|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Aplicaciones Distribuidas*** | | | | |
| *Laboratorio REST, JPA* | | | | |
| **Integrantes**  *CHRISTIAN TORRES* | | | **Fecha de realización de la práctica**  23/01/2017 | |
| **Carrera**  Ingeniería Informática y Ciencias de la Computación | **Curso**  *7 A* | **Grupo** | | **Fecha de presentación del informe**  **23/01/2017** |
| **Objetivos:**  Desarrolle un servicio web usando REST y JPA.  **Bases conceptuales:**   * REST * JPA   **Material:**   * Java * Notas de clase   **Método, Instrucciones y actividades:**  El servidor guarda en una tabla dentro de la base de datos "sample" la siguiente información:   * id del dato * nombre de usuario, ej: diego, pedro, maria (una sola palabra) * tipo del dato (una sola palabra, ej: nota, cita, dirección,...) * dato (cualquier contenido que se guardará como texto)   Para iniciar creamos una aplicación web, la que funcionará como nuestro servidor dentro de nuestro pequeño sistema, vamos a file->new project->web application, al concluir estos pasos debemos agregar nuestro path para consumir los servicios web, de la siguiente manera:    Creamos una clase java con el nombre Jpa.java en la que construi las entidades que vamos usar en la base de datos de la siguiente manera:    hay que tomar muy en cuenta que a mas de agregar el campo @Entity, tenemos que agregar los constructores de cada variable.    Al momento de terminar de programar todo el entity que funcionara como nuestra base de datos, nos pedirá agregar un archivo persistence.xml donde tendremos que agregar una pequeña información básica de la siguiente manera:    Ahora para darle funcionalidad a nuestra aplicacion REST, creamos nuevamente un archivo java, llamado Services.java (o cómo usted quiera) donde tendremos nuestras funciones GET, POST, PUT, DELETE, que serviran para hacer nuestro CRUD, lo hacemos de la siguiente manera:    agregamos @Stateless que nos sirve de protocolo de comunicaciones que trata cada petición como una transacción independiente que no tiene relación con cualquier solicitud anterior, de modo que la comunicación se compone de pares independientes de *solicitud* y *respuesta*.  También agregamos @Path que nos sirve para consumir nuestro servicio web por medio de la URL, de ahí a continuación tenemos todas las funciones @GET, @POST, @PUT, @DELETE, que como ya lo comentamos nos ayudará a realizar nuestro CRUD.      Ahí podríamos dar por culminada la parte del servidor.  Parte del cliente  Creamos unas ventanas básicas para ir asignando las respectivas funciones a cada una, lo hice de las siguiente manera:  Menú Principal y la pantalla de index:    Menú de CRUD:    Pantalla para Crear:    Pantalla para retrieve:    Pantalla para Update:    Pantalla para Deleted:    Ahora por medio de los botones de cada pantalla llamaremos al servicio web, esperando información de retorno, lo hice con llamadas por URL.  La funcion del boton CREATE:    la variable url va contener nuestro path hacia el servicio REST que estamos llamando, la variable contendrá la conexión a nuestro Url, es importante poner el setRequestMethod, para saber cual es el método que estamos llamando en este caso es el método de POST, en el setRequestProperty, vamos a enviar nuestro valor de escritura, en este caso es un texto plano, el getOutputStream nos servirá para “escribir en la URL”, en este caso de post, vamos a crear un nuevo usuario.  Después de nuestra explicación agregaremos las siguientes funciones a nuestros botones, para realizar el CRUD.  Métodos para deleted:    Métodos para retrieve:  para toda la información:    mostrar el campo dato, consulta por medio de ID    Mostrar la información por nombre de Usuario    Método para update:    **Conclusiones:**   * XML sobre HTTP es una interfaz muy poderosa que permite que aplicaciones internas, como interfaces basadas en JavaScript Asincrónico + XML (AJAX) puedan conectarse, ubicar y consumir recursos. De hecho, es justamente esta gran combinación con AJAX que generó esta gran atención que tiene REST hoy en día. * Resulta muy flexible el poder exponer los recursos del sistema con un API REST, de manera de brindar datos a distintas aplicaciones, formateados en distintas maneras. REST ayuda a cumplir con los requerimientos de integración que son críticos para construir sistemas en donde los datos tienen que poder combinarse fácilmente (mashups) y extenderse. Desde este punto de vista, los servicios REST se convierten en algo mucho más grande.   **Recomendaciones:**   * No siempre REST es la mejor opción. Está surgiendo como una alternativa para diseñar servicios web con menos dependencia en middleware propietario (por ejemplo, un servidor de aplicaciones), que su contraparte SOAP y los servicios basados en WSDL. De algún modo, REST es la vuelta a la Web antes de la aparición de los grandes servidores de aplicaciones, ya que hace énfasis en los primeros estándares de Internet, URI y HTTP. | | | | |