Controlador)* es un patrón de diseño estándar que se utiliza para crear aplicaciones dinámicas con Java. Consiste en separar la parte lógica de los proyectos de la parte visual, integrando *servlets y* JSP. En este patrón, los *servlets* se encargan de la parte lógica y los JSP de la parte visual.

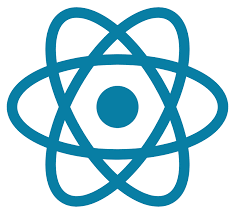
División de la aplicación realizada por el patrón:

Arquitectura MVC

A diagram of a process

Description automatically generated

1. **Fundamentos de React JS**



React JS es una nueva tecnología importante para la construcción de *software* en su componente Front-end, que requiere tener en cuenta algunas consideraciones, como se pueden conocer a continuación:

|  |
| --- |
| PESTAÑAS  CF020\_ 3\_ Fundamentos de React JS |

**¿Qué es React JS?**

Es una biblioteca JavaScript de código abierto, desarrollada por la multinacional Facebook para crear interfaces de usuario. Es importante tener en cuenta que una biblioteca o librería es un conjunto de funcionalidades ya construidas y reutilizables siguiendo su sintaxis. Su principal objetivo, además de facilitar el desarrollo de aplicaciones de una sola página, es ser sencilla, declarativa y fácil de combinar.

|  |  |
| --- | --- |
| A yellow triangle with a purple exclamation mark  Description automatically generated | **¡Importante!** Los datos deben ser atractivos, entendibles y accesibles para el usuario. Para esto, se deben tener en cuenta los objetivos de alistamiento y preparación de la información. |

**¿Qué es JSX?**

Su nombre viene de la composición de JavaScript (JS) + XML = (JSX) y corresponde a la sintaxis propuesta en React JS para escribir elementos de interfaz web compuestos por JavaScript + HTML, donde el HTML sigue las normas de código bien formado XML.

**3.1. Instalando React JS**

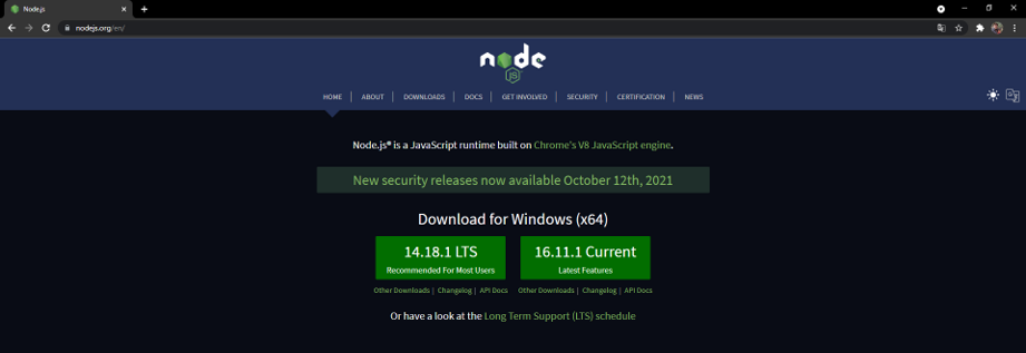
Las siguientes instrucciones presentan el proceso de instalación para sistemas operativos Linux versión Ubuntu 20.04.3 LTS y Windows. En ambos casos, primero se necesita instalar el servidor denominado Node y el manejador de paquetes npm.

|  |
| --- |
| **PASOS**  **CF020\_3.1\_Instalando React JS** |

A continuación siga los siguientes pasos de Instalación sobre sistema operativo Windows.

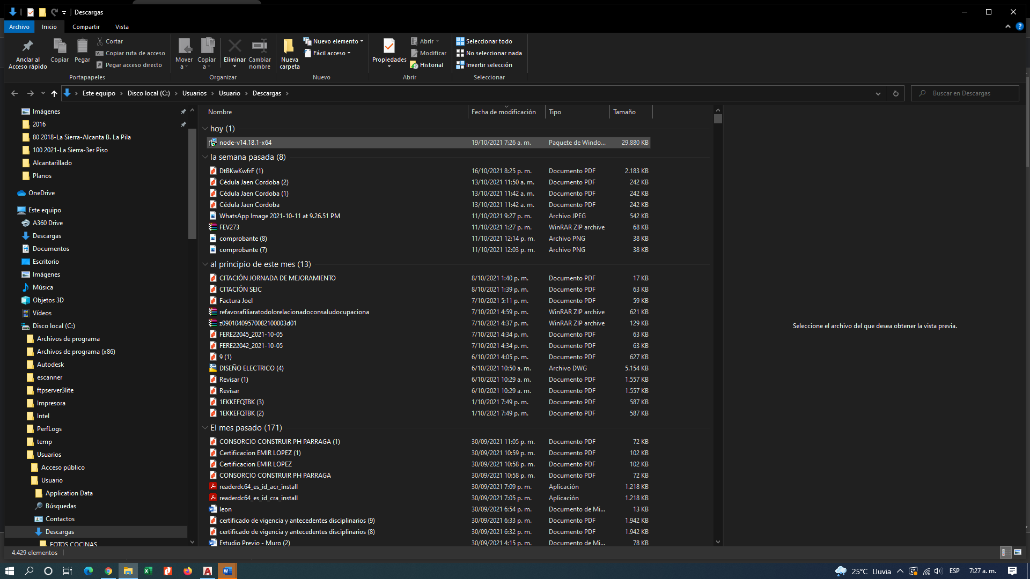
**Paso 1.** Ingrese a Node >= 14.0.0 and npm >= 5.6 que corresponde a la página oficial de nodejs <https://nodejs.org/en/> y descargue la versión para windows de 64 bits.

Paso 1



**Paso 2.** Descargue en el sitio de su preferencia

Paso 2



**Paso 3.** Ejecute el archivo descargado como administrador de Windows y continúe el *wizard* haciendo clic en ***next*** en cada pantalla.

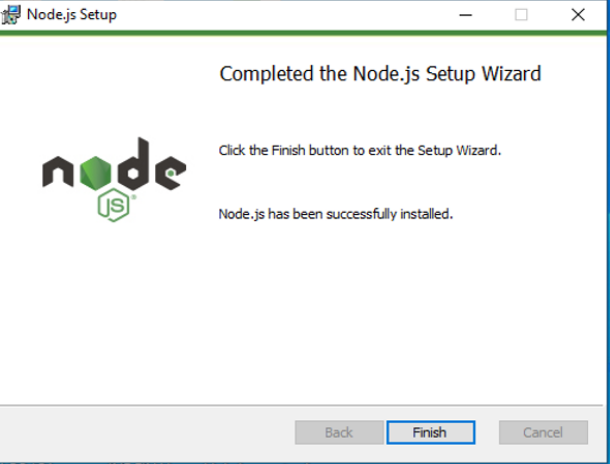
Paso 3

A screenshot of a computer

Description automatically generated

**Paso 4.** Por último de clic en *finish* para terminar la instalación.

Paso 4



**3.2. Crear un proyecto**

Una vez llevado a cabo el proceso de instalación del paso anterior de node y npm, se puede crear el proyecto React e ir ejecutando los siguientes comandos:

1. npx create-react-app my-app
2. cd my-app
3. npm start

Iniciar la creación del proyecto, a través de la siguiente secuencia:

|  |
| --- |
| Pasos  CF020\_3.2\_Crear un proyecto |

**3.3. Propiedades y estados de los componentes**

Un componente en React hace referencia a la posibilidad de definir de manera independiente una funcionalidad que puede ser reutilizable dentro del sistema. Esencialmente, estas funcionalidades se escriben como funciones JavaScript que reciben entradas denominadas propiedades y devuelven una forma de visualización.

Ejemplo de un componente denominado "Bienvenido" que recibe una propiedad y devuelve un párrafo de saludo.

Propiedades

A screenshot of a computer

Description automatically generated

Además, a un componente se le puede aplicar el concepto de estado en donde este, se define como la capacidad de especificar variables que pueden cambiar su valor en el tiempo, convirtiéndose en un comportamiento interno del componente, estos componentes se definen con el nombre de clase con la palabra reservada *class.*

|  |
| --- |
| class Clock extends React.Component {  constructor(props) {  super(props);  this.state = { date: new Date() };  }  render() {  return (  <div>  <h1>Hola, Mundo!</h1>  <h2>Es {this.state.date.toLocaleTimeString()}.</h2>  </div>  );  }  } |

**Eventos en React**

Los eventos en React se manejan semejante a los del DOM se le pasa una función al controlador del evento.

|  |
| --- |
| function Bienvenido(propiedad) {  return <p>Saludos, {propiedad.nombre}</p>;  }  function activarCamaras(propiedad) {  return <p>Camara Uno, {propiedad.nombre}</p>;  }  <button onClick={activarCamaras}>  Activar Camaras  </button>  class Clock extends React.Component {  constructor(props) {  super(props);  this.state = { date: new Date() };  }  render() {  return (  <div>  <h1>Hola, Mundo!</h1>  <h2>Es {this.state.date.toLocaleTimeString()}.</h2>  </div>  );  }  } |

**Listas y *Keys***

Las listas hacen referencia a un arreglo de datos, que puede ser un arreglo de objetos o de datos simples como caracteres, números, etc. Como toda lista en programación, React proporciona un método para extraer estos datos y mostrarlos en la página en la cual se está trabajando. A continuación, se presenta cómo se hace:

|  |  |
| --- | --- |
| En React es necesario pasar arreglos de datos a listas para el manejo de algunos componentes lo cual se trabaja con la función map de la siguiente manera en donde se devuelve un elemento <li> por cada número. | const numbers = [1, 2, 3, 4, 5];  const listItems = numbers.map((number) =>  <li>{number}</li>  ); |
| Adicional al manejo de listas es muy importante tener en cuenta el identificador único de componentes para evitar errores en el DOM por lo tanto es necesario manejar el atributo *key.* | function NumberList(props) {  const numbers = props.numbers;  const listItems = numbers.map((number) =>  <li key={number.toString()}>  {number}  </li>  );  return (  <ul>{listItems}</ul>  );  } |

**Formularios**

En React el envío de formularios se comporta como en HTML, solo que aquí, el ingreso y cambio de valores en las entradas se deben manejar con componentes controlados por medio de estados.

|  |
| --- |
| class Clock extends React.Component {  constructor(props) {  super(props);  this.state = { date: new Date() };  }  render() {  return (  <div>  <h1>Hola, Mundo!</h1>  <h2>Es {this.state.date.toLocaleTimeString()}.</h2>  </div>  );  }  } |

**Concepto de clases en React**

Las clases son otra forma de definir un componente en React. Se caracterizan por tener propiedades y, a diferencia de las funciones, pueden definir estados propios sin utilizar el concepto de *Hook.* Dentro de los componentes de clase, existe una función constructora que se utiliza para definir el estado inicial del componente. Es muy importante llamar a la función **super()** dentro del constructor para que los parámetros funcionen adecuadamente, como puede verse en la siguiente imagen:

|  |
| --- |
| class Counter extends React.Component {  constructor(props) {  super(props);  this.state = { foo: 123, bar: 456 };  }  render() {  return (  <div>foo: {this.state.foo} bar: {this.state.bar}</div>  );  }  } |

Explorar el siguiente recurso y conocer los aspectos más relevantes relacionados con el concepto de clases en React:

|  |
| --- |
| PASOS  CF020\_3.3\_Propiedades y estados de los componentes |

**Funciones *Hooks***

Las funciones *Hooks* permiten a los componentes definidos por medio de funciones manejar variables de estado locales, similar a cómo lo hacen las clases. Los *Hooks* son funciones JavaScript y están sujetas a dos reglas:

Ejemplo de una función que utiliza el *Hook* **useState**:

|  |
| --- |
| import React, { useState } from 'react';  function Example() {  // Declara una nueva variable de estado, que llamaremos "count"  const [count, setCount] = useState(0);  return (  <div>  <p>You clicked {count} times</p>  <button onClick={() => setCount(count + 1)}>  Click me  </button>  </div>  );  } |

## **Ciclos de vida de los componentes de React**

Hace referencia a los métodos que se pueden utilizar por defecto (es decir, que forman parte de la librería de React). Cuando se ejecuta o se renderiza un componente en la línea de tiempo de ejecución del programa, estos métodos son:

**Tabla 2.** Métodos de ciclo de vida de React

|  |  |
| --- | --- |
| Métodos de ciclo de vida de React | Descripción |
| Nombre del Método | Utilidad |
| componentWillMount() | Se ejecuta justo antes del primer pintado del componente. |
| componentWillReceiveProps(nextProps) | Se invoca cuando las propiedades se van a actualizar. |
| shouldComponentUpdate(nextProps, nextState) | Se invocará cuando se producen cambios de propiedades o cambios de estado. |
| componentWillUpdate(nextProps, nextState) | Este se invocará justo antes de que el componente vaya a actualizar su vista. |
| componentDidUpdate(prevProps, prevState) | Este se invocará justo después de que el componente vaya a actualizar su vista. |
| componentWillUnmount() | Se ejecuta en el momento que el componente se va a retirar del DOM. |

Se puede revisar en el siguiente código cómo quedará el archivo:

|  |
| --- |
| import React from 'react';  export default class CicloVida extends React.Component {  constructor(props) {  console.log('Ejecutando constructor', ...args);  super(props);  this.state = {  estado: 'Inicializado en el constructor'  };  }  componentDidMount() {  console.log('Ejecuta componentDidMount');  }  componentWillUnmount() {  console.log('Ejecuta componentWillUnmount');  }  componentDidUpdate(prevProps, prevState) {  console.log('Ejecuta componentDidUpdate. Anteriores propiedades y estado:', prevProps, prevState);  }  shouldComponentUpdate(nextProps, nextState) {  console.log('Ejecuta shouldComponentUpdate. Próximas propiedades y estado:', nextProps, nextState);  return true;  }  componentWillReceiveProps(nextProps) {  console.log('Ejecuta componentWillReceiveProps con las propiedades futuras:', nextProps);  }  componentWillUpdate(nextProps, nextState) {  console.log('Ejecuta componentWillUpdate. Próximas propiedades y estado:', nextProps, nextState);  }  componentDidCatch(error, info) {  console.log('Ejecuta componentDidCatch:', error, info);  }  render() {  console.log('Ejecuta render');  return (  <div>  <h1>Ciclo de vida del componente</h1>  <p>Estado: {this.state.estado}</p>  <p>Propiedad: {this.props.propiedad}</p>  </div>  );  }  } |

|  |  |
| --- | --- |
| A red bell with a exclamation mark  Description automatically generated | **¡Atención!** En React.js los componentes que no sean puros (todos los que se crean mediante clases o React.createClass) poseen algo conocido como el ciclo de vida. Este ciclo de vida, son una serie de funciones que se ejecutan en distintos momentos de la vida del componente y nos permiten realizar distintas acciones en estos momentos. |

**Ejecutar la aplicación**

Para ejecutar una aplicación React JS, se utiliza el comando **npm start**. Para generar la versión de producción, se debe ejecutar el siguiente comando: **npm run build**.

|  |
| --- |
| > my-app@0.1.0 build /home/chmunoz/my-app  > react-scripts build  Creating an optimized production build...  Compiled successfully.  File sizes after gzip:  41.34 KB build/static/js/2.8833cde.chunk.js  1.62 KB build/static/js/3.ff0311e9.chunk.js  1.38 KB build/static/js/runtime~main.1c420381.js  1.31 KB build/static/js/main.68b9e600.chunk.js  506 B build/static/css/main.cfa70044.chunk.css  The project was built assuming it is hosted at /.  You can control this with the homepage field in your package.json.  The build folder is ready to be deployed.  You may serve it with a static server:  npm install -g serve  serve -s build  Find out more about deployment here:  https://cra.link/deployment |

Este comando genera la carpeta **build** con la versión compilada y minificada del proyecto React SPA (Single Page Application). A continuación, se describirá cómo debe quedar la organización y el registro de este proceso.

|  |
| --- |
| build  ├── asset-manifest.json  ├── favicon.ico  ├── index.html  ├── logo192.png  ├── logo512.png  ├── manifest.json  ├── robots.txt  └── static  ├── css  │ ├── main.a617e044.chunk.css  │ ├── main.a617e044.chunk.css.map  ├── js  │ ├── 2.8833cd6e.chunk.js  │ ├── 2.8833cd6e.chunk.js.LICENSE.txt  │ ├── 2.8833cd6e.chunk.js.map  │ ├── 3.ff0311e9.chunk.js  │ ├── 3.ff0311e9.chunk.js.map  │ ├── main.6bc689e0.chunk.js  │ ├── main.6bc689e0.chunk.js.map  │ ├── runtime-main.1c420381.js  │ └── runtime-main.1c420381.js.map  └── media  └── logo.6ce24c58.svg |

1. **Hibernate**

Hibernate es *software* libre con licencia GNU LGPL que presta servicios de ORM para Java. Es un plugin que puede ser integrado al IDE de desarrollo en Java y facilita el proceso de mapeo de tablas en una base de datos relacional a objetos Java mediante el uso de archivos declarativos en formato XML o anotaciones.

|  |  |
| --- | --- |
| A black background with grey and yellow letters  Description automatically generated | “Hibernate utiliza JDBC para todos los procesos de comunicación con la base de datos, funcionando como una capa adicional de abstracción de las funcionalidades del JDBC. En una aplicación Java, podremos delegar la mayoría de las operaciones de bases de datos al *framework de* Hibernate*,* pero en segundo plano todo estará pasando por la API de JDBC” (Martinez, 2015). |

Así que, cuando se configura Hibernate para conectarse a una base de datos, lo que realmente se está haciendo es configurar Hibernate para el uso del JDBC, y es este quien realiza las operaciones. A continuación, se presenta el esquema general del funcionamiento:

Hibernate

A close-up of a white and grey square

Description automatically generated

Teniendo en cuenta que Hibernatedebe usar el *driver* JDBC para la ejecución de las instrucciones en la base de datos, se debe construir un archivo de configuración en formato XML donde se especifique la URL del JDBC y las credenciales de acceso a la base de datos. En la siguiente figura se presenta el esquema general de funcionamiento en el proceso de configuración:

Hibernate archivo de configuración

A black rectangular object with white text and arrows

Description automatically generated

Este archivo de configuración define un conjunto de propiedades que serán utilizadas para crear las sesiones mediante las cuales se inician un conjunto de transacciones u operaciones sobre la base de datos. Entre las propiedades más importantes se encuentran las siguientes:

**Tabla 3.** Propiedades Hibernate

|  |  |
| --- | --- |
| Propiedad | Definición |
| connection.driver.class | Parte de la configuración del JDBC donde se define el *driver* a usar. |
| connection.url | Parte de la configuración del JDBC donde se define la url de la base de datos a utilizar (protocolo: url base de datos: puerto / nombre DB). |
| connection.username | Parte de la configuración del JDBC donde se define el usuario de la base de datos a utilizar en la conexión. |
| connection.password | Parte de la configuración del JDBC donde se define el *password d*el usuario de la base de datos a utilizar en la conexión. |
| connection.pool\_size | Número de conexiones simultáneas al JDBC. |
| dialect | Aunque SQL es un estándar, cada motor de bases de datos tiene una implementación particular. En esta propiedad se especifica el dialecto a utilizar por Hibernate. |
| show\_sql | Esta propiedad permite ver en la salida estándar las sentencias SQL que ejecuta en *background* Hibernate lo cual sirve mucho para entender lo que está sucediendo en entornos de desarrollo. |
| current\_session\_context\_class | Especifica el contexto de la sesión actual, por defecto se puede asignar *thread*. |

Es importante tener en cuenta que este archivo de configuración de Hibernate debe estar ubicado en la raíz de la carpeta **src** del proyecto, de forma que pueda ser utilizado por Hibernate en cualquier contexto de la aplicación. A continuación, se muestra cómo debe verse un archivo de configuración de *Hibernate* completo, donde se establece una conexión con una base de datos MySQL en un servidor que se está ejecutando en la máquina local (**localhost**) y puerto 3306, con el usuario *“root”* y contraseña 12345:

|  |
| --- |
| <!DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">  <hibernate-configuration>  <session-factory>  <!-- JDBC Database connection settings -->  <property name="connection.driver\_class">com.mysql.cj.jdbc.Driver</property>  <property name="connection.url">jdbc:mysql://localhost:3306/prueba?useSSL=false&serverTimezone=UTC</property>  <property name="connection.username">root</property>  <property name="connection.password">12345</property>  <!-- JDBC connection pool settings ... using built-in test pool -->  <property name="connection.pool\_size">1</property>  <!-- Select our SQL dialect -->  <property name="dialect">org.hibernate.dialect.MySQLDialect</property>  </session-factory>  </hibernate-configuration>  <!-- Echo the SQL to stdout -->  <property name="show\_sql">true</property>  <!-- Set the current session context -->  <property name="current\_session\_context\_class">thread</property>  </session-factory>  </hibernate-configuration> |

Se invita a revisar en los anexos el archivo de configuración mostrado anteriormente para que pueda ser descargado y utilizado en la construcción de su proyecto, teniendo en cuenta que debe editar las propiedades de acuerdo con sus necesidades.

Luego de esto, se requiere definir todas las **“Entity Classes”** que serán utilizadas por *Hibernate* para realizar el proceso de mapeo hacia una base de datos relacional y viceversa. Una **“Entity Class”** básicamente es una clase Java tradicional que tiene las mismas características definidas en una tabla de la base de datos, es decir, es una imagen de la tabla, pero desde la perspectiva de la programación orientada a objetos. Adicionalmente, tiene unas anotaciones especiales que le permiten a *Hibernate* realizar el trabajo adecuado.

Entity Classes

A diagram of a diagram

Description automatically generated with medium confidence

Las anotaciones pueden ser utilizadas al importar la librería de persistencia de JAVA javax.persistence.\*, entre las más utilizadas encontramos:

**Tabla 4.** Anotaciones Java

|  |  |
| --- | --- |
| Anotación | Significado |
| @Entity | Especifica que la clase definida en una *Entity Class*. |
| @Table(name = “ ”) | Permite especificar que la clase será mapeada con una tabla en la base de datos especificada en la propiedad *name*. |
| @Column(name = “ ”) | Permite especificar que el atributo por ser definido será mapeado con una columna especificada por la propiedad *name*. Cuando el nombre del atributo coincide con el nombre de la columna en la tabla se puede omitir la especificación de la propiedad *name* y solo dejar la anotación @Column. |
| @Id | Permite especificar que el atributo a definir corresponde con el identificador primario a ser mapeado en la tabla. |

A continuación, se presenta un corto video donde se agrega el archivo de configuración de *Hibernate* al proyecto y se crea la clase *Entity* para la gestión de una tabla en una base de datos MySQL.

**Video.** *Creación clase Entity*

A screen shot of a computer

Description automatically generated

<https://youtu.be/qYCN1ymhE4Y>

**4.1. Operaciones básicas**

Una vez se tienen registradas todas las dependencias, los archivos de configuración de Hibernate y las respectivas clases de entidad, se pueden utilizar las facilidades ofrecidas por Hibernate para la manipulación de la base de datos de forma transparente por medio de los procesos de mapeo que ofrece este *framework.*

Hibernate utiliza principalmente dos clases que debemos conocer y utilizar, las cuales se describen a continuación:

En el siguiente fragmento de código se detalla cómo es el proceso de definición de los objetos *SessionFactory y Session* para poder realizar la manipulación de la base de datos:

|  |
| --- |
| // creación del session factory  SessionFactory factory = new Configuration()  .configure("hibernate.cfg.xml")  .addAnnotatedClass(Usuario.class)  .buildSessionFactory();  // creación de la session a partir del Session factory  Session session = factory.getCurrentSession();  try {  // Ahora se puede usar el objeto session para la manipulación de la BD  } finally {  // Se cierra el objeto session factory una vez se deja de utilizar  factory.close();  } |

Note que en el código anterior, para la creación del objeto **SessionFactory**, se indica el nombre del archivo de configuración de Hibernate en el método **configure()**. Si este parámetro no se incluyera, Hibernate buscará en el directorio raíz del proyecto un archivo que contenga el nombre por defecto de este tipo de archivos de configuración, el cual es **hibernate.cfg.xml**.

También note que en el método **addAnnotatedClass()** se indica la clase de entidad a ser utilizada.

**Método Save**

El método **save()** del objeto **Session** permite realizar el proceso de inserción de un nuevo registro en la base de datos enviando como parámetro un objeto de tipo entidad con sus valores. La forma correcta de realizar un proceso de inserción de un nuevo registro en una base de datos con un objeto de tipo sesión incluye los siguientes pasos:

|  |
| --- |
| import java.util.List;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import com.jonathan.hibernate.demo.entity.Usuario;  public class CrearUsuarioDemo {  public static void main(String[] args) {  // Crear SessionFactory  SessionFactory factory = new Configuration()  .configure("hibernate.cfg.xml")  .addAnnotatedClass(Usuario.class)  .buildSessionFactory();  // Crear sesión  Session session = factory.getCurrentSession();  try {  System.out.println("Creando un usuario...");  // Crear un objeto Usuario  Usuario usuario1 = new Usuario("Francisco", "FFFFF");  // Iniciar una transacción  session.beginTransaction();  // Guardar el objeto Usuario  System.out.println("Guardando el nuevo usuario...");  session.save(usuario1);  // Confirmar la transacción  session.getTransaction().commit();  System.out.println("Proceso finalizado.");  } finally {  factory.close();  }  }  } |

**Método Get**

El método **get()** del objeto **session** permite obtener un registro de la base de datos por medio de una consulta por su llave primaria. Este método recibe como parámetro la clase de entidad que será utilizada para hacer el mapeo del registro en la respuesta y un segundo parámetro que corresponde al valor a ser buscado en la llave primaria de la tabla. Este método ofrece como resultado un objeto de la clase de entidad especificada en la invocación del método.

La forma correcta de realizar un proceso de consulta de un registro particular por medio de su llave primaria y un objeto de tipo sesión incluye los siguientes pasos:

|  |
| --- |
| import java.util.List;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import com.jonathan.hibernate.demo.entity.Usuario;  public class CrearLeerDemo {  public static void main(String[] args) {  // Crear SessionFactory  SessionFactory factory = new Configuration()  .configure("hibernate.cfg.xml")  .addAnnotatedClass(Usuario.class)  .buildSessionFactory();  // Crear sesión  Session session = factory.getCurrentSession();  try {  // Iniciar una transacción  session.beginTransaction();  // Consulta de un usuario cuyo pk tiene el valor de 16  System.out.println("Consulta de un usuario");  Usuario resultado = session.get(Usuario.class, 16);  System.out.println("Resultado = " + resultado);  // Confirmar la transacción  session.getTransaction().commit();  System.out.println("Proceso finalizado.");  } finally {  factory.close();  }  }  } |

**Método CreateQuery**

Hibernate permite el proceso de creación de consultas usando HQL, el cual es una versión especial de Hibernate para establecer consultas SQL orientadas a objetos y que comprende nociones propias del paradigma orientado a objetos, como la herencia, el polimorfismo y las asociaciones. Es importante tener en cuenta que, aunque las palabras reservadas y sentencias de **SQL** no son sensibles a mayúsculas, los nombres de las clases y sus atributos usados en HQL sí lo son.

Para el caso en el que se requiera realizar consultas más avanzadas, se puede utilizar la potencia de HQL para establecer las características de los registros a ser consultados. Como este tipo de consulta puede producir más de un registro proveniente de la base de datos, es necesario usar el método **getResultList()** para mapear la respuesta a una lista de objetos Java. En la siguiente tabla se presentan algunas de las cláusulas HQL más comúnmente utilizadas. Se debe tener en cuenta que en HQL se hace referencia a las clases de entidad Java con sus atributos, los cuales son sensibles a mayúsculas. Estas sentencias son mapeadas a su estructura equivalente en la base de datos usando los nombres de tablas y atributos, los cuales ya no son sensibles a mayúsculas.

**Tabla 5.** Sentencias y equivalentes

|  |  |
| --- | --- |
| Sentencia HQL | Equivalente en SQL |
| from Usuario | SELECT \* FROM usuario |
| from Usuario u where u.userName = ‘Raul’ | SELECT \* FROM usuario U  WHERE u.username = ‘Raul’ |
| from DomesticCat cat | SELECT \* FROM domesticCat Cat |
| order by cat.name asc | ORDER BY Cat.name ASC |
| order by cat.name asc | ORDER BY Cat.name ASC |
| from Cat as cat  inner join Cat.mate as mate | SELECT \* FROM cat cat  INNER JOIN cat\_mate mate ON Cat.id = mate.id |

A continuación, se presentan diferentes ejemplos haciendo uso de consultas por medio de HQL:

|  |
| --- |
| import java.util.List;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import com.javahibernate.hibernate.demo.entity.Usuario;  public class ConsultarUsuarios {  public static void main(String[] args) {  // Crear el SessionFactory  SessionFactory factory = new Configuration()  .configure("hibernate.cfg.xml")  .addAnnotatedClass(Usuario.class)  .buildSessionFactory();    // Crear el Session  Session session = factory.getCurrentSession();  try {  // Consultar todos los usuarios de la tabla Usuario  List<Usuario> usuariosRegistrados = session.createQuery("from Usuario").list();  for (Usuario usuario : usuariosRegistrados) {  System.out.println(usuario);  }  // Consultar usuarios con nombre que comienza con 'J'  usuariosRegistrados = session.createQuery("from Usuario u where u.nombre like 'J%'").list();  for (Usuario usuario : usuariosRegistrados) {  System.out.println(usuario);  }  // Consultar usuarios con apellido 'Ramirez'  usuariosRegistrados = session.createQuery("from Usuario u where u.apellido = 'Ramirez'").list();  for (Usuario usuario : usuariosRegistrados) {  System.out.println(usuario);  }  // Consultar usuarios con email que contiene 'javacode'  usuariosRegistrados = session.createQuery("from Usuario u where u.email like '%javacode%'").list();  for (Usuario usuario : usuariosRegistrados) {  System.out.println(usuario);  }  // Consultar usuarios con nombre 'Francisco' o que su password es '12345'  usuariosRegistrados = session.createQuery("from Usuario u where u.nombre = 'Francisco' OR u.password = '12345'").list();  for (Usuario usuario : usuariosRegistrados) {  System.out.println(usuario);  }  } finally {  factory.close();  }  }  } |

**Actualizaciones**

El proceso de actualización es muy similar al proceso de consulta. Se puede hacer uso directo de los objetos obtenidos de las clases de entidad mediante los métodos modificadores de sus atributos (métodos **set()**) y confirmar las transacciones realizadas. También se puede hacer uso de **createQuery()**, el cual permite enviar consultas HQL un poco más elaboradas, pero asociándolo con el método **executeUpdate()** para que los cambios se reflejen en la base de datos una vez se confirma la transacción.

|  |
| --- |
| import java.util.List;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import com.javahibernate.demo.entity.Usuario;  public class ConsultarUsuario {  public static void main(String[] args) {  // Crear el SessionFactory  SessionFactory factory = new Configuration()  .configure("hibernate.cfg.xml")  .addAnnotatedClass(Usuario.class)  .buildSessionFactory();    // Crear el Session  Session session = factory.getCurrentSession();  try {  // Iniciar una transacción  session.beginTransaction();  // Consultar un usuario por su clave primaria igual a 6  Usuario consultaUsuario = session.get(Usuario.class, 6);  consultaUsuario.setUserName("Nuevo Raul");  System.out.println("Usuario = " + consultaUsuario.getUserId() + " Actualizado");  // Confirmar la transacción  session.getTransaction().commit();  System.out.println("Proceso finalizado");  } finally {  factory.close();  }  }  } |

|  |
| --- |
| import java.util.List;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import com.javahibernate.demo.entity.Usuario;  public class ActualizarUsuarios {  public static void main(String[] args) {  // Crear el SessionFactory  SessionFactory factory = new Configuration()  .configure("hibernate.cfg.xml")  .addAnnotatedClass(Usuario.class)  .buildSessionFactory();    // Crear el Session  Session session = factory.getCurrentSession();  try {  // Iniciar una transacción  session.beginTransaction();  // Actualizar la contraseña de todos los usuarios cuyo password actual sea '678'  System.out.println("Actualizando varios usuarios...");  session.createQuery("update Usuario u set u.userPassword = 'Z@Q#J!' where u.userPassword = '678'")  .executeUpdate();  // Confirmar la transacción  session.getTransaction().commit();  System.out.println("Proceso finalizado");  } finally {  factory.close();  }  }  } |

**Método Delete**  
El método **delete()** del objeto **session** permite eliminar un registro de la base de datos mediante un objeto recibido como parámetro, que identifica el registro de la tabla a eliminar. La forma correcta de realizar un proceso de eliminación de un registro particular incluye los siguientes pasos:

|  |
| --- |
| import java.util.List;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import com.javahibernate.demo.entity.Usuario;  public class EliminarUsuarioDemo {  public static void main(String[] args) {  // Crear el SessionFactory  SessionFactory factory = new Configuration()  .configure("hibernate.cfg.xml")  .addAnnotatedClass(Usuario.class)  .buildSessionFactory();    // Crear el Session  Session session = factory.getCurrentSession();  try {  // Iniciar una transacción  session.beginTransaction();  // Eliminar un usuario cuyo id es 6  System.out.println("Eliminación de un usuario");  Usuario usuarioDelete = session.get(Usuario.class, 6);  session.delete(usuarioDelete);  System.out.println("Usuario " + usuarioDelete.getUserName() + " eliminado");  // Confirmar la transacción  session.getTransaction().commit();  System.out.println("Proceso finalizado");  } finally {  factory.close();  }  } |

|  |
| --- |
| import java.util.List;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import com.javahibernate.demo.entity.Usuario;  public class EliminarUsuarioDemo {  public static void main(String[] args) {  // Crear el SessionFactory  SessionFactory factory = new Configuration()  .configure("hibernate.cfg.xml")  .addAnnotatedClass(Usuario.class)  .buildSessionFactory();    // Crear el Session  Session session = factory.getCurrentSession();  try {  // Iniciar una transacción  session.beginTransaction();  // Eliminar todos los usuarios cuyo id es 6  System.out.println("Eliminando el usuario de id = 6");  session.createQuery("delete from Usuario u where u.userId = 6").executeUpdate();  // Confirmar la transacción  session.getTransaction().commit();  System.out.println("Proceso finalizado");  } finally {  factory.close();  }  }  } |

**4.2. Asociaciones**

Normalmente, al desarrollar una aplicación que gestione una base de datos, se requiere mucho más que una única tabla para la gestión de la información. Es común tener modelos de datos con múltiples tablas relacionadas con asociaciones de diferentes tipos y diferentes cardinalidades. Por lo tanto, es importante establecer en las clases de entidad este tipo de relaciones para que el proceso de mapeo sea coherente. Los tipos de asociaciones más comúnmente utilizadas son las que expresan relaciones uno a uno, relaciones uno a muchos, relaciones muchos a uno y relaciones muchos a muchos.

Para explicar cada uno de estos casos, utilizaremos como referencia el siguiente modelo de datos que ya fue mapeado de forma individualizada. A medida que se agregue una asociación de diferente tipo, se establecerán los cambios que deberían estar presentes, y que se detallan en la siguiente figura:

Asociaciones

A screenshot of a computer program

Description automatically generated

**Relaciones *OneToOne***

En este tipo de relaciones, se quiere expresar un vínculo en el que un ejemplar de una tabla tiene relación con uno y solo un ejemplar de una segunda tabla. Desde el punto de vista de las bases de datos, se creará una relación de integridad referencial vinculando una llave primaria con una llave foránea. Tomando como referencia el modelo de datos presentado anteriormente, supongamos que se desea indicar que un usuario tiene asociado un recurso. Para resolver este nuevo requerimiento, debemos modificar la clase de entidad de usuario para agregar un nuevo atributo que haga referencia al objeto recurso con el cual está relacionado y agregar la anotación correspondiente llamada **@OneToOne**, de forma que Hibernate haga el mapeo correcto. El esquema se detalla a continuación en la siguiente figura:

Relaciones *OneToOne*

A screenshot of a computer

Description automatically generated

Mientras que en la base de datos se debe agregar un nuevo campo foráneo en la tabla usuario que haga referencia a la llave primaria de la tabla recurso para poder establecer la relación, en las clases de tipo entidad bastará con especificar el nuevo atributo del tipo de objeto correspondiente que referencia agregando la anotación de tipo **@OneToOne**. Si el campo nuevo requerido en la tabla usuario no existe, el mapeador de Hibernate se encargará de agregarlo.

**Relaciones *OneToMany – ManyToOne***

Las relaciones **"*OneToMany"*** tienen la particularidad de afectar cada una de las entidades vinculadas en la relación y es importante identificar en qué lugar es una relación ***"OneToMany"*** y en qué lugar la relación es***"ManyToOne"****.* Recordando las reglas de transformación del modelo lógico de una base de datos relacional, en este tipo de relaciones se indica que en la tabla del lado de la relación donde la cardinalidad máxima es muchos (\*) deberá ser transferido como un campo foráneo la llave primaria de la tabla donde la cardinalidad máxima es 1, como se aprecia en la siguiente figura:

Relaciones *OneToMany – ManyToOne*

A screenshot of a computer

Description automatically generated

Para el caso de las clases de entidad, deberá especificarse en la clase donde la cardinalidad máxima es 1, un atributo del tipo lista de la clase de entidad con la que tiene un vínculo **@OneToMany**. Este atributo será de tipo lista porque se está vinculando con varios. En el ejemplo, la entidad usuario se relaciona con varios recursos, es decir, un usuario puede estar vinculado con múltiples recursos.

Por otra parte, la clase de entidad donde la cardinalidad máxima es muchos deberá especificar un atributo de tipo objeto con la entidad vinculada, agregándole la anotación **@ManyToOne**. En el ejemplo, un recurso estará siempre vinculado con un único usuario. Es importante anexar en la entidad de cardinalidad **@OneToMany** el nombre del atributo en la entidad **@ManyToOne** que será usado para el mapeo de la asociación por parte de Hibernate. En el ejemplo, se indica en la entidad Usuario que la relación **@OneToMany** será mapeada usando el atributo *user* de la entidad **@ManyToOne**.

**Relaciones *ManyToMany***

Las relaciones ***ManyToMany*** obligan a la construcción de una nueva entidad de mapeo en la cual se alojarán las respectivas llaves primarias de las entidades vinculadas por medio de la relación **@ManyToMany.** Para su correcta implementación, deberán agregarse en cada una de las clases entidad un objeto de tipo lista de la entidad a la cual está relacionada, etiquetada con la anotación @ManyToMany. Adicionalmente, en una de las clases de entidad se deberá especificar el objeto de mapeo del otro extremo que será utilizado por Hibernate para realizar la asociación.

En el ejemplo siguiente, se establece una relación *ManyToMany* entre la clase Usuario y la clase Recurso, por lo cual en cada una de ellas se agrega la correspondiente lista del tipo de entidad a la cual se asocian con la anotación **@ManyToMany.** En el caso de la clase Usuario, se agrega el atributo de tipo lista de Recursos llamado recursos para indicar que un usuario puede estar vinculado con muchos recursos. Por otro lado, en la clase Recurso se agrega una lista de tipo Usuario para indicar que un recurso puede estar vinculado con múltiples usuarios. Adicionalmente, la relación entre usuario y recurso se mapea con el objeto *users* presente en la clase Recurso. La relación entre recurso y usuario no se mapea explícitamente, por lo que generará la tabla intermedia **recurso\_usuario,** lo cual cumple las reglas de transformación de un modelo lógico para una relación de muchos a muchos.

Relaciones *ManyToMany*

A screenshot of a computer

Description automatically generated

**4.3 Ciclo de vida**

Los objetos manipulados por Hibernate para el proceso de gestión de la persistencia pasa por cuatro estados diferentes, a continuación, se describe cada uno de ellos:

|  |
| --- |
| **PESTAÑAS**  **CF020\_4.3\_Ciclo de vida** |

A continuación, se presenta un resumen de los estados del ciclo de vida y sus transiciones completas con todos los posibles métodos.

Ciclo de vida

A diagram of a diagram

Description automatically generated

1. **Spring**

Spring es un *framework* de código abierto que facilita la creación de aplicaciones empresariales de alto rendimiento en Java. Además de Java, tiene soporte para los lenguajes Groovy y Kotlin. La estructura que lo compone es modular, ofreciendo flexibilidad para implementar una variedad de arquitecturas dependiendo de los requerimientos de las aplicaciones, haciendo que la programación sea lo más ágil posible, rápida y muy segura.

La siguiente figura presenta las versiones de *Spring Framework*:

Versiones de *Spring Framework*

A diagram of a timeline

Description automatically generated

A continuación, se presentan algunas de las características clave que hacen de Spring una opción popular entre los desarrolladores:

|  |
| --- |
| ACORDEÓN  CF020\_5\_Spring |

A continuación, se detalla lo que se puede crear con *Spring Framework*: aplicaciones de consola, aplicaciones de escritorio, aplicaciones web, aplicaciones móviles, servicios SOAP y REST. Spring está orientado esencialmente al *back-end*.

**5.1. Introducción**

El *framework* Spring, básicamente con su inyección de dependencias, se encarga de construir objetos y configurar todas las dependencias que estos necesiten. En Spring, estos objetos creados son llamados *beans*, y se anotan en las clases de la siguiente manera: ***@Component*, *@Service*, *@Repository*,** etc. Uno de los principios principales de Spring es el de **Inversión de Dependencia (DIP),** el cual establece que los módulos de alto nivel no deben depender de los de bajo nivel. Esto significa que se debe hacer referencia a las abstracciones más que a las implementaciones, evitando depender de una clase o implementación en particular.

La implementación de este principio se llama Inversión de Control (IoC), que se encarga de desacoplar el código utilizando eventos. Este principio también se conoce como el principio de Hollywood: "no nos llames; nosotros te llamaremos". Consiste en una manera de programar donde la aplicación interviene en lugar del código. Para implementar el IoC, se necesita un agente externo llamado contenedor, que es el encargado de verificar el flujo de la aplicación. Los IoC utilizan *Service Locator*, eventos y DI (inyecciones de dependencias), enfocando el Spring *Framework* en la DI.

**Inyección de dependencias:** es un patrón de diseño de *software*, subtipo de inversión de control, cuyo objetivo es tener un código fácil de mantener o desacoplado. Su función es determinar lo que los objetos necesitan.

Principio inversión de control

A diagram of a company

Description automatically generated

A continuación, se presenta un ejemplo de código donde los objetos dependen entre sí:

Acoplamiento hermético

A screenshot of a computer screen

Description automatically generated

En el código anterior, se obtuvo el objeto desde el exterior y que el objeto puede ser cualquier tipo de café, sin necesidad de modificar la clase TiendaCafe. Resumiendo, la inversión de control es un patrón de programación para obtener un acoplamiento flexible, logrando que los objetos independientes no sean responsables de sus dependencias, sino que las reciban desde afuera. El contenedor de IoC recoge los metadatos que vienen de un archivo XML o código Java y recibe las indicaciones de los objetos que debe crear, enlazar, configurar y destruir por medio de POJOs (Objetos simples de Java), creando objetos por medio del proceso que recibe el nombre de *Spring Bean*s y resultando en objetos preparados para la inyección de dependencias.

La siguiente figura describe el funcionamiento de una vista abstracta de Spring.

Vista abstracta de *Spring*.

A diagram of a spring container

Description automatically generated

**Módulos del Framework**

*Spring Framework* es un marco de trabajo muy amplio y brinda cerca de 20 módulos, los cuales se asocian en el contenedor principal: accesos e integración de datos, Web, *AOP (Programación orientada a aspectos),* Instrumentación, Mensajes y Pruebas (Spring, 2021).

Para trabajar con *Spring* se necesita tener instalados los siguientes requerimientos:

**Tabla 6.** Entornos para desarrollo en Spring

|  |  |
| --- | --- |
| Entornos de desarrollo | Enlace |
| IntelliJ IDEA | <https://www.jetbrains.com/idea/> |
| Spring Tools | <https://spring.io/tools> |
| Visual Studio Code | <https://code.visualstudio.com/docs/languages/java> |
| Eclipse | <https://www.eclipse.org/downloads/packages/> |
| NetBeans | <https://netbeans.apache.org/> |

**Tabla 7.** JDK para desarrollo en Spring

|  |  |
| --- | --- |
| JDK | Enlace |
| JDK Oracle | <https://www.oracle.com/java/technologies/downloads/> |
| AdoptOpenJDK | <https://adoptopenjdk.net/> |

Para construir aplicaciones en *Spring Framework* necesitamos:

|  |
| --- |
| **PASOS**  **CF020\_5.1\_Introducción (1)** |

**Creación de la aplicación**

A continuación, se presenta como crear la aplicación:

|  |
| --- |
| **Acordeón**  **CF020\_5.1\_Introducción (2)** |

**5.2. Spring Data**

Spring ofrece una variedad de proyectos independientes que se pueden articular en un proyecto principal. Cada proyecto permite resolver una necesidad especial del negocio; algunos de ellos se ilustran en la siguiente figura.

Proyectos Spring

A screenshot of a black background

Description automatically generated

Uno de los proyectos más utilizados es **Spring Data,** ya que es uno de los módulos más grandes de Spring debido a que cuenta con muchos submódulos. Su objetivo principal es proporcionar una serie de herramientas que permiten realizar integraciones muy sencillas con los diferentes entornos de almacenes de datos disponibles. Los datos que se utilizan pueden proceder de orígenes tanto de bases de datos relacionales como no relacionales.

Las funcionalidades de Spring Data son:

**Módulos de Spring Data**

Los módulos oficiales de Spring Data son:

A continuación, se presenta un ejemplo de un mapeo para JPA, que es uno de los submódulos de *Spring Data*. En el ejemplo, se desarrolla la clase **Persona**, la cual representa una tabla en la base de datos llamada **Persona**:

|  |  |
| --- | --- |
| @Entity(name = "persona")  public class Persona {  @Column(name = "id")  private Long id;  @Column(name = "persona\_nombre")  private String nombre;  @Column(name = "persona\_apellido")  private String apellido;  @Column(name = "persona\_telefono")  private String telefono;  } | 1. La creación de la tabla se especifica mediante la anotación **@Entity** y en el atributo **name** se coloca el nombre de la tabla. 2. Por medio de la anotación @Column se asigna cada uno de los campos de la tabla de la base de datos a los atributos de la clase Persona con la etiqueta “name” se hace referencia al nombre de la columna de la tabla. |

Al extender de la clase **CrudRepository**, se facilita el uso de métodos como **findByNombre**, que genera una consulta de todas las personas con el parámetro **nombre**. En bases de datos relacionales, esta operación se traduce en **SELECT \* FROM persona WHERE nombre=?**. Este enfoque reduce considerablemente las sentencias SQL, permitiendo obtener consultas dinámicas de forma más eficiente.

**5.3. *Spring Boot* y MySQL**

*Spring Boot* es una infraestructura liviana que suprime en gran parte la tarea de configurar las aplicaciones fundamentadas en Spring. A diferencia de *Spring Framework*, *Spring Boot* realiza una simplificación en cuanto a configuración y el despliegue de las aplicaciones.

Las características de *Spring Boot son*:

*Spring Boot* cuenta con dos mecanismos que permiten la simplificación de los procesos: el contenedor de aplicaciones integrado y los *starters*.

**Contenedor de aplicaciones integrado**

La compilación de las aplicaciones web se realiza como un archivo .jar desplegado en un servidor Tomcat, facilitando la distribución de las aplicaciones. Anteriormente, se trabajaba con .war, que si se requiere, se puede seguir utilizando.

***Starters***

Son los encargados de brindar una variedad de dependencias que se agregan al proyecto; estas pueden ser de terceros o de Spring. Los *starters* por defecto traen una configuración de valores predeterminados.

Otra de las ventajas de *Spring Boot* es que ofrece un mecanismo que genera la estructura del proyecto automáticamente. Este inicializador crea el archivo *pom.xml*, el main de la aplicación y un archivo de test para verificar que la aplicación funcione correctamente.

Inicializador de Spring

A screenshot of a computer

Description automatically generated

A continuación, se realizan los pasos para la creación de un proyecto en Spring Boot. Hay que recordar que se deben tener instaladas cualquiera de las herramientas de Entorno de desarrollo y JDK.

|  |
| --- |
| **Acordeón**  **CF020\_5.3\_Spring Boot y MySQL (1)** |

Lo que hace es ejecutar el servidor Tomcat que ya viene por defecto dentro de *Spring Boot*. Ahora se debe ingresar al navegador y escribir "localhost" con el puerto 8080, y se presentará la ejecución de la aplicación. En la siguiente videoclase se explica de forma más detallada cómo utilizar *Spring Boot*:

**Video.** *Spring boot*

A computer screen with a black and green background

Description automatically generated

<https://youtu.be/j-_0AwuvLl4>

|  |
| --- |
| **Spring Boot y MySQL**  Para tener una aplicación web completa, es necesario almacenar información. Por esta razón, es importante realizar conexiones con bases de datos. En este ejemplo, se detalla cómo realizar la conexión con la base de datos MySQL. |

Los requerimientos fundamentales son tener instalado MySQL, el entorno de desarrollo y seguir estos pasos:

|  |
| --- |
| **Slide**  **CF020\_5.3\_Spring Boot y MySQL (2)** |

El paso siguiente es abrir el archivo **“aplication.properties”** y configurar todas las propiedades de la conexión así:

|  |
| --- |
| 1.spring.jpa.hibernate.ddl-auto=update  2.spring.datasource.url=jdbc:mysql://**localhost:3306/demo**?useUnicode=true&characterEncoding=utf8&createDatabaseIfNotExist=true  3.spring.datasource.username=root  4.spring.datasource.password=password  5.spring.jpa.show-sql=true |
| A continuación, se explicará cada línea del anterior código que va en el archivo **“aplication. properties”.**  **Línea 1:** permite obtener los metadatos de la base de datos.  **Línea 2:** se muestra la url de la conexión con el motor de base de datos.  **Línea 3:** Se especifica el nombre del usuario de la base de datos para la conexión  **Línea 4:** Se especifica clave de la base de datos para la conexión  **Línea 5:** la propiedad true permite utilizar las consultas JPA a realizar en la base de datos. |

Lo que queda es ejecutar la aplicación y comprobar que no existan problemas con la conexión a la base de datos. Para ejecutar, dar clic derecho en el archivo Java, seleccionar la opción >Run As y >Spring Boot App.

Si todo está bien, ahora solo queda ir al motor de bases de datos y verificar si se creó el esquema.

Ejecución de la aplicación

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* 1. **Validaciones**

Es muy importante realizar validaciones cuando se están desarrollando aplicaciones web en Java para impedir que se ingresen datos que afecten al programa. En Spring viene incluido el *hibernate-validator* en el paquete *spring-boot-starter-web*, que permite validar valores de los campos *beans*.

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  <version>2.0.0.RELEASE</version>  </dependency> |

Para el ejemplo, se realiza la creación de un bean con unas anotaciones que permiten la validación de ingreso de un usuario.

|  |
| --- |
| @Data  @NoArgsConstructor  public class Usuario {  @NotBlank(message = "Debe ingresar un nombre de usuario")  private String nombreUsuario;  @NotBlank(message = "Debe ingresar un valor correcto para la edad")  @Pattern(regexp = "^[0-9]{1,2}$", message = "Edad incorrecta")  private String edad;  @AssertFalse(message = "Debe ser falso")  private Boolean esfalso;  @Pattern(regexp = "^[0-9]{4}-[0-9]{2}-[0-9]{2}$", message = "No es correcto el formato de la fecha")  private String fechanaciemiento; |

Una vez se crean las anotaciones, se debe crear el controlador.

|  |
| --- |
| @Controller  public class UserController {  @RequestMapping("/usuario")  public void create(@RequestBody @Validated Usuario usuario) {  System.out.println("Hola: " + usuario.getNombreUsuario());  }  } |

Una vez se realice el controlador, se puede realizar la prueba enviando una solicitud **POST.**

|  |
| --- |
| {"nombreUsuario": "Ana", "edad": 120, "esFalso": true, "nacimiento": "21010-21-12"} |

La verificación que mostraría el resultado es:

* No es correcto el formato de la fecha
* Debe ser falso
* Debe ingresar un valor correcto para la edad

1. **SÍNTESIS**

A continuación, se presenta una síntesis de la temática estudiada en el componente formativo.

A screenshot of a computer screen

Description automatically generated

1. **ACTIVIDADES DIDÁCTICAS (Se debe incorporar mínimo 1, máximo 2)**

|  |  |
| --- | --- |
| DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA | |
| Nombre de la Actividad | Identificación de tecnologías Java |
| Objetivo de la actividad | Fortalecer los conocimientos sobre las principales tecnologías y *frameworks* utilizados en el desarrollo de aplicaciones web con Java. |
| Tipo de actividad sugerida | Relacionar conceptos |
| Archivo de la actividad  (Anexo donde se describe la actividad propuesta) | *CF020\_Actividad didáctica* |

1. **MATERIAL COMPLEMENTARIO:**

|  |  |  |  |
| --- | --- | --- | --- |
| Tema | Referencia APA del Material | Tipo de material  (Video, capítulo de libro, artículo, otro) | Enlace del Recurso o  Archivo del documento o material |
| Instalando React | Editor HTML gratuito en línea limpiador y convertidor. (2021, 27 de octubre). Editor HTML gratuito en línea limpiador y convertidor. | Página web | [htmled.it/or](https://htmled.it/or) |
| Instalando React | Cubic Factory. (2021, 27 de octubre). Cubic Factory. | Página web | [cubicfactory.com/jseditor](https://www.cubicfactory.com/jseditor) |
| Propiedades y estados de los componentes | CodeSandbox. (2021, 27 de octubre). CodeSandbox: Online Code Editor and IDE for Rapid Web Development. | Página web | [codesandbox.io](https://codesandbox.io/) |
| Crear un proyecto | React Playground. (2021, 27 de octubre). React Playground. | Página web | [playcode.io/react](https://playcode.io/react) |
| Hibernate | Hibernate Community Documentation. (s. f.). Capítulo 15. HQL: El lenguaje de consulta de Hibernate. | Sitio web | [docs.jboss.org/hibernate/orm/3.5/reference/es-ES/html/query](https://docs.jboss.org/hibernate/orm/3.5/reference/es-ES/html/queryhql.html) |

1. **GLOSARIO:**

*s*

|  |  |
| --- | --- |
| TÉRMINO | SIGNIFICADO |
| API: | conjunto de métodos ofrecidos por una librería que puede ser utilizado por otro *software* por medio de un conjunto de reglas establecidas. |
| CGI: | interfaz de entrada común se refiere a un sistema de comunicación que le indica al servidor web cómo enviar y recibir datos de una aplicación de servidor a un cliente. |
| CSS3: | hojas de estilo en cascada versión 3. |
| DOM: | modelo de objetos de un documento HTML. |
| ES6: | abreviación del estándar ECMA SCRIPT. Versión 6. |
| *Framework:* | es un marco o esquema de trabajo que define un conjunto de utilidades implementadas para un contexto particular para ser reutilizadas por los desarrolladores y así facilitar y acelerar el desarrollo de ciertas actividades propias de la construcción de *software.* |
| *Groovy:* | lenguaje de programación efectuado sobre la plataforma JAVA el cual permite realizar APIs de JAVA. |
| HQL: | lenguaje de consulta parecido al SQL implementado por el *framework Hibernate.* |
| HTML: | lenguaje de marcado hipertexto. |
| HTML: | HyperText Markup Language, lenguaje de Marcas de Hipertexto. |
| JAVASCRIPT: | lenguaje de programación. |
| JDBC: | API para JAVA que permite el acceso y manipulación de bases de datos relacionales. |
| JDK: | kit de Desarrollo de Java. |
| JPA: | API de persistencia de JAVA. |
| JSX: | sintaxis propuesta en React JS para escribir elementos de interfaz web. |
| Kotlin: | lenguaje de programación de tipado estático, sencillo y fácil de entender y aprender. |
| NODE.js: | es un servidor de aplicaciones. |
| NPM: | es un manejador de paquetes de sistema operativo. |
| *Plugin:* | programas pequeños que se anexan a otros para ofrecer alguna funcionalidad nueva. |
| SPA: | tipo de aplicación Web. |
| URL: | localizador uniforme de recursos, facilita la recuperación de información determinada en internet, se escribe en el navegador para ingresar a una página determinada. |

1. **REFERENCIAS BIBLIOGRÁFICAS:**

Berzal, F., Cubero, J., & Cortijo, F. (2007). Desarrollo profesional de aplicaciones web con Asp. net. <https://books.google.es/books?hl=es&lr=&id=J1d_9l6zlAIC&oi=fnd&pg=PA3&dq=servlests+y+jsp+fernando+berzal&ots=GsR9oMYn8M&sig=YaCnuz6ed8E_kEo_G1CPHH-KjjU>.

Gallagher, J. (2021, 4 de mayo). Los lenguajes de programación más populares. Career Karma. <https://careerkarma.com/blog/top-programming-languages-2021/>.

Gómez, M., & Cervantes, J. (2017). Introducción a la Programación Web con Java: JSP y Servlets JavaServer Faces.

Johnson, R. (2004). Expert one-on-one J2EE design and development. John Wiley & Sons.

Martínez, P. E. (2015). Hibernate. Persistencia de objetos en JEE (1ª ed.). Ra-Ma.

Node. (2021). Node JS Server. <https://nodejs.org/en/>.

React JS. (2021). React – A JavaScript library for building user interfaces. <https://reactjs.org/>.

Spring. (2021). Spring makes Java simple. <https://spring.io>.

Statistics Times. (2021). Principales idiomas informáticos. STATISTICSTIMES. <https://statisticstimes.com/tech/top-computer-languages.php>.

Capítulo 15. HQL: El lenguaje de consulta de Hibernate. (s. f.). Hibernate Community Documentation. <https://docs.jboss.org/hibernate/orm/3.5/reference/es-ES/html/queryhql.html>.

1. **CONTROL DEL DOCUMENTO**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia  *(Para el SENA indicar Regional y Centro de Formación)* | Fecha |
| Autor (es) | Carlos Hernán Muñoz Carvajal | Experto temático | Regional Distrito Capital, Grupo de Apoyo Administrativo - CGI | Noviembre 2022 |
| Zulema León Escobar | Experta temática | Centro de Teleinformática y Producción Industrial | Noviembre 2022 |
| Paola Alexandra Moya | Evaluadora instruccional | Regional Antioquia - Centro de Servicios de Salud | Mayo 2024 |
|  | Olga Constanza Bermúdez Jaimes | Responsable Línea de Producción Antioquia | Regional Antioquia - Centro de Servicios de Salud | Mayo 2024 |

1. **CONTROL DE CAMBIOS**

**(Diligenciar únicamente si realiza ajustes a la Unidad Temática)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia | Fecha | Razón del Cambio |
| Autor (es) |  |  |  |  |  |