

Tecnologías para la integración de soluciones

Servicio Web Pastelería

Consultar productos

Contreras Lezama Jonathan

Cuervo Sánchez Freddy Sahid

Pérez Cortina Denilson

Doc. Rojano Cáceres José Rafael

20 de marzo de 2022



Contenido

Introducción	2
Motivación.....	3
Problemática	4
Solución.....	4
Servicio de productos.....	4
Servicio de promociones.....	5
Servicios pedidos.....	5
Costos	6
Github.....	6
Heroku.....	6
Docker	6
Visual Studio Code	6
Clever cloud.....	6
Spring boot	6
EndPoint.....	8
Producto.....	8
Promoción.....	11
Parámetros de recepción y Devueltos.....	16
Microservicio productos.....	16
Microservicio promociones.....	19
Microservicio cliente-pedido.....	21
Plan de pruebas.....	23
Servicio productos.....	24
Servicio promociones.....	25
Servicio clientepedido.....	27
Dockerfile microservicio cliente-pedido.....	29
Dockerfile microservicio promociones.....	29
Dockerfile microservicio productos.....	29
Github.....	29
Heroku.....	29

Introducción

Los servicios web o web services son herramientas necesarias para el intercambio de datos entre diferentes aplicaciones, su uso va más allá de algo simple y fácil. Nos permite una clase de comunicación entre múltiples servicios. Con ello, un usuario es capaz de realizar tareas específicas para hacer uso de los diversos microservicios que conforman un servicio.

En los webs services podemos hacer uso de los servicios, aunque un solo servicio es más que eso, debido a su construcción con microservicios. Con su ayuda se crean diferentes partes que al unisonó, puede crear los servicios necesarios para realizar tareas específicas, ejemplos tan claros como el poder iniciar sesión en alguna cuenta de una red social.

En nuestro contexto, es posible crear un servicio para un cliente de una pastelería, el servicio consiste en que un cliente, es capaz de consultar los productos en existencia dentro de la pastelería. Aunque dicho de esa manera no representa un reto, pero dentro de los microservicios encontramos el cómo se puede ingresar un nuevo producto a la pastelería para el gusto del cliente, modificar los datos de un producto (llámese descripción, nombre, precio, entre otros) y el eliminar un producto necesario; siendo microservicios que complementan un funcionamiento óptimo para el servicio completo.

Pero como se ha afirmado, un servicio puede hacer uso de los datos de otro servicio, para ello, un segundo servicio se encarga de mostrar promociones de diferentes postres, siendo un complemento ayuda del motivo del servicio, pero de igual manera es necesario realizar diferentes microservicios, para mostrar las promociones es requerido que se ingresen o registren, que se modifiquen las promociones creadas anteriormente y el poder eliminar promociones que ya no se requieran.

Por último, un servicio necesario, vital para lograr el objetivo general del servicio web, es el poder general un servicio de pedidos, en el cual, un cliente sea capaz de registrar un pedido con el producto de su agrado.

Motivación

Debido a la pandemia culpa de SARS-CoV-2, muchos negocios tuvieron que suspender sus trabajos diarios por un tiempo indefinido, algunos de ellos se vieron en la necesidad de cerrar sus locales, siendo los emprendedores, los negocios pequeños y medianos los más afectados. En el caso de los reposteros, algunos de ellos se vieron en la necesidad de realizar trabajos a pedidos o a domicilios, para mantener a flote su economía personal.

El servicio, pensado para una pastelería que busca ayudar al dueño a atraer más cliente a su pequeño emprendimiento, puede ser usado para otros negocios con los mismos intereses solo haciendo pequeñas variaciones que no representen un alto costo en tiempo y esfuerzo, esto para buscar cubrir las necesidades en sus negocios.

Problemática

El dueño de la pastelería se preocupa por el alcance de su recién iniciado negocio, sabe que no puede realizar sus pasteles y al mismo tiempo intentar buscar nuevos clientes, por ello, tuvo la idea y la necesidad de un servicio web el cual sea posible llegar a nuevos clientes que sean capaces de conocer de su trabajo y servicio en la repostería. Quiere que se haga uso de información sencilla pero que para sus nuevos clientes sean capaces de interesarse por sus productos pero que sea lo suficiente clara para que no tenga que invertir tiempo en tratar de responder las dudas extra de sus clientes y solo enfocarse en las necesarias para que su negocio despegue.

Intentado llamar la atención de nuevos clientes con ayuda de promociones que vayan de acuerdo con eventos en donde hay una necesidad de un producto de una repostería. También buscando la forma de que un cliente, realice un pedido de manera sencilla, la cual cumple el propósito de mejorar su experiencia al realizar una compra de un producto, facilitando el trabajo de nuestro emprendedor.

Solución

Para la solución se realizó un servicio API el cuál está dividido en servicios y estos a su vez en microservicios.

Servicio de productos.

Para el primer servicio, fue necesario hacer uso de SOAP, con el cual es posible crear los microservicios necesarios. Al momento de ingresar los datos de un producto, es necesario registrar el tipo de producto, la descripción de este y el precio, siendo esto el primer microservicio. Para el segundo microservicio, se debe poder eliminar los productos por medio del identificador de los productos antes creados y un microservicio enfocado en la modificación de los productos, permitiendo modificar el tipo de producto, la descripción y el precio. Estos microservicios son necesarios para cumplir con el objetivo del servicio, que es el poder consultar los productos de la pastelería existentes.

Servicio de promociones.

Este microservicio también se realizó por medio de SOAP. Al momento de ingresar la promoción se debe escribir el nombre de la promoción, el día de la semana que es válida, y el precio que se hará a esa promoción. Para el segundo microservicio se debe poder eliminar alguna promoción en caso de ya no ser necesaria, pero si solo habrá un cambio en la promoción existe la opción de modificarla. Finalmente, el último microservicio permite mostrar todas las promociones que haya.

Servicios pedidos.

Este microservicio se realizó con RES, El primer microservicio permite registrar el cliente al que se le hará el pedido, el segundo microservicio permite mostrar los clientes. Con el tercer microservicio se puede eliminar al cliente. El cuarto microservicio permite registrar un pedido, el cual obtiene datos del producto, cliente y promoción. El último microservicio nos permite obtener los pedidos realizados.

Costos

Las tecnologías para realizar nuestra API fueron gratuitas, por lo cual no hubo costos.

Las tecnologías son:

Github

Utilizamos Github para crear nuestro repositorio y poder compartir cambios y avances entre los integrantes del equipo. Gracias a la tecnología pudimos trabajar de manera separada y aún así obtener los cambios de los demás integrantes.

Heroku

Heroku nos proporcionó la tecnología para subir de manera gratuita los microservicios a la nube.

Docker

Docker nos proporciona los medios para realizar el contenedor para desplegar nuestra aplicación en Heroku.

Visual Studio Code

Es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. Cuenta con versión gratuita.

Clever cloud

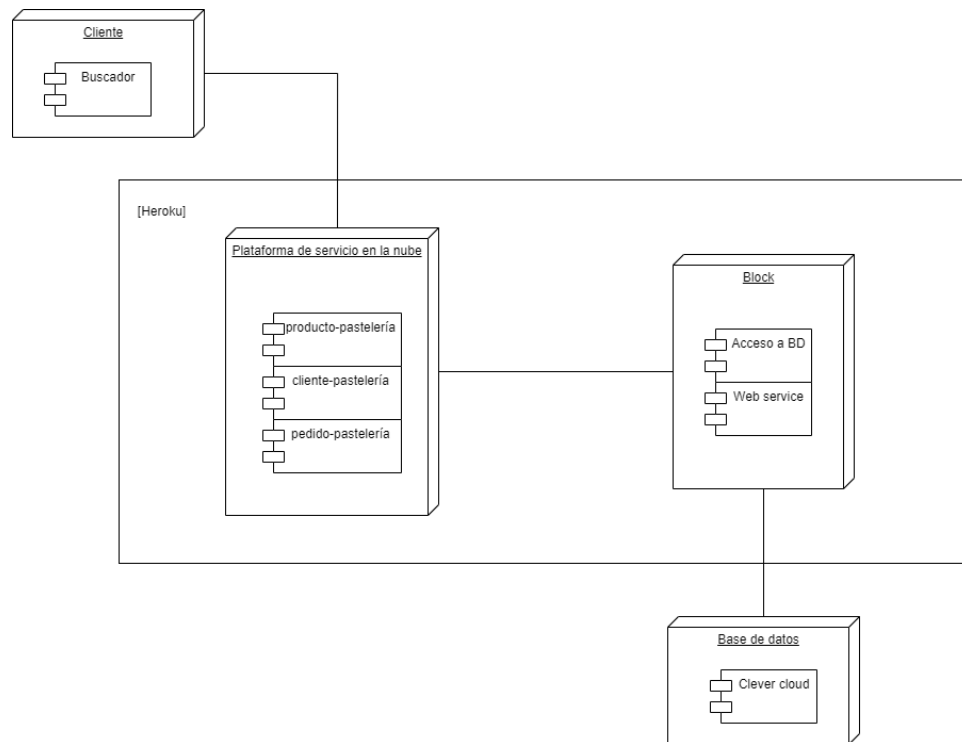
Es una plataforma como servicio que ha sido diseñada para ayudar a las organizaciones a operar, automatizar y ampliar sus negocios con diversos tiempos de ejecución y complementos. Cuenta con versión gratuita.

Spring boot

Es una herramienta web que puedes utilizar para autogenerar el esqueleto de tu aplicación Spring Boot. Se trata de un inicializador muy útil, ya que genera el pom.xml

(o build.gradle), el main de la aplicación, y un test que comprueba que la aplicación arranca correctamente. Cuenta con versión gratuita.

Diagrama de despliegue



EndPoint

Producto

Se importa las librerías y archivos necesarios para la ejecución del proyecto.

```
package mx.uv.productos;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.ws.server.endpoint.annotation.Endpoint;
import org.springframework.ws.server.endpoint.annotation.PayloadRoot;
import org.springframework.ws.server.endpoint.annotation.RequestPayload;
import org.springframework.ws.server.endpoint.annotation.ResponsePayload;

import https.uv_mx.productos.ProductoResponse;
import https.uv_mx.productos.RegistrarProductoRequest;
import https.uv_mx.productos.MostrarProductosResponse;
import https.uv_mx.productos.ModificarProductoRequest;
import https.uv_mx.productos.ModificarProductoResponse;

import https.uv_mx.productos.BorrarProductoRequest;
import https.uv_mx.productos.BorrarProductoResponse;

@Endpoint
public class ProductoEndPoint {
```

Declaramos la variable iproductos de la clase Iproductos.

```
@Autowired
Iproductos iproductos;
```

Se establece la función request para lograr registrar un producto.

```
@PayloadRoot(localPart = "RegistrarProductoRequest", namespace =
"https://uv.mx/productos")
@ResponsePayload
```

Se crea el método para registrar un producto, especificando que para su funcionamiento debe de recibir datos en la petición.

```
public ProductoResponse RegistrarProducto(@RequestPayload
RegistrarProductoRequest peticion){
```

Creamos una variable de la clase ProductoResponse, que en conjunto con una variable de la clase productos, vamos a guardar un producto especificando sus

atributos que son: tipo, descripción y precio. Ahora con ayuda de la variable iproductos se hace posible guardarlo en la base de datos.

```
ProductoResponse respuesta = new ProductoResponse();
respuesta.setRespuesta(peticion.getTipo());

Producto producto = new Producto();
producto.setTipo(peticion.getTipo());
producto.setDescripcion(peticion.getDescripcion());
producto.setPrecio(peticion.getPrecio());

iproductos.save(producto);
return respuesta;
}
```

Ahora vamos con un método distinto, este se encarga de mostrar los productos registrados en la base de datos.

```
@PayloadRoot(localPart = "MostrarProductosRequest", namespace =
"https://uv.mx/productos")
@ResponsePayload
public MostrarProductosResponse MostrarProductos(){
```

Para el método mostrar productos, debemos declarar la respuesta, para la cual necesitamos iterar de la clase productos una lista, la cual contenga todos los datos de iproductos. Al final, debemos establecer la respuesta y le agregamos los valores para que se muestre al cliente.

```
MostrarProductosResponse respuesta = new MostrarProductosResponse();
Iterable<Producto> lista = iproductos.findAll();
for (Producto o : lista) {
    MostrarProductosResponse.Productos mostrarProducto = new
MostrarProductosResponse.Productos();
    mostrarProducto.setId(o.getId());
    mostrarProducto.setTipo(o.getTipo());
    mostrarProducto.setDescripcion(o.getDescripcion());
    mostrarProducto.setPrecio(o.getPrecio());

    respuesta.getProductos().add(mostrarProducto);

}
return respuesta;
}
```

Para el microservicio de modificar los productos existentes. Debemos de recibir los datos que son necesarios modificar, dentro de la función es necesario especificar la respuesta. Para ellos obtenemos que obtener los datos que se van a modificar en la base de datos y luego ser subido a la misma.

```
@PayloadRoot(localPart = "ModificarProductoRequest", namespace =
"https://uv.mx/productos")
@ResponsePayload
public ModificarProductoResponse ModificarProducto(@RequestPayload
ModificarProductoRequest peticion){
    ModificarProductoResponse respuesta = new ModificarProductoResponse();
    Producto e = new Producto();

    e.setId(peticion.getId());
    e.setTipo(peticion.getTipo());
    e.setDescripcion(peticion.getDescripcion());
    e.setPrecio(peticion.getPrecio());
    iproductos.save(e);
    respuesta.setRespuesta(true);
    return respuesta;
}
```

Para el ultimo microservicio de productos, se recibe un id como petición, el id del producto que se desea eliminar, luego de se eliminar el producto de a base de datos con el método deleteById.

```
@PayloadRoot(localPart = "BorrarProductoRequest", namespace =
"https://uv.mx/productos")
@ResponsePayload
public BorrarProductoResponse BorrarProducto(@RequestPayload
BorrarProductoRequest peticion){
    BorrarProductoResponse respuesta = new BorrarProductoResponse();

    iproductos.deleteById(peticion.getId());
    respuesta.setRespuesta(true);
    return respuesta;
}
}
```

Promoción

De igual manera, es requerido importar las librerías y los archivos que se utilizaran para el servicio.

```
package mx.uv.t4is.ConsultaPromociones;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.ws.server.endpoint.annotation.Endpoint;
import org.springframework.ws.server.endpoint.annotation.PayloadRoot;
import org.springframework.ws.server.endpoint.annotation.RequestPayload;
import org.springframework.ws.server.endpoint.annotation.ResponsePayload;

import https.t4is_uv_mx.consultapromociones.ActualizarPromocionRequest;
import https.t4is_uv_mx.consultapromociones.ActualizarPromocionResponse;
import https.t4is_uv_mx.consultapromociones.EliminarPromocionRequest;
import https.t4is_uv_mx.consultapromociones.EliminarPromocionResponse;
import https.t4is_uv_mx.consultapromociones.MostrarPromocionResponse;
import https.t4is_uv_mx.consultapromociones.RegistrarPromocionRequest;
import https.t4is_uv_mx.consultapromociones.RegistrarPromocionResponse;

@Endpoint
public class ConsultaPromocionesEndPoint{
```

Declaramos la variable agregarbd de la clase IConsultarPromociones.

```
@Autowired
IConsultaPromociones agregarbd;
```

Dentro de la función para registrar promoción, se va a recibir datos a través de la petición, para ello se requiere una variable de la clase ConsultarPromociones para guardar los datos de la petición para después guardarla en la base de datos.

```
@PayloadRoot(localPart = "registrarPromocionRequest", namespace =
"https://t4is.uv.mx/ConsultaPromociones")
@ResponsePayload
public RegistrarPromocionResponse RegistrarPromocion(@RequestPayload
RegistrarPromocionRequest peticion) {
    RegistrarPromocionResponse respuesta = new
RegistrarPromocionResponse();
    //se agrega a la a la base de datos
    ConsultaPromociones consultar = new ConsultaPromociones();
    respuesta.setRespuesta("OK, Se registro la promoción "+
peticion.getNombre()+" actualizado en: "+peticion.getDia());
    //consultar.setNombre(peticion.getNombre());
    consultar.setNombre(peticion.getNombre());
```

```

        consultar.setDia(peticion.getDia());
        consultar.setPrecio(peticion.getPrecio());
        //
        agregarbd.save(consultar);
        //lista.add(e);
        return respuesta;
    }

```

Para consultar promociones, no vamos a recibir una petición con parámetros, pero si se requiere construir la respuesta para mostrar las promociones. Para ello tenemos que hacer una lista con las promociones de la base de datos e iterarla para que se pueda mostrar.

```

@PayloadRoot(localPart = "mostrarPromocionRequest", namespace =
"https://t4is.uv.mx/ConsultaPromociones")
@ResponsePayload
public MostrarPromocionResponse mostrarPromociones(){
    MostrarPromocionResponse respuesta = new MostrarPromocionResponse();
    Iterable<ConsultaPromociones> lista = agregarbd.findAll();
    for (ConsultaPromociones a : lista) {
        MostrarPromocionResponse.Consulta e = new
MostrarPromocionResponse.Consulta();
        e.setNombre(a.getNombre());
        e.setId(a.getId());
        e.setDia(a.getDia());
        e.setPrecio(a.getPrecio());
        respuesta.getConsulta().add(e);
    }
    return respuesta;
}

```

Para el microservicio de modificar promoción o actualizar promoción. Debemos de recibir los datos que son necesarios modificar. Para ellos obtenemos que obtener los datos que se van a modificar en la base de datos y luego ser subido a la misma.

```

@PayloadRoot(localPart = "actualizarPromocionRequest", namespace =
"https://t4is.uv.mx/ConsultaPromociones")
@ResponsePayload
public ActualizarPromocionResponse actualizarPromocion(@RequestPayload
ActualizarPromocionRequest peticion){
    ActualizarPromocionResponse respuesta = new
ActualizarPromocionResponse();
    ConsultaPromociones element = new ConsultaPromociones();
}

```

```

        element.setId(peticion.getId());
        element.setNombre(peticion.getNombre());
        element.setDia(peticion.getDia());
        element.setPrecio(peticion.getPrecio());
        agregarbd.save(element);
        respuesta.setRespuesta(true);
        return respuesta;
    }

```

Para el ultimo microservicio de productos, necesita recibir un id como petición, el id de la promoción que se desea eliminar, luego de se elimina de la base de datos con el método deleteById.

```

    @PayloadRoot(localPart = "eliminarPromocionRequest", namespace =
"https://t4is.uv.mx/ConsultaPromociones")
    @ResponsePayload
    public EliminarPromocionResponse eliminarPromocion(@RequestPayload
EliminarPromocionRequest peticion){
        EliminarPromocionResponse respuesta = new EliminarPromocionResponse();

        agregarbd.deleteById(peticion.getId());
        respuesta.setRespuesta(true);
        return respuesta;
    }
}

```

Pedido

Librerías de Spring utilizadas para el funcionamiento del EndPoint.

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import java.time.LocalDate;

```

Agregamos las variables icliente e ipedido de las clases Icliente e Ipedido, estas clases tienen la importancia de poder crear el objeto del cliente y el pedido.

```
@RestController
public class pedidoClienteControlador {
    @Autowired
    private Icliente icliente;
    @Autowired
    private Ipedido ipedido;
```

Función para crear a un cliente. Se trata de una función Post en la que se pueden agregar datos.

```
@RequestMapping(value =
"/crearCliente/{nombre}/{domicilio}/{ciudad}/{telefono}", method =
RequestMethod.POST)
public String crearCliente(@PathVariable("nombre") String nombre,
    @PathVariable("domicilio") String domicilio,
    @PathVariable("ciudad") String ciudad,
    @PathVariable("telefono") long telefono) {

    Cliente cliente = new Cliente();
    String mensaje = "";

    cliente.setNombre(nombre);
    cliente.setDomicilio(domicilio);
    cliente.setCiudad(ciudad);
    cliente.setTelefono(telefono);

    icliente.save(cliente);

    if( icliente.save(cliente)==null){
        mensaje = "Algo a salido mal, información no guardada";

    }else{
        mensaje ="Información del cliente registrada";

    }
    return mensaje;
}
```

Esta función nos permite mostrar a los clientes. Esta función es Get para obtener los datos.

```
@RequestMapping(value = "/obtenerclientes", method = RequestMethod.GET)
public Iterable<Cliente> obtenerclientes() {
```

```
        return icliente.findAll();
    }
```

En caso de ingresar datos erróneos se genero la función para eliminar el cliente. Este método es tipo Delete.

```
@RequestMapping(value = "/eliminarcliente/{id}", method =
RequestMethod.DELETE)
public boolean eliminarcliente(@PathVariable() int id) {
    icliente.deleteById(id);
    return true;
}
```

Este método nos permite agregar datos para generar un pedido.

```
@RequestMapping(value = "/crearpedido/{total}/{idProducto}/{idPromocion}",
method = RequestMethod.POST)
public String crearpedido(@PathVariable(("total")) float total,
    @PathVariable(("idProducto")) int idProducto,
    @PathVariable(("idPromocion")) int idPromocion) {
    Pedido pedido = new Pedido();
    String mensaje = "";

    LocalDate todaysDate = LocalDate.now();

    pedido.setFecha(todaysDate);
    pedido.setTotal(total);
    pedido.setIdProducto(idProducto);
    pedido.setIdPromocion(idPromocion);

    ipedido.save(pedido);

    if( ipedido.save(pedido)==null){
        mensaje = "Algo a salido mal, información no guardada";
    }else{
        mensaje ="Información del pedido registrada";
    }
    return mensaje;
}
```

Finalmente se permite ver cuáles son los pedidos.

```
@RequestMapping(value = "/obtenerPedidos", method = RequestMethod.GET)
```



```
public Iterable<Pedido> obtenerPedidos() {
    return ipedido.findAll();
}
```

Parámetros de recepción y Devueltos.

Microservicio productos.

Registrar producto

Para poder registrar un producto se requiere registrar los campos en el código XSD con ayuda de la extensión Wizdler. Los campos son: Tipo, refiriéndose al tipo de producto. Descripción del producto, y precio. El ID del producto se genera automáticamente.

```
<xs:element name="RegistrarProductoRequest">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="tipo" type="xs:string"/>
      <xs:element name="descripcion" type="xs:string"/>
      <xs:element name="precio" type="xs:float"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

POST

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <RegistrarProductoRequest xmlns="https://uv.mx/productos">
      <tipo>Pastel</tipo>
      <descripcion>Sabor chocolate.</descripcion>
      <precio>250.00</precio>
    </RegistrarProductoRequest>
  </Body>
</Envelope>
```

Mostrar productos.

El igual que en el microservicio anterior con ayuda del código XSD se muestran los datos de los productos.

```

<xs:element name="MostrarProductosRequest"/>
▼<xs:element name="MostrarProductosResponse">
  ▼<xs:complexType>
    ▼<xs:sequence>
      ▼<xs:element maxOccurs="unbounded" name="productos">
        ▼<xs:complexType>
          ▼<xs:sequence>
            <xs:element name="id" type="xs:int"/>
            <xs:element name="tipo" type="xs:string"/>
            <xs:element name="descripcion" type="xs:string"/>
            <xs:element name="precio" type="xs:float"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

POST

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:MostrarProductosResponse xmlns:ns2="https://uv.mx/productos">
      <ns2:productos>
        <ns2:id>1</ns2:id>
        <ns2:tipo>Pastel</ns2:tipo>
        <ns2:descripcion>Pastel de 3 leches sabor chocolate</ns2:descripcion>
        <ns2:precio>250.0</ns2:precio>
      </ns2:productos>
      <ns2:productos>
        <ns2:id>2</ns2:id>
        <ns2:tipo>Gelatina</ns2:tipo>
        <ns2:descripcion>Gelatina sabor limon de tamaño mediano</ns2:descripcion>
        <ns2:precio>83.0</ns2:precio>
      </ns2:productos>
      <ns2:productos>
        <ns2:id>3</ns2:id>
        <ns2:tipo>Pastel</ns2:tipo>
        <ns2:descripcion>Pastel de fresa pequeño</ns2:descripcion>
        <ns2:precio>53.0</ns2:precio>
      </ns2:productos>
    </ns2:MostrarProductosResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Modificar producto

Con ayuda del código XSD se ingresan los campos del producto y a través del ID se identifica cual producto será modificado.

```

▼<xs:element name="ModificarProductoRequest">
  ▼<xs:complexType>
    ▼<xs:sequence>
      <xs:element name="id" type="xs:int"/>
      <xs:element name="tipo" type="xs:string"/>
      <xs:element name="descripcion" type="xs:string"/>
      <xs:element name="precio" type="xs:float"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Como respuesta se obtiene el valor true en caso de haberse modificado correctamente.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:ModificarProductoResponse xmlns:ns2="https://uv.mx/productos">
      <ns2:respuesta>true</ns2:respuesta>
    </ns2:ModificarProductoResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Eliminar producto

Para borrar un producto se ingresa el ID del producto.

```
<?xml version='1.0' encoding='utf-8'>
  <xs:element name="BorrarProductoRequest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="id" type="xs:int"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="BorrarProductoResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="respuesta" type="xs:boolean"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
</wscdl:tvns>
```

Como se respuesta se obtiene un valor true en caso de haberse borrado correctamente.

POST

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:BorrarProductoResponse xmlns:ns2="https://uv.mx/productos">
      <ns2:respuesta>true</ns2:respuesta>
    </ns2:BorrarProductoResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Microservicio promociones

Registrar promociones

Para registrar una promoción se hace a través del código XSD. Los campos de la promoción son nombre de la promoción, día de la semana en la que estará disponible la promoción y precio que se le dará al producto.

```
<xs:element name="registrarPromocionRequest">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="dia" type="xs:string"/>
      <xs:element name="precio" type="xs:float"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="registrarPromocionResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="respuesta" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

POST <https://promociones-pasteleria.herokuapp.com:443/ws>

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <registrarPromocionRequest xmlns="https://t4is.uv.mx/ConsultaPromociones">
      <nombre>Descuento el los flanes</nombre>
      <dia>Jueves</dia>
      <precio>75</precio>
    </registrarPromocionRequest>
  </Body>
</Envelope>
```

Mostrar promociones

Para mostrar productos se hace a través del código XSD.

```

<xs:element name="mostrarPromocionRequest"/>
▼<xs:element name="mostrarPromocionResponse">
  ▼<xs:complexType>
    ▼<xs:sequence>
      ▼<xs:element maxOccurs="unbounded" name="consulta">
        ▼<xs:complexType>
          ▼<xs:sequence>
            <xs:element name="id" type="xs:int"/>
            <xs:element name="nombre" type="xs:string"/>
            <xs:element name="dia" type="xs:string"/>
            <xs:element name="precio" type="xs:float"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

POST

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:mostrarPromocionResponse xmlns:ns2="https://t4is.uv.mx/ConsultaPromociones">
      <ns2:consulta>
        <ns2:id>1</ns2:id>
        <ns2:nombre>Pastel helado más un flan</ns2:nombre>
        <ns2:dia>viernes</ns2:dia>
        <ns2:precio>400.0</ns2:precio>
      </ns2:consulta>
    </ns2:mostrarPromocionResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Eliminar promoción

Para eliminar la promoción se ingresa el ID y a través del código XSD se elimina el producto. También se debe ingresar el nombre de la promoción.

```

▼<xs:element name="eliminarPromocionRequest">
  ▼<xs:complexType>
    ▼<xs:sequence>
      <xs:element name="id" type="xs:int"/>
      <xs:element name="nombre" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
▼<xs:element name="eliminarPromocionResponse">
  ▼<xs:complexType>
    ▼<xs:sequence>
      <xs:element name="respuesta" type="xs:boolean"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>

```

Como respuesta se obtiene el valor true en caso de eliminarse correctamente.

```

POST https://promociones-pasteleria.herokuapp.com:443/ws
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:eliminarPromocionResponse xmlns:ns2="https://t4is.uv.mx/ConsultaPromociones">
      <ns2:respuesta>true</ns2:respuesta>
    </ns2:eliminarPromocionResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Actualizar promoción

A través del código se puede actualizar la promoción en caso de que deba ser cambiada pero no sea necesario eliminarla. El microservicio a través del ID identifica el producto a modificar.

```

▼ <xs:element name="actualizarPromocionRequest">
  ▼ <xs:complexType>
    ▼ <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="id" type="xs:int"/>
      <xs:element name="dia" type="xs:string"/>
      <xs:element name="precio" type="xs:float"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
▼ <xs:element name="actualizarPromocionResponse">
  ▼ <xs:complexType>
    ▼ <xs:sequence>
      <xs:element name="respuesta" type="xs:boolean"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

POST https://promociones-pasteleria.herokuapp.com:443/ws
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:actualizarPromocionResponse xmlns:ns2="https://t4is.uv.mx/ConsultaPromociones">
      <ns2:respuesta>true</ns2:respuesta>
    </ns2:actualizarPromocionResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

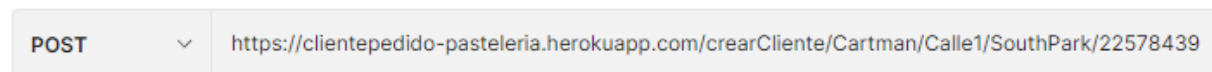
```

Microservicio cliente-pedido.

Con ayuda de postman podemos hacer uso de los servicios que nos brinda esta API. Los datos se obtienen en formato Json y se ingresan a través del URL.

Crear cliente

En este caso seleccionamos el tipo POST, pues enviaremos datos para ser guardados. Se deben ingresar el nombre, calle, ciudad y número de teléfono por medio de la URL. Si el cliente se registra correctamente se regresa un mensaje de que la información ha sido guardada.



Información del cliente registrada.

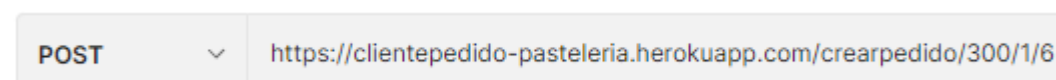
Obtener clientes.

En este caso seleccionamos el tipo GET, a que se obtienen los datos de los clientes realizados.

```
{
  "id": 1,
  "nombre": "freddy",
  "domicilio": "siempreviva",
  "ciudad": "sprinffield",
  "telefono": 2284103051
},
{
  "id": 6,
  "nombre": "Cartman",
  "domicilio": "Calle1",
  "ciudad": "SouthPark",
  "telefono": 22578439
}
```

Registrar un pedido

En este caso seleccionamos el tipo POST, pues enviaremos datos para ser guardados. Se deben ingresar el precio, id del producto y el id del cliente por medio de la URL, el API se encarga de generar la fecha en que se realiza. Si el pedido se registra correctamente se regresa un mensaje de que la información ha sido guardada.



Información del pedido registrada

Obtener clientes.

En este caso seleccionamos el tipo GET, a que se obtienen los datos de los pedidos realizados.

GET	▼	https://clientepedido-pasteleria.herokuapp.com/obtenerPedidos
-----	---	---

```
{
  {
    "id": 3,
    "fecha": "2022-06-03",
    "total": 2000.66,
    "idProducto": 4,
    "idPromocion": 5
  },
  {
    "id": 4,
    "fecha": "2022-06-03",
    "total": 700.0,
    "idProducto": 8,
    "idPromocion": 2
  }
}
```

Eliminar un cliente

Existe la opción de eliminar un cliente, en caso de que su información ya no sea necesaria, los pedidos se guardan por temas de inventario. En este caso usamos el método DELETE y con el id del cliente se identifica cual será borrado. Al final regresa la opción true si fue eliminado correctamente.

DELETE	▼	https://clientepedido-pasteleria.herokuapp.com/eliminarcliente/6
--------	---	---

Plan de pruebas

El plan de pruebas presentado se realizó con el propósito de identificar posibles fallas y errores de los servicios, también uno de los propósitos es verificar el correcto

funcionamiento de cada uno de los microservicios y las funcionalidades que lo componen a cada uno de ellos. Las pruebas se realizaron a todos los servicios.

Servicio productos.

Recursos

Requerimientos de entornos – Hardware

Hardware: Una computadora con acceso a un navegador

Red: Conexión a internet wifi



Requerimientos de entornos – Software



Software: extensión de wizzler en nuestro navegador.

Fecha	03/06/2022
Título	Registrar producto
Descripción	El usuario registra un producto llenando los campos correspondientes
Condiciones de Entrada	<ul style="list-style-type: none">El usuario debe entrar al enlace de Heroku: https://microservicio-pasteleria.herokuapp.com/ws/productos.wsdlDebe usar la extensión Wizzler y seleccionar la funcionalidad “RegistrarProducto”Llenar los campos correspondientes
Resultado Esperado	El servicio debe guardar los datos introducidos en la base de datos Clever Cloud y mostrar un mensaje de que el producto fue registrado correctamente.
Resultado Obtenido	El servicio guarda los datos introducidos en la base de datos Clever Cloud y mostrar un mensaje de que el producto fue registrado correctamente.

Fecha	03/06/2022
Título	Mostrar productos
Descripción	El usuario observa los productos disponibles
Condiciones de Entrada	<ul style="list-style-type: none">El usuario debe entrar al enlace de Heroku: https://microservicio-pasteleria.herokuapp.com/ws/productos.wsdlDebe usar la extensión Wizzler y seleccionar la funcionalidad “MostrarProductos”
Resultado Esperado	El servicio debe recuperar los datos en la base de datos Clever Cloud y mostrar los productos.

Resultado Obtenido	El servicio recupera los datos en la base de datos Clever Cloud y mostrar los productos.
---------------------------	--

Fecha	03/06/2022
Título	Modificar producto
Descripción	El usuario modifica un producto.
Condiciones de Entrada	 El usuario debe entrar al enlace de Heroku: https://microservicio-pasteleria.herokuapp.com/ws/productos.wsdl  Debe usar la extensión Wizdler y seleccionar la funcionalidad "ModificarProductos"
Resultado Esperado	El servicio debe actualizar los datos en la base de datos Clever Cloud y mostrar un mensaje de producto modificado.
Resultado Obtenido	El servicio actualiza los datos en la base de datos Clever Cloud y mostrar un mensaje de producto modificado.

Fecha	03/06/2022
Título	Eliminar producto
Descripción	El usuario Elimina un producto.
Condiciones de Entrada	 El usuario debe entrar al enlace de Heroku: https://microservicio-pasteleria.herokuapp.com/ws/productos.wsdl  Debe usar la extensión Wizdler y seleccionar la funcionalidad "EliminarProductos"
Resultado Esperado	El servicio debe eliminar los datos en la base de datos Clever Cloud y mostrar un mensaje si el producto fue eliminado.
Resultado Obtenido	El servicio elimina los datos en la base de datos Clever Cloud y mostrar un mensaje si el producto fue eliminado.

Servicio promociones.

Recursos



Requerimientos de entornos – Hardware



Hardware: Una computadora con acceso a un navegador



Red: Conexión a internet wifi



Requerimientos de entornos – Software

Software: extensión de wizdler en nuestro navegador.

Fecha	03/06/2022
Título	Registrar promoción
Descripción	El usuario registra una promoción
Condiciones de Entrada	 El usuario debe entrar al enlace de Heroku: https://promociones-pasteleria.herokuapp.com/ws/promociones.wsdl  Debe usar la extensión Wizdler y seleccionar la funcionalidad "RegistrarPromocion"
Resultado Esperado	El servicio debe registrar los datos en la base de datos Clever Cloud y mostrar un mensaje si la promoción fue registrada.
Resultado Obtenido	El servicio registra los datos en la base de datos Clever Cloud y mostrar un mensaje si la promoción fue registrada.

Fecha	03/06/2022
Título	Mostrar promoción.
Descripción	El usuario registra una promoción
Condiciones de Entrada	 El usuario debe entrar al enlace de Heroku: https://promociones-pasteleria.herokuapp.com/ws/promociones.wsdl  Debe usar la extensión Wizdler y seleccionar la funcionalidad "MostrarPromocion"
Resultado Esperado	El servicio debe buscar los datos en la base de datos Clever Cloud y mostrar los datos de las promociones actuales.
Resultado Obtenido	El servicio busca los datos en la base de datos Clever Cloud y muestra los datos de las promociones actuales.

Fecha	03/06/2022
Título	Actualizar promoción
Descripción	El usuario actualiza una promoción
Condiciones de Entrada	 El usuario debe entrar al enlace de Heroku: https://promociones-pasteleria.herokuapp.com/ws/promociones.wsdl  Debe usar la extensión Wizdler y seleccionar la funcionalidad "ActualizarPromocion"
Resultado Esperado	El servicio debe modificar los datos en la base de datos Clever Cloud y mostrar un mensaje si la promoción fue modificada.
Resultado Obtenido	El servicio modifica los datos en la base de datos Clever Cloud y muestra un mensaje si la promoción fue modificada.

Fecha	03/06/2022
Título	Eliminar promoción
Descripción	El usuario elimina una promoción
Condiciones de Entrada	 El usuario debe entrar al enlace de Heroku: https://promociones-pasteleria.herokuapp.com/ws/promociones.wsdl  Debe usar la extensión Widdler y seleccionar la funcionalidad "EliminarPromocion"
Resultado Esperado	El servicio debe borrar los datos en la base de datos Clever Cloud y mostrar un mensaje si la promoción fue eliminada.
Resultado Obtenido	El servicio borra los datos en la base de datos Clever Cloud y muestra un mensaje si la promoción fue eliminada.

Servicio clientepedido.

Recursos



Requerimientos de entornos – Hardware


Hardware: Una computadora con acceso a un navegador


Red: Conexión a internet wifi



Requerimientos de entornos – Software



Software: puedes usar el servicio Postman.



Fecha	03/06/2022
Título	Crear cliente
Descripción	El usuario crea un cliente enviando los datos a través de la URL
Condiciones de Entrada	 El usuario debe entrar al enlace de Heroku: https://clientepedido-pasteleria.herokuapp.com  Debe enviar los datos del cliente agregando a la ruta /crearCliente/nombre del cliente/ domicilio /ciudad / teléfono
Resultado Esperado	El servicio debe registrar los datos en la base de datos Clever Cloud y mostrar un mensaje si se agregó al cliente.
Resultado Obtenido	El servicio registrarlos datos en la base de datos Clever Cloud y muestra un mensaje si se agregó al cliente.

Fecha	03/06/2022
Título	Crear pedido
Descripción	El usuario crea un pedido enviando los datos a través de la URL
Condiciones de Entrada	 El usuario debe entrar al enlace de Heroku: https://clientepedido-pasteleria.herokuapp.com

	 Debe enviar los datos del pedido agregando a la ruta /crearpedido/total del pedido /id producto/ id cliente
Resultado Esperado	El servicio debe registrar los datos en la base de datos Clever Cloud y mostrar un mensaje si se agregó el pedido.
Resultado Obtenido	El servicio registra los datos en la base de datos Clever Cloud y muestra un mensaje si se agregó el pedido.

Fecha	03/06/2022
Título	Obtener clientes
Descripción	El usuario obtiene los datos de los clientes.
Condiciones de Entrada	 El usuario debe entrar al enlace de Heroku: https://clientepedido-pasteleria.herokuapp.com  Debe ingresar datos e la ruta la extensión /obtenerclientes
Resultado Esperado	El servicio debe buscar los datos en la base de datos Clever Cloud y mostrarlos.
Resultado Obtenido	El servicio busca los datos en la base de datos Clever Cloud y los muestra.

Fecha	03/06/2022
Título	Obtener pedidos
Descripción	El usuario obtiene los datos de los pedidos.
Condiciones de Entrada	 El usuario debe entrar al enlace de Heroku: https://clientepedido-pasteleria.herokuapp.com  Debe ingresar datos e la ruta la extensión /obtenerPedidos
Resultado Esperado	El servicio debe buscar los datos en la base de datos Clever Cloud y mostrarlos.
Resultado Obtenido	El servicio busca los datos en la base de datos Clever Cloud y los muestra.

Fecha	03/06/2022
Título	Eliminar cliente
Descripción	El usuario elimina los datos de un cliente.
Condiciones de Entrada	 El usuario debe entrar al enlace de Heroku: https://clientepedido-pasteleria.herokuapp.com  Debe ingresar datos e la ruta la extensión /eliminarcliente / id del cliente
Resultado Esperado	El servicio debe buscar los datos en la base de datos Clever Cloud y eliminar al cliente con el id.
Resultado Obtenido	El servicio busca los datos en la base de datos Clever Cloud y elimina al cliente con el id.

Dockerfile microservicio cliente-pedido

From rrojano/jdk8

add app/pedidocliente-0.0.1-SNAPSHOT.jar /app/pedidocliente-0.0.1-SNAPSHOT.jar

WORKDIR /app

cmd java -jar -Dserver.port=\$PORT pedidocliente-0.0.1-SNAPSHOT.jar

Dockerfile microservicio promociones

From rrojano/jdk8

add app/ConsultaPromociones-0.0.1-SNAPSHOT.jar /app/ConsultaPromociones-0.0.1-SNAPSHOT.jar

WORKDIR /app

cmd java -jar -Dserver.port=\$PORT ConsultaPromociones-0.0.1-SNAPSHOT.jar

Dockerfile microservicio productos

From rrojano/jdk8

add app/Productos-0.0.1-SNAPSHOT.jar /app/Productos-0.0.1-SNAPSHOT.jar

WORKDIR /app

cmd java -jar -Dserver.port=\$PORT Productos-0.0.1-SNAPSHOT.jar

Github

https://github.com/FreddySahid/Tecnologias_integracion_soluciones

Heroku

<https://promociones-pasteleria.herokuapp.com/ws/promociones.wsdl>

<https://clientepedido-pasteleria.herokuapp.com/obtenerPedidos>

<https://microservicio-pasteleria.herokuapp.com/ws/productos.wsdl>

