

PROCESO DIRECCIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- Denominación del Programa de Formación: ANALISIS Y DESARROLLO DE SISTEMAS DE INFORMACION.
- Código del Programa de Formación: 228106
- Nombre del Proyecto:
- Fase del Proyecto: PLANEACIÓN
- Actividad de Proyecto: DISEÑAR LA ESTRUCTURA TECNOLÓGICA DEL SISTEMA INTEGRAL
- Competencia: PARTICIPAR EN EL PROCESO DE NEGOCIACIÓN DE TECNOLOGÍA INFORMÁTICA PARA PERMITIR LA IMPLEMENTACIÓN DEL SISTEMA DE INFORMACIÓN.
- Resultados de Aprendizaje: DEFINIR ESTRATEGIAS PARA LA ELABORACIÓN DE TÉRMINOS DE REFERENCIA Y PROCESOS DE EVALUACIÓN DE PROVEEDORES, EN LA ADQUISICIÓN DE TECNOLOGÍA, SEGÚN PROTOCOLOS ESTABLECIDOS.
- Duración de la Guía

2. PRESENTACIÓN

Para muchos, dominar la arquitectura de software es uno de los objetivos que han buscado durante algún tiempo, sin embargo, no siempre es claro el camino para dominarla, ya que la arquitectura de software es un concepto abstracto que cada persona lo interpreta de una forma diferente, dificultando con ello comprender y diseñar con éxito una arquitectura de software.

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

- Materiales:
Git, Spring Boot
Portatil ó Computador de Escritorio
- Ambiente Requerido
- Descripción de la(s) Actividad(es)

ESTILOS ARQUITETONICOS

Un estilo arquitectónico establece un marco de referencia a partir del cual es posible construir aplicaciones que comparten un conjunto de atributos y características mediante el cual es posible identificarlos y clasificarlos.

Representational State Transfer (REST)

REST se ha convertido sin lugar a duda en uno de los estilos arquitectónicos más utilizados en la industria, ya que es común ver que casi todas las aplicaciones tienen un API REST que nos permite interactuar con ellas por medio de servicios.

Lo primero que tenemos que saber es que REST es un conjunto de restricciones que crean un estilo arquitectónico y que es común utilizarse para crear aplicaciones distribuidas. REST fue nombrado por primera vez por Roy Fielding en el año 2000 donde definió a REST como:

REST proporciona un conjunto de restricciones arquitectónicas que, cuando se aplican como un todo, enfatizan la escalabilidad de las interacciones de los componentes, la generalidad de las interfaces, la implementación independiente de los componentes y los componentes intermedios para reducir la latencia de interacción, reforzar la seguridad y encapsular los sistemas heredados.
— Roy Fielding

Como se estructura REST

REST describe 3 conceptos clave, que son: Datos, Conectores y Componentes, los cuales trataremos de definir a continuación.

Datos

Uno de los aspectos más importantes que propone REST es que los datos deben de ser transmitidos a donde serán procesados, en lugar de que los datos sean transmitidos ya procesados, pero ¿qué quiere decir esto exactamente?

En cualquier arquitectura distribuida, son los componentes de negocio quien procesa la información y nos responde la información que se produjo de tal procesamiento, por ejemplo, una aplicación web tradicional, donde la vista es construida en el backend y nos responde con la página web ya creada, con los elementos HTML que la componente y los datos incrustados. En este sentido, la aplicación web proceso los datos en el backend, y como resultado nos arroja el resultado de dicho procesamiento, sin embargo, REST lo que propone es que los datos sean enviados en bruto para que sea el interesado de los datos el que los procese para generar la página (o cualquier otra cosa que tenga que generar), en este sentido, lo que REST propone no es generar la página web, si no que mandar los datos en bruto para que el consumidor sea el responsable de generar la página a través de los datos que responde REST.

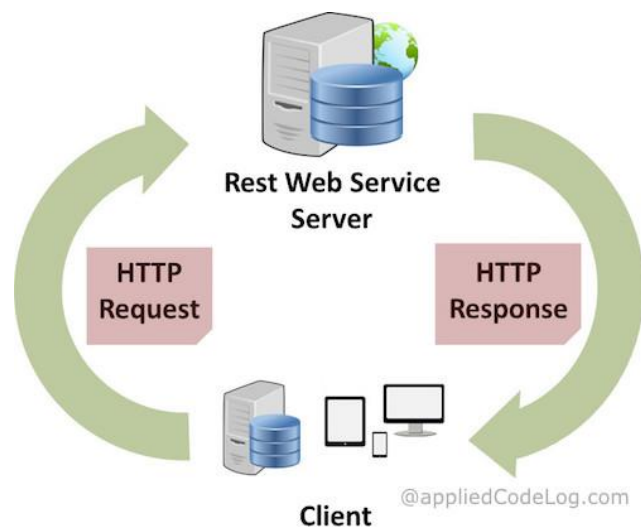


Imagen 6 – Procesamiento

En la imagen anterior podemos apreciar más claramente lo que comentábamos anteriormente, en el cual, los datos son procesados por el servidor y como resultado se obtiene un documento HTML procesado con todos los datos a mostrar.

Este enfoque fue utilizado durante mucho tiempo, pues las aplicaciones web requerían ir al servidor a consultar la siguiente página y este les regresa el HTML que el navegador solamente tenía que renderizar, sin embargo, que pasa con nuevas tecnologías como Angular, Vue, o React que se caracterizan por generar la siguiente página directamente en el navegador, lo que significa que no requieren ir al servidor por la siguiente página y en su lugar, solo requieren los datos del servidor presentarlo al usuario.

Bajo este enfoque podemos entender que REST propone servir los datos en crudo para que sea el consumidor quien se encargue de procesar los datos y mostrarlo como mejor le convenga.

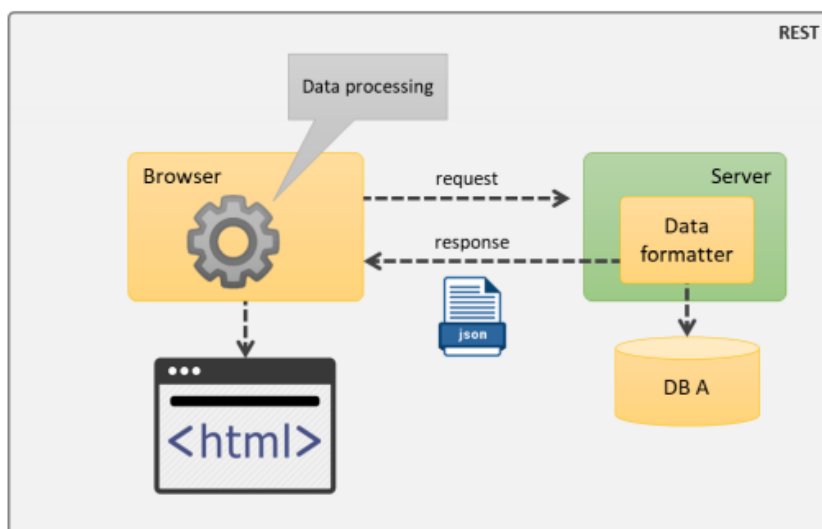


Imagen 7 – Procesamiento del lado del cliente

La imagen anterior representa mejor lo que acabamos de decir, pues ya vemos que el servidor solo se limita a servir la información en un formato esperado por el cliente, el cual por lo general es JSON, aunque podría existen más formatos. Por otra parte, el cliente recibe esa información, la procesa y genera la vista a partir de los datos proporcionados por el servidor, de esta forma, varias aplicaciones ya sean Web, Móviles, escritorio, o en la nube pueden solicitar esa misma información y mostrarla de diferentes formas, sin estar limitados por el formato procesado que regresa el servidor, como sería el caso de una página HTML.

Ahora bien, el caso de la página HTML es solo un ejemplo, pero se podría aplicar para cualquier otra cosa que no sea una página, como mandar los datos a una app móvil, imágenes o enviarlos a otro servicio para su procesamiento, etc.

Recursos

En REST se utiliza la palabra Recurso para hacer referencia a la información, y de esta forma, todos lo que podamos nombrar puede ser un recurso, como un documento, imagen, servicios, una colección de recursos, etc. En otras palabras, un recurso puede ser cualquier cosa que pueda ser recuperada por una referencia de hipertexto (URL)

Cada recurso tiene una dirección única que permite hacerle referencia, de tal forma que ningún otro recurso puede compartir la misma referencia, algo muy parecido con la web, donde cada página tiene un dominio y no pueden existir dos páginas web en el mismo dominio, o dos imágenes en la misma URL, de la misma forma, en REST cada recurso tiene su propia URL o referencia que hace posible recuperar un recurso de forma inconfundible.

Representaciones

Una de las características de los recursos es que pueden tener diferentes representaciones, es decir, que un mismo recurso puede tener diferentes formatos, por ejemplo, una imagen que está representada por un recurso (URL) puede estar en formato PNG o JPG en la misma dirección, con la única diferencia de que los metadatos que describen el recurso cambian según la representación del recurso.

En este sentido, podemos decir que una representación consta de datos y metadatos que describen los datos. Los metadatos tienen una estructura de nombre-valor, donde el nombre describe el metadato y el valor corresponde al valor asignado al metadato. Por ejemplo, un metadato para representar una representación en formato JSON sería “Content-Type”=“application/json” y el mismo recurso en formato XML sería “Content-Type”=“application/xml”.

Conectores

Los conectores son los componentes que encapsulan la actividad de acceso a los recursos y transferencia, de esta forma, REST describe los conectores como interfaces abstractas que permite la comunicación entre componentes, mejorando la simplicidad al proporcionar una separación clara de las preocupaciones y ocultando la implementación subyacente de los recursos y los mecanismos de comunicación.

Por otro lado, REST describe que todas las solicitudes a los conectores sean sin estado, es decir, que deberán contener toda la información necesaria para que el conector comprenda la solicitud sin importar las llamadas que se hallan echo en el pasado, de esta forma, ante una misma solicitud deberíamos de obtener la misma respuesta. Esta restricción cumple cuatro funciones:

1. Eliminar cualquier necesidad de que los conectores retengan el estado de la aplicación entre solicitudes, lo que reduce el consumo de recursos físicos y mejora la escalabilidad.
2. Permitir que las interacciones se procesen en paralelo sin requerir que el mecanismo de procesamiento entienda la semántica de la interacción.
3. Permite a un intermediario ver y comprender una solicitud de forma aislada, lo que puede ser necesario cuando los servicios se reorganizan dinámicamente.
4. Facilita el almacenamiento en caché, lo que ayuda a reutilizar respuestas anteriores.

En REST existen dos tipos de conectores principales, el cliente y el servidor, donde el cliente es quien inicial la comunicación enviando una solicitud al servidor, mientras que el servidor es el que escucha las peticiones de los clientes y responde las solicitudes para dar acceso a sus servicios.

Otro tipo de conector es el conector de caché, el cual puede estar ubicado en la interfaz de un conector cliente o servidor, el cual tiene como propósito almacenar las respuestas y reutilizarlas en llamadas iguales en el futuro, evitando con ello, solicitar recursos consultados anteriormente al servidor. El cliente puede evitar hacer llamadas adicionales al servidor, mientras que el servidor puede evitar sobrecargar a la base de datos, disco o procesador con consultas repetitivas. En ambos casos se mejora la latencia.

Componentes

Los componentes con software concretos que utilizan los conectores para consultar los recursos o servirlos, ya sea una aplicación cliente como lo son los navegadores (Chrome, Firefox, etc) o Web Servers como Apache, IIS, etc. Pero también existen los componentes intermedios, que actúan como cliente y servidor al mismo tiempo, como es el caso de un proxy o túnel de red, los cuales actúan como servidor al aceptar solicitudes, pero también como cliente a redireccionar las peticiones a otro servidor.

Los componentes se pueden confundir con los conectores, sin embargo, un conector es una interface abstracta que describe la forma en que se deben de comunicar los componentes, mientras que un componente es una pieza de software concreta que utiliza los conectores para establecer la comunicación.

Características de REST

Analizaremos las características que distinguen al estilo REST del resto. Las características no son ventajas o desventajas, si no que más bien, nos sirven para identificar cuando una aplicación sigue un determinado estilo arquitectónico.

Las características se pueden convertir en ventajas o desventajas solo cuando analizamos la problemática que vamos a resolver, mientras tanto, son solo características:

1. Todos los recursos deben de poder ser nombrados, es decir, que tiene una dirección (URL) única que la diferencia de los demás recursos.
2. Un recurso puede tener más de una representación, donde un recurso puede estar representado en diferentes formatos al mismo tiempo y en la misma dirección.
3. Los datos están acompañados de metadatos que describen el contenido de los datos.
4. Las solicitudes tienen toda la información para poder ser entendidas por el servidor, lo que implica que podrán ser atendidas sin importar las invocaciones pasadas (sin estado) ni el contexto de la misma.
5. Debido a la naturaleza sin estado de REST, es posible cachear las peticiones, de tal forma que podemos retornar un resultado previamente consultado sin necesidad de llamar al servidor o la base de datos.
6. REST es interoperable, lo que implica que no está casado con un lenguaje de programación o tecnología específica.

RESTful y su relación con REST

Sin lugar a duda, uno de los conceptos que más se confunden cuando hablamos de arquitecturas REST, es el concepto de RESTful, pues aún que podrían resultar similares, tiene diferencias fundamentales que las hacen distintas.

Como ya habíamos mencionado antes, REST parte de la idea de que tenemos datos que pueden tener diferentes representaciones y los datos siempre están acompañados de metadatos que describen el contenido de los datos, además, existen conectores que permiten que se pueda establecer una conexión, los cuales son abstractos desde su definición, finalmente, existen los componentes que son piezas de software concretas que utilizan los conectores para transmitir los datos.

Por otra parte, RESTfull toma las bases de REST para crear servicios web mediante el protocolo HTTP, de esta forma, podemos ver que los conectores ya no son abstractos, si no que delimita que la comunicación siempre tendrá que ser por medio de HTTP, además, define que la forma para recuperar los datos es por medio de los diferentes métodos que ofrece HTTP, como lo son GET, POST, DELETE, PUT, PATCH. En este sentido, podemos observar que REST jamás define que protocolo utilizar, y mucho menos que debemos utilizar los métodos de HTTP para realizar las diferentes operaciones.

Entonces, podemos decir que RESTful promueve la creación de servicios basados en las restricciones de REST, utilizando como protocolo de comunicación HTTP.

También podemos observar que los servicios RESTful mantiene las propiedades de los datos que define REST, ya que en RESTful, todos los recursos deben de poder ser nombrados, es decir, deberán tener un URL única que pueda ser alcanzada por HTTP, por otra parte, define una serie de metadatos que describen el contenido de los datos, es por ello que con RESTful podemos retornar un XML, JSON, Imagen, CSS, HTML, etc. Cualquier cosa que pueda ser descrita por un metadato, puede ser retornada por REST.

Ventajas

- Interoperable: REST no está casado con una tecnología, por lo que es posible implementarla en cualquier lenguaje de programación.
- Escalabilidad: Debido a todas las peticiones son sin estado, es posible agregar nuevos nodos para distribuir la carga entre varios servidores.
- Reducción del acoplamiento: Debido a que los conectores definen interfaces abstractas y que los recursos son accedidos por una URL, es posible mantener un bajo acoplamiento entre los componentes.
- Testabilidad: Debido a que todas las peticiones tienen toda la información y que son sin estado, es posible probar los recursos de forma individual y probar su funcionalidad por separado.
- Reutilización: Una de los principios de REST es llevar los datos sin procesar al cliente, para que sea el cliente quién decida cuál es la mejor forma de representar los datos al usuario, por este motivo, los recursos de REST pueden ser reutilizado por otros componentes y representar los datos de diferentes maneras.

Desventajas

- Performance: Como en todas las arquitecturas distribuidas, el costo en el performance es una constante, ya que cada solicitud implica costos en latencia, conexión y autenticaciones.
- Más puntos de falla: Nuevamente, las arquitecturas distribuidas traen con siqa ciertas problemáticas, como es el caso de los puntos de falla, ya que al haber múltiples componentes que conforman la arquitectura, multiplica los puntos de falla.
- Gobierno: Hablando exclusivamente desde el punto de vista de servicios basados en REST, es necesario mantener un gobierno sobre los servicios implementados para llevar un orden, ya que es común que diferentes equipos creen servicios similares, provocando funcionalidad repetida que tiene que administrarse.

Cuando debo de utilizar el estilo REST

Actualmente REST es la base sobre la que están construidas casi todas las API de las principales empresas del mundo, con Google, Slack, Twitter, Facebook, Amazon, Apple, etc, incluso algunas otras como Paypal han migrado de los Webservices tradicionales (SOAP) para migrar a API REST.

En este sentido, REST puede ser utilizado para crear integraciones con diferentes dispositivos, incluso con empresas o proveedores externos, que requieren información de nuestros sistemas para opera con nosotros. Por ejemplo, mediante API REST, Twitter logra que existan aplicaciones que puedan administrar nuestra cuenta, pueda mandar Tweets, puedan seguir personas, puedan retwittear y todo esto sin entrar a Twitter. Otro ejemplo, es Slack, que nos permite mandar mensaje a canales, enviar notificaciones a un grupo si algo pasa en una aplicación externa, como Trello o Jira.

En Google Maps, podemos ir guardando nuestra posición cada X tiempo, podemos analizar rutas, analizar el tráfico, tiempo de llega a un destino, todo esto mediante el API REST, y sin necesidad de entrar a Google Maps para consultar la información.

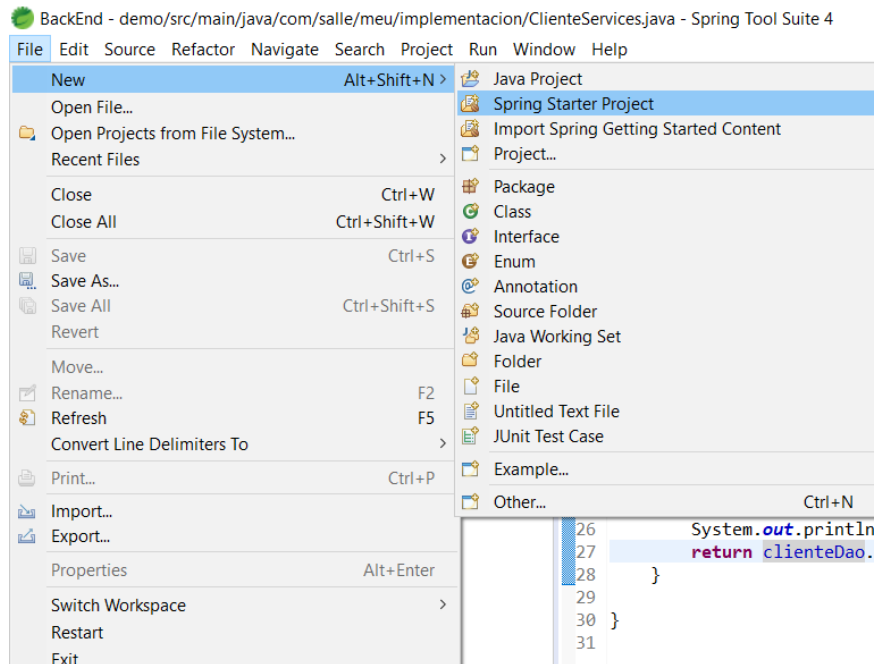
En otras palabras, REST se ha convertido prácticamente en la Arquitectura por defecto para integrar aplicaciones, incluso ha desplazado casi en su totalidad a los Webservices tradicionales (SOAP).

Entonces, cuando debo de utilizar REST, la respuesta es fácil, cuando necesitamos crear un API que nos permite interactuar con otra plataformas, tecnologías o aplicaciones, sin importar la tecnología y que necesitemos tener múltiples representaciones de los datos, como consultar información en formato JSON, consultar imágenes en PNG o JPG, envías formularios directamente desde un formulario HTML o simplemente enviar información binaria de un punto a otro, ya que REST permite describir el formato de la información mediante metadatos y no está casado un solo formato como SOAP, el cual solo puede enviar mensajes en XML.

EJERCICIO

Suba el servicio de MySQL, abrir el editor de mysql y cree un esquema llamado proyecto.

Crear un proyecto en spring boot

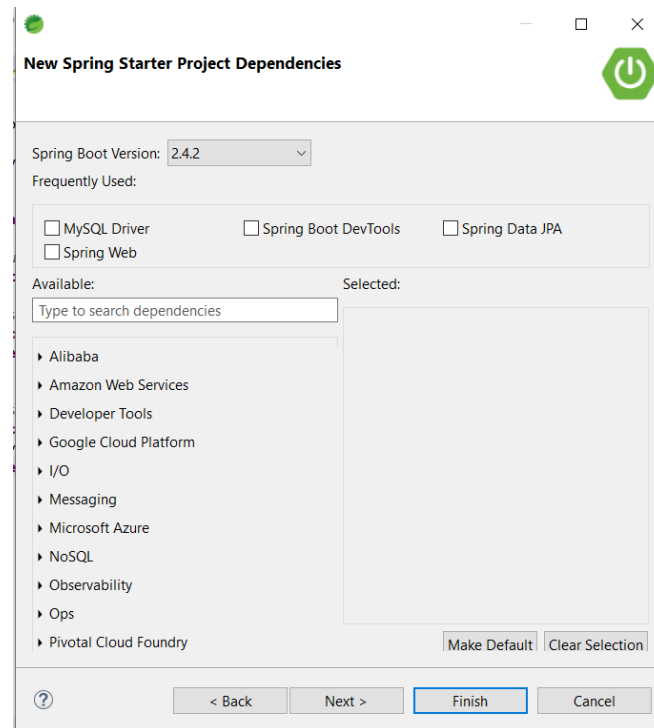


Diligenciar los campos requeridos para el proyecto como lo es name, group, artifact, etc

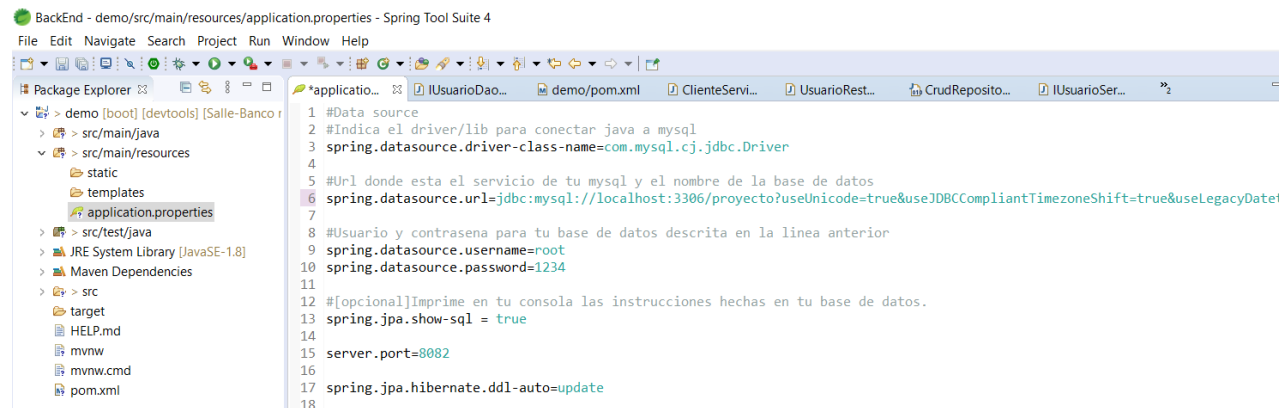
The 'New Spring Starter Project' dialog box is shown with the following fields and values:

- Service URL: <https://start.spring.io>
- Name: Proyecto
- ☒ Use default location
- Location: D:\Nestor\Desarrollo\Java\Salle-Banco\BackEnd\Proyecto
- Type: Maven
- Packaging: Jar
- Java Version: 8
- Language: Java
- Group: com.salle
- Artifact: salle
- Version: 0.0.1-SNAPSHOT
- Description: Demo project for Spring Boot
- Package: com.salle.menu
- Working sets: ☐ Add project to working sets
- Buttons: < Back, Next >, Finish, Cancel

Agregue las dependencias de mysql driver, spring boot dev tools, spring data JPA y spring web.



Una vez creado el proyecto, agregue en el archivo application.properties la siguiente sintaxis (agregar el usuario, el password y el puerto de configuración de mysql de su pc):



`spring.datasource.url=jdbc:mysql://localhost:3306/proyecto?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC`

Crear un package en src\main\java con nombre entities y cree 3 clases llamadas Rol, Dependencia y Usuario.

En la entidad Rol, agregue los atributos, métodos y constructor

```

...
@Entity
@Table( name = "ROL" )

```

```
public class Rol implements Serializable{

    //-----
    //Constantes de la Clase.
    //-----

    /**
     * Constante de Serialización.
     */
    private static final long serialVersionUID = -6362794385792247263L;

    //-----
    //Atributos de la Clase.
    //-----

    /**
     * Identificador del Rol.
     */
    private int id;

    /**
     * Descripción del rol
     */
    private String descripcion;

    /**
     * Sigla que sirve como constante para hacer validaciones en el código.
     */
    private String siglaRol;

    /**
     * Estado del rol
     */
    private boolean estado;

    //-----
    //Constructor de la Clase.
    //-----

    /**
     * Constructor por Defecto.
     */
    public Rol( ) {
    }
}
```



```
//-----  
//Métodos de la Clase.  
//-----  
  
/**  
 * @return the idRol  
 */  
@Id  
@-( strategy = GenerationType.IDENTITY )  
@Column( name = "ID_ROL_PK", nullable = false )  
public int getId( ) {  
    return id;  
}  
  
/**  
 * @param idRol the idRol to set  
 */  
public void setId( int idRol ) {  
    this.id = idRol;  
}  
  
/**  
 * @return the descripcion  
 */  
@Column( name = "DESCRIPCION", nullable = false )  
public String getDescripcion( ) {  
    return descripcion;  
}  
  
/**  
 * @param descripcion the descripcion to set  
 */  
public void setDescripcion( String descripcion ) {  
    this.descripcion = descripcion;  
}  
  
/**  
 * @return the siglaRol  
 */  
@Column( name = "SIGLA_ROL" )  
public String getSiglaRol( ) {  
    return siglaRol;  
}  
  
/**
```



```
    * @param siglaRol the siglaRol to set
    */
    public void setSiglaRol(String siglaRol) {
        this.siglaRol = siglaRol;
    }

    /**
     * @return the estado
     */
    @Column( name = "ESTADO", nullable = false )
    public boolean getEstado( ) {
        return estado;
    }

    /**
     * @param estado the estado to set
     */
    public void setEstado( boolean estado ) {
        this.estado = estado;
    }

    /* (non-Javadoc)
     * @see java.lang.Object#toString()
     */
    @Override
    public String toString() {
        return "Rol [id=" + id + ", descripcion=" + descripcion + ", siglaRol="
            + siglaRol + ", estado=" + estado + "]";
    }

}
```

En la entidad Dependencia, realice las mismas acciones de Rol pero con los siguientes atributos

```
/**
 * Identificador de la entidad Dependencias
 */
private int id;

/**
 * Codigo de la dependencia
```

```

*/
private String codigo;

/**
 * Descripción de la dependencia
 */
private String descripcion;

/**
 * Cargo que dirige la dependencia.
 */
private String cargo;

/**
 * Estado de la dependencia
 */
private boolean estado;

```

Adicione los métodos, el constructor y el toString. Adicional, inyecte la tabla de nombre “DEPENDENCIAS”

Posteriormente, agregue la entidad usuario con las siguientes características:

```

@Entity
@Table( name = "USUARIOS" )
public class Usuario implements Serializable{

    //-----
    //Constantes de la clase.
    //-----

    /**
     * Constante de serialización.
     */
    private static final long serialVersionUID = 1265660971450473499L;

    //-----
    //Atributos de la clase.
    //-----

    /**
     * Identificador de la entidad UsuarioBO
     */
    private int id;

```

```
/**
 * Numero de documento de la persona
 */
private String documento;

/**
 * Nombre de usuario en el sistema
 */
private String userName;

/**
 * Nombre de la persona
 */
private String nombre;

/**
 * Email de la persona
 */
private String mail;

/**
 * Estado del usuario
 */
private boolean estado;

/**
 * RolBO al que pertenece este usuario
 */
private Rol rol;

/**
 * DependenciaBO a la que pertenece el usuario.
 */
private Dependencia dependencia;

//-----
//      //Constructor de la clase.
//-----

/**
 * Constructor por defecto.
 */
public Usuario( ) {

}
```



```
//-----  
    //Metodos de la clase.  
//-----  
  
/**  
 * @return the id  
 */  
@Id  
@GeneratedValue( strategy = GenerationType.IDENTITY)  
@Column( name = "ID_USUARIO_PK", nullable = false )  
public int getId() {  
    return id;  
}  
  
/**  
 * @param id the id to set  
 */  
public void setId(int id) {  
    this.id = id;  
}  
  
/**  
 * @return the documento  
 */  
@Column( name = "DOCUMENTO", nullable = false )  
public String getDocumento() {  
    return documento;  
}  
  
/**  
 * @param documento the documento to set  
 */  
public void setDocumento(String documento) {  
    this.documento = documento;  
}  
  
/**  
 * @return the userName  
 */  
@Column( name = "USERNAME", nullable = false, unique = true )  
public String getUserName() {  
    return userName;  
}  
  
/**
```




```

    * @param userName the userName to set
    */
    public void setUserName(String userName) {
        this.userName = userName;
    }

    /**
     * @return the nombre
     */
    @Column( name = "NOMBRE", nullable = false )
    public String getNombre() {
        return nombre;
    }

    /**
     * @param nombre the nombre to set
     */
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    /**
     * @return the mail
     */
    @Column( name = "EMAIL", nullable = false )
    public String getMail() {
        return mail;
    }

    /**
     * @param mail the mail to set
     */
    public void setMail(String mail) {
        this.mail = mail;
    }

    /**
     * @return the estado
     */
    @Column( name = "ESTADO", nullable = false )
    public boolean isEstado() {
        return estado;
    }

    /**
     * @param estado the estado to set
     */

```



```
public void setEstado(boolean estado) {
    this.estado = estado;
}

/**
 * @return the rol
 */
@OneToOne
@JoinColumn( name = "ID_ROL_FK", nullable = false )
public Rol getRol() {
    return rol;
}

/**
 * @param rol the rol to set
 */
public void setRol(Rol rol) {
    this.rol = rol;
}

/**
 * @return the dependencia
 */
@ManyToOne( fetch = FetchType.EAGER )
@JoinColumn( name = "ID_DEPENDENCIA_FK", nullable = false )
public Dependencia getDependencia( ) {
    return dependencia;
}

/**
 * @param dependencia the dependencia to set
 */
public void setDependencia( Dependencia dependencia ) {
    this.dependencia = dependencia;
}

/* (non-Javadoc)
 * @see java.lang.Object#toString()
 */
@Override
public String toString() {
    return "UsuarioBO [id=" + id + ", documento=" + documento + ", userName="
        + userName + ", nombre=" + nombre + ", mail=" + mail
        + ", estado=" + estado + ", rol=" + rol + ", dependencia="
        + dependencia + " ]";
}
```

```
}
```

Crear el package de nombre dao y en el cree una interface de nombre IUserdao, el cual, debe quedar con la siguiente sintaxis

```
import org.springframework.data.repository.CrudRepository;

import com.salle.menu.entities.Usuario;

public interface IUserdao extends CrudRepository<Usuario, Long> {

}
```

Cree un package llamado implementación y en agregue una clase llamada clienteServices con la siguiente sintaxis:

```
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.salle.meu.dao.IUserdao;
import com.salle.meu.entities.Usuario;
import com.salle.meu.services.IUsuarioServices;
```

```
@Service
public class ClienteServices implements IUsuarioServices{

    @Autowired
    private IUserdao clienteDao;

    @Transactional(readOnly = true)
    public List<Usuario> findAll() {
        return (List<Usuario>) clienteDao.findAll();
    }

    @Transactional(readOnly = false)
    public Usuario create(Usuario usuario){
        System.out.println(usuario.getDependencia());
        return clienteDao.save(usuario);
    }

}
```

Cree un package llamado services y agregue una interface llamada IUsuarioServices y en el agregue la siguiente sintaxis:

```
import java.util.List;
```

```
import com.salle.meu.entities.Usuario;

public interface IUserioServices {

    public List<Usuario> findAll();

    public Usuario create(Usuario usuario);

}
```

Cree un paquete llamado controller y en el cree una clase con la siguiente sintaxis:

```
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.salle.meu.entities.Usuario;
import com.salle.meu.services.IUsuarioServices;

@RestController
@RequestMapping("/api")
public class UsuarioRestController {

    @Autowired
    private IUserioServices usuarioService;

    @GetMapping("/usuario")
    public List<Usuario> index(){
        return usuarioService.findAll();
    }

    @PostMapping("/usuario")
    public Usuario create(@RequestBody Usuario usuario){
        return usuarioService.create(usuario);
    }

}
```

Corra el proyecto que debe salir sin errores, abra el editor de Mysql y busque el esquema proyecto, allí agregue datos en las tablas rol, dependencia y usuario. Posteriormente, abra postman en la url indicada por el servidor en el puerto que configuro en el application.properties y realice un get de todos los usuarios y de un usuario específico por id.

Realice el Post y el put en java y pruébelo en postman.

4. ACTIVIDADES DE EVALUACIÓN

Evidencia de Conocimiento: Recopile los conceptos más importantes de la arquitectura REST en una infografía.

Evidencia de Desempeño: Describa cual(es) patrón(es) arquitectónico(s) tiene el proyecto realizado.

Evidencia de Producto: Entrega del mapa conceptual y del repositorio del proyecto en git hub. Debe hacer un commit cada que realice la creación de un package.

Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación
Evidencias de Conocimiento:	Reconoce las principales características de la arquitectura rest	Recopilación
Evidencias de Desempeño:	Interpreta el uso de una aplicación rest en un entorno empresarial	Desarrollo
Evidencias de Producto	Realiza el entregable con la documentación completa	Entrega de la guía con todas las evidencias requeridas.

1. GLOSARIO DE TÉRMINOS

De acuerdo a la práctica realizar su propio glosario de términos.

6. REFERENTES BIBLIOGRÁFICOS

Construya o cite documentos de apoyo para el desarrollo de la guía, según lo establecido en la guía de desarrollo curricular

7. CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)	Néstor Rodríguez	Instructor	Teleinformática	DICIEMBRE-2020

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)



	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					