



UNIVERSIDAD VERACRUZANA

Facultad de Estadística e Informática

Proyecto Maíz

Tecnologías para la Integración de Soluciones
José Rafael Rojano Cáceres

García Ceballos Jair
López Aguilar Ingrid Yadelí
César Alejandro Vallejo Galván
05 de mayo de 2022

Contenido

Introducción	2
Motivación	2
Problemática	2
Solución	4
Costos	4
Diagrama de despliegue	5
Documentación del API SOAP	5
EndPoint	5
Parámetros de recepción	8
Parámetros devueltos	2
Plan de pruebas	11
Forma de ejecución de los contenedores	11
Archivos de Dockerfile y docker-compose.yml	11
Documentación del API REST	12
EndPoint	2
Parámetros de recepción	2
Parámetros devueltos	2
Plan de pruebas	19
Forma de ejecución de los contenedores	19
Archivos de Dockerfile y docker-compose.yml	20

Introducción

Durante la experiencia educativa Tecnologías para la Integración de Soluciones aprendimos a crear servicios web de tipo SOAP y REST. Para evidenciar dichos aprendizajes, para nuestro proyecto final decidimos implementar un servicio web para una tortillería ficticia. Este servicio no tiene implementada una interfaz gráfica, pues el objetivo es priorizar el uso de los servicios.

En este documento se plasma la documentación de la implementación para este servicio, dando a conocer a mayor detalle la problemática y la solución para esta. A su vez, se mostrará con mayor detalle la funcionalidad de este.

Motivación

Nuestra motivación como equipo para la realización de este servicio es practicar y mejorar los aprendizajes adquiridos durante el semestre en la experiencia educativa Tecnologías para la Integración de Soluciones.

Así como también, nos motiva que, debido al avance tecnológico con el paso de los días, este pueda ser un servicio que sea requerido por los negocios de tortillerías para una mayor eficiencia, tanto para los clientes como para los dueños y/o trabajadores.

Problemática

Nuestra problemática está enfocada en una tortillería ficticia, en la que debido al aumento de ventas requería de un mejor sistema para agilizar la administración de los productos, así como también permitir que los clientes tengan mayor comodidad para realizar la compra.

La tortillería provee a terceros de sus servicios a través de dos formas. En local o a través de reparto. En cualquiera de las dos formas, sustenta la necesidad de sus clientes. La cual es proveer de tortilla, ya sea para consumo propio, como para reventa de esta.

Venta en local

La venta local, consiste en que los clientes vayan directamente al establecimiento a recoger su producto. Esto desemboca en una transacción entre el vendedor en punto de venta y el cliente.

En punto de venta, se pueden proveer los siguientes productos:

- Tortilla
- Salsas
- Mole
- Totopos
- Tostadas
- Tortilla tostada en tiras

- Masa de Maíz

Cada producto cuenta con los siguientes atributos:

- Nombre
- Gramaje
- Precio

A cada producto escrito anteriormente, se le lleva un conteo. Cada uno está dado de alta en inventario. Cada vez que un producto es vendido, este se resta del inventario.

A parte de proveer a los clientes, el vendedor en local tiene la obligación de informarle al vendedor de reparto, la cantidad total de kilos de tortilla que lleva en su salida en ruta.

Venta en reparto

El repartidor, cada mañana llega al local. Lo primero que hace es registrar su entrada y prepararse para su salida. Una de las actividades que realiza antes de salir a repartir, es mirar la lista de clientes a los cuales proveerá. Una vez trazada su ruta, lo que debe hacer es abastecerse de tortilla. La cantidad que lleva de tortilla en su vehículo, varía en cada salida a ruta. El encargado de decirle al repartidor cuantos kilos de tortilla lleva a vender, es el vendedor en local. Quien guarda un registro de lo que se lleva el repartidor en cada salida a ruta. El repartidor confirma la cantidad que recibe de kilos de tortilla, y sale a vender.

Una vez llega al primer destino, revisa la cantidad de kilos de tortilla que pudieran haber sobrado. Los lleva a su vehículo y prepara la hielera para reabastecer al cliente. Por cada kilo de tortilla que haya sobrado del día anterior, se le descontará al cliente del total que compre hoy. También se lleva un conteo de la cantidad de kilos de tortilla que se han regresado cada cliente.

Una vez reabastecido el cliente, se hace un total de costos. A ese total se le descuenta la tortilla sobrante, cada descuento es equivalente al precio total de la tortilla para ese cliente. Por ejemplo, si al cliente le sobraron 2 kilos, y cada kilo le cuesta \$12 pesos mexicanos, al cliente se le descuenta \$24 pesos mexicanos del total que compre el día de hoy. Otra cosa a tener en cuenta es que existen 2 tipos de clientes. Existen los clientes individuales, y aquellos que son empresas. Los clientes individuales tienen un crédito fijo, el cual se va agotando cada vez que el cliente decide usarlo. Si se les agota el crédito, no se les podrá abastecer más hasta que paguen. Los clientes que son empresas solo pagan a plazo. Se decide un plazo, y en base a ese se hacen los pagos. Por ejemplo, se fija el plazo de una semana, se decide que hoy lunes se les abastecerá de forma indeterminada. Pero el siguiente lunes pagarán todo lo que hayan consumido toda la semana. En caso de no hacer el debido pago, no se les podrá abastecer hasta que se haga el pago correspondiente.

Tomando todo esto en cuenta, se hace un total a pagar. En caso de los clientes individuales, se revisa si se tiene adeudo en su crédito, en caso de tenerlo, se suma al total a pagar. Si

hay tortilla sobrante, se resta del total a pagar. Y se suma toda la tortilla que se dejará en ese momento. En caso de ser una empresa, se descuentan los kilos de tortilla fríos al total que se dejará en ese momento. En ambos casos se entrega un ticket que demuestre que la transacción se llevó a cabo.

La cantidad de repartidores crecerá conforme crezca el negocio, por esa razón, se requiere de un sistema de usuarios.

Solución

La solución planeada por el equipo es la implementación de servicios para la administración de ventas y productos de la tortillería, estos divididos en micro-servicios, un servicio es para el punto de venta y otro servicio para delivery o repartidor.

Las funciones de cada servicio se describen a continuación:

Servicio de punto de venta:

- Agregar productos
- Buscar productos
- Modificar inventario
- Borrar producto
- Modificar stock
- Hacer ventas

Servicio de delivery o repartidor:

- Obtener lista de clientes.
- Obtener lista de compañías.
- Hacer ventas.
- Agendar ventas.
- Mirar pedidos agendados.
- Obtener pedido.
- Obtener cliente.
- Login.

Costos

Visual Studio Code

Para el proyecto optamos por utilizar Visual Studio Code como editor de código, esto porque creemos que tiene una mejor organización, además creemos que es más fácil poder subir el repositorio a GitHub. Es gratuito, por lo cual nos generó \$0.00 pesos en costo.

Github

Utilizamos la plataforma de GitHub como plataforma de desarrollo colaborativo, para alojar nuestro proyecto. Es gratuito, por lo cual nos generó \$0.00 pesos en costo.

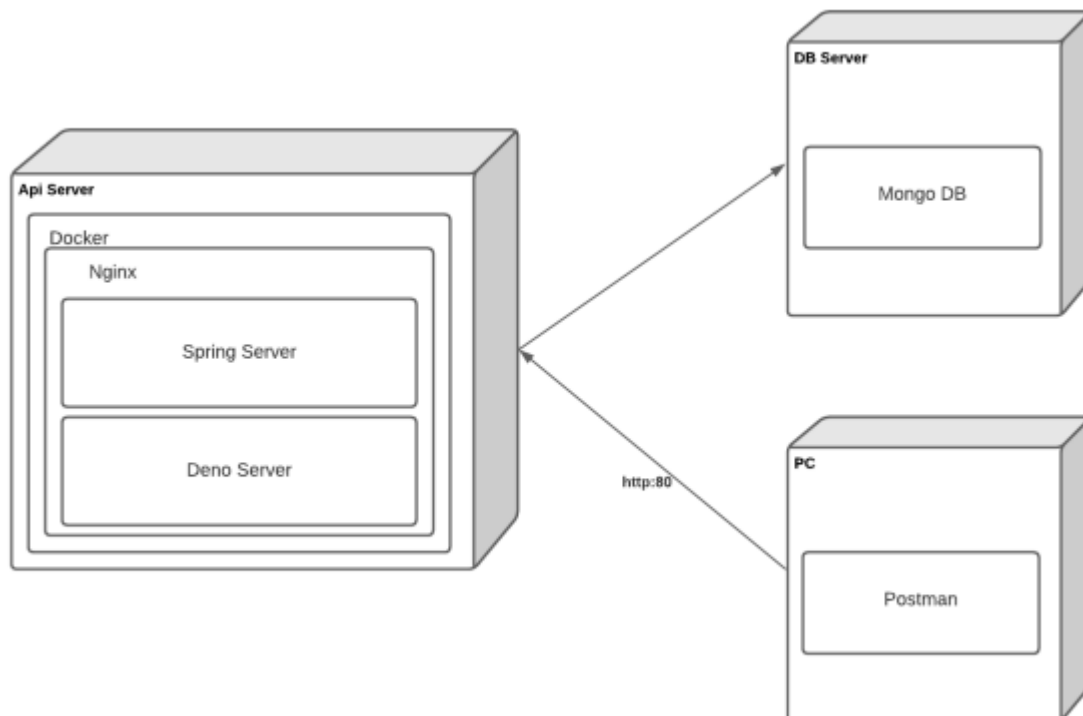
Heroku

Usamos la plataforma de Heroku para desplegar el sistema de nuestro proyecto en la nube, debido a que ofrece una mejor flexibilidad y agilidad a la hora de construir aplicaciones en Web. Es gratuito, por lo cual nos generó \$0.00 pesos en costo.

Clever Cloud

Elegimos Clever Cloud en su versión gratuita para alojar la base de datos de nuestro sistema.

Diagrama de despliegue



Documentación del API SOAP

EndPoint

En este servicio se implementan 6 métodos diferentes para lograr una buena administración de la tortillería. En estos encontramos un método para la recepción de datos y otro para el envío de estos.

```

// Agregar producto
@PayloadRoot(namespace = "https://t4is.uv.mx/pos", localPart = "AgregarProductoRequest")
@ResponsePayload
public AgregarProductoResponse agregarProducto(@RequestPayload AgregarProductoRequest peticion) {
    AgregarProductoResponse respuesta = new AgregarProductoResponse();

    try {
        // Verificar si están vacíos los parametros
        if (peticion.getNombre().isEmpty() || (peticion.getCantidad() == 0 && peticion.getGramaje() == 0 && peticion.getPrecio() == 0)) {
            respuesta.setMsg("Error: Parametro/s vacío/s");
            return respuesta;
        }

        // Obtener el producto (Se usa optional porque no sabemos si va a estar)
        Optional<Inventario> producto = iinventory.findByNombre(peticion.getNombre());
        // En caso de estar el producto en la base de datos
        if (producto.isPresent()) {
            respuesta.setMsg("El producto ya se encuentra en la base de datos");
            return respuesta;
        }

        // Crear nuevo producto
        Inventario nuevoProducto = new Inventario();
        nuevoProducto.setNombre(peticion.getNombre());
        nuevoProducto.setCantidad(peticion.getCantidad());
        nuevoProducto.setPrecio(peticion.getPrecio());
        nuevoProducto.setGramaje(peticion.getGramaje());

        // Crear producto
        iinventory.save(nuevoProducto);
    } catch (Exception e) {
        System.out.println(e);
        respuesta.setMsg("Algo salió muy mal :(");
        return respuesta;
    }

    respuesta.setMsg("Se insertó correctamente el producto");
    return respuesta;
}

```

```

// Mirar todos los productos de la base de datos
@PayloadRoot(namespace = "https://t4is.uv.mx/pos", localPart = "BuscarProductosRequest")
@ResponsePayload
public BuscarProductosResponse buscar() {
    BuscarProductosResponse respuesta = new BuscarProductosResponse();

    Iterable<Inventario> lista = iinventory.findAll();
    for (Inventario producto : lista) {
        BuscarProductosResponse.Inventario i = new BuscarProductosResponse.Inventario();
        i.setId(producto.getId());
        i.setNombre(producto.getNombre());
        i.setCantidad(producto.getCantidad());
        i.setGramaje(producto.getGramaje());
        i.setPrecio(producto.getPrecio());

        respuesta.getInventario().add(i);
    }
    return respuesta;
}

```

```

// Modificar inventario
@PayloadRoot(namespace = "https://t4is.uv.mx/pos", localPart = "ModificarInventarioRequest")
@ResponsePayload
public ModificarInventarioResponse modificarInventario(@RequestPayload ModificarInventarioRequest peticion) {
    ModificarInventarioResponse respuesta = new ModificarInventarioResponse();

    try {
        // Verificar si están vacíos los parametros
        if (peticion.getNombre().isEmpty()) {
            respuesta.setMsg("Error: No dejes el nombre vacío");
            return respuesta;
        }

        // Obtener el producto (Se usa optional porque no sabemos si va a estar)
        Optional<Inventario> producto = iinventory.findByNombre(peticion.getNombre());
        // En caso de no estar el producto en la base de datos
        if (!producto.isPresent()) {
            respuesta.setMsg("Producto no encontrado");
            return respuesta;
        }

        // Crear producto
        Inventario nuevoProducto = new Inventario();
        nuevoProducto.setId(producto.get().getId());
        nuevoProducto.setNombre(peticion.getNombre());
        nuevoProducto.setCantidad(peticion.getCantidad());
        nuevoProducto.setGramaje(peticion.getGramaje());
        nuevoProducto.setPrecio(peticion.getPrecio());

        // Actualizar stock
        iinventory.save(nuevoProducto);
    } catch (Exception e) {
        System.out.println(e);
        respuesta.setMsg("Algo salió muy mal :(");
        return respuesta;
    }

    respuesta.setMsg("Se actualizó correctamente el producto");
    return respuesta;
}

```

```

// Borrar por nombre
@PayloadRoot(namespace = "https://t4is.uv.mx/pos", localPart = "BorrarProductoRequest")
@ResponsePayload
public BorrarProductoResponse borrarInventario(@RequestPayload BorrarProductoRequest peticion) {
    BorrarProductoResponse respuesta = new BorrarProductoResponse();

    try {
        // Verificar si están vacíos los parametros
        if (peticion.getNombre().isEmpty()) {
            respuesta.setMsg("Error: No dejes el nombre vacío");
            return respuesta;
        }

        // Obtener el producto (Se usa optional porque no sabemos si va a estar)
        Optional<Inventario> producto = iinventory.findByNombre(peticion.getNombre());
        // En caso de no estar el producto en la base de datos
        if (!producto.isPresent()) {
            respuesta.setMsg("Producto no encontrado");
            return respuesta;
        }

        // borrar
        iinventory.deleteById(producto.get().getId());
    } catch (Exception e) {
        System.out.println(e);
        respuesta.setMsg("Algo salió muy mal :(");
        return respuesta;
    }

    respuesta.setMsg("Se borró correctamente el producto");
    return respuesta;
}

```


Parámetros de recepción

Los métodos en el EndPoint reciben datos correspondientes a las acciones que se desean.



AgregarProducto

En este método, como su nombre lo dice, funciona para agregar productos nuevos al inventario de la tortillería. Si se intenta agregar un producto ya antes registrado, nos muestra un mensaje de que el producto ya está registrado en la base de datos. Los datos que recibirá son:

- ✓ **Nombre** del producto a agregar.
- ✓ **Cantidad** del producto.
- ✓ **Gramaje** del producto.
- ✓ **Precio** del producto.

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <AgregarProductoRequest xmlns="https://t4is.uv.mx/pos">
      <nombre>totopos</nombre>
      <cantidad>90</cantidad>
      <gramaje>7900</gramaje>
      <precio>50</precio>
    </AgregarProductoRequest>
  </Body>
</Envelope>
```

```
POST http://localhost:8080/ws
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:AgregarProductoResponse xmlns:ns2="https://t4is.uv.mx/pos">
      <ns2:msg>Se insertó correctamente el producto</ns2:msg>
    </ns2:AgregarProductoResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

BorrarProducto

Aquí se elimina el producto deseado que se encuentre registrado previamente, la manera de realizar esto es por medio del nombre del producto. En caso de no estar dentro de la base de datos, se muestra mensaje de error. Al igual que si el nombre del producto es ingresado incorrectamente también resultará en error.

Datos que obtiene:

- ✓ **Nombre** del producto previamente agregado y almacenado en el inventario.

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <BorrarProductoRequest xmlns="https://t4is.uv.mx/pos">
      <nombre>[string]</nombre>
    </BorrarProductoRequest>
  </Body>
</Envelope>
```

POST http://localhost:8080/ws

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:BorrarProductoResponse xmlns:ns2="https://t4is.uv.mx/pos">
      <ns2:msg>Se borró correctamente el producto</ns2:msg>
    </ns2:BorrarProductoResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

BuscarProductos

Este método mostrará la lista de los productos almacenados en el inventario sin recibir nada.

Los datos que despliega son:

- ✓ **Id** del producto.
- ✓ **Nombre** del producto.
- ✓ **Cantidad** producto en existencia.
- ✓ **Gramaje** del producto.
- ✓ **Precio** del producto.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:BuscarProductosResponse xmlns:ns2="https://t4is.uv.mx/pos">
      <ns2:inventario>
        <ns2:id>1</ns2:id>
        <ns2:nombre>papas</ns2:nombre>
        <ns2:cantidad>15</ns2:cantidad>
        <ns2:gramaje>2000</ns2:gramaje>
        <ns2:precio>50.0</ns2:precio>
      </ns2:inventario>
      <ns2:inventario>
        <ns2:id>4</ns2:id>
        <ns2:nombre>tortillas</ns2:nombre>
        <ns2:cantidad>0</ns2:cantidad>
        <ns2:gramaje>153</ns2:gramaje>
        <ns2:precio>1.2</ns2:precio>
      </ns2:inventario>
      <ns2:inventario>
        <ns2:id>6</ns2:id>
        <ns2:nombre>totopos</ns2:nombre>
        <ns2:cantidad>35</ns2:cantidad>
        <ns2:gramaje>5000</ns2:gramaje>
        <ns2:precio>12.0</ns2:precio>
      </ns2:inventario>
    </ns2:BuscarProductosResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

ModificarInventario

Aquí podemos modificar los elementos de los productos, solo debemos especificar el nombre del producto que se quiere modificar y posterior a esto cambiar los valores a los demás atributos. Una vez realizado esto, visualizamos un mensaje de modificación exitosa.

Datos que recibe:

- ✓ Nombre
- ✓ Cantidad
- ✓ Gramaje
- ✓ Precio

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <ModificarInventarioRequest xmlns="https://t4is.uv.mx/pos">
      <nombre>[string]</nombre>
      <cantidad>[int]</cantidad>
      <gramaje>[int]</gramaje>
      <precio>[float]</precio>
    </ModificarInventarioRequest>
  </Body>
</Envelope>
```

```
POST http://localhost:8080/ws
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:ModificarInventarioResponse xmlns:ns2="https://t4is.uv.mx/pos">
      <ns2:msg>Se actualizó correctamente el producto</ns2:msg>
    </ns2:ModificarInventarioResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Venta

El servicio de venta permitirá realizar ventas de los productos que estén registrados (tortillas, papas, totopos, entre otros) y estos están definidos con un id el cual necesitaremos para realizar la venta: ingresamos el id, posteriormente la cantidad y por último el nombre del vendedor.

Datos que recibe:

- ✓ Artículo
- ✓ Cantidad
- ✓ Vendedor

Algunas de las excepciones que puede marcar son:

- * Si intentas vender más cantidad de la que hay en un producto, no te dejará realizar la venta por productos insuficientes.
- * Si un producto está agotado tampoco dejará realizar la venta.
- * Si se ingresa un id de artículo que no esté registrado, no dejará realizar la venta.

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <VentaRequest xmlns="https://t4is.uv.mx/pos">
      <articulo>[int]</articulo>
      <cantidad>[int]</cantidad>
      <vendedor>[string]</vendedor>
    </VentaRequest>
  </Body>
</Envelope>
```

Modificarstock

Para el método de modificar stock necesitaremos el nombre del usuario y debemos ingresar la cantidad por la que modificaremos el stock, abriremos la base de datos porque ya no me dió tiempo de implementar esta parte jaja y observamos que se ha modificado exitosamente y eso sería todo de mi parte.

Datos que recibe:

- ✓ Nickname
- ✓ Tortillas
- ✓ Totopos

Excepciones:

- * Si no está registrado el usuario no dejará modificar el stock.

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <ModificarStockRequest xmlns="https://t4is.uv.mx/pos">
      <nickname>[string]</nickname>
      <tortillas>[int]</tortillas>
      <totopos>[int]</totopos>
    </ModificarStockRequest>
  </Body>
</Envelope>
```

Plan de pruebas

Forma de ejecución de los contenedores

Archivos de Dockerfile y docker-compose.yml

Documentación del API REST

API REST

URL Base
https://tproject4is.herokuapp.com/user

Login

Para comenzar a utilizar el api del repartidor, lo primero que necesitamos es loggearnos con el usuario y contraseña para que podamos recibir nuestro token.

Method	Path
POST	/login

El email y contraseña mostrados aquí están dados de alta en la base de datos y son válidos

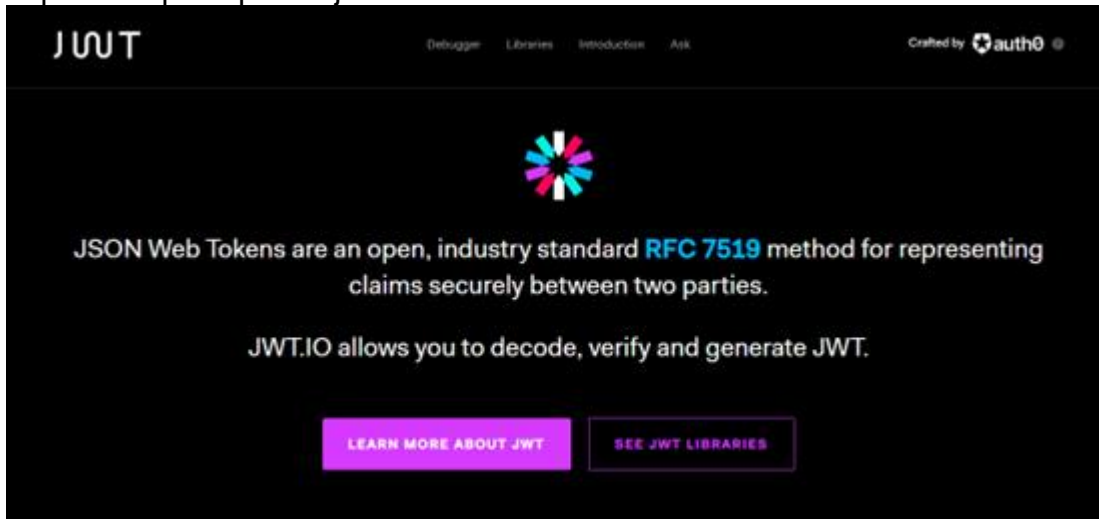
Request Params
<pre>{ "email": "prueba@gmail.com", "pass": "prueba123" }</pre>

En la respuesta se nos habrá generado un token, el cual utilizaremos a la hora de consumir el resto del api.

Response
<pre>{ "sucess": true, "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjYwNDQyMDQ5ZWQyMWZlMDNjYjQ2MDkzOSIsIm5pY2tuYW1lIjoIY2VzYXJfcndyIiwidHlwZSI6ImVzZXliLCJjbGllbnRzIjpbeysJfaWQiOiI2MDQ0MjFjNjN2V2kMjFmZTAzY2I0NjA5M2EiLCJuYW1lIjoIIGEgZGllbmRpdGEiLCJ0eXBlljoIY2xpZW50In0seyJfaWQiOiI2MDQ0MjRIMGVkMjFmZTAzY2I0NjA5M2IiLCJuYW1lIjoIUGhhbnRvbSBTdG9yZSIsInR5cGUiOiJjbGllbnQifV0sImNvbXBhbmlcyI6W3siaX2IkljoIjA0NjRiMWE4ZGVhMzZIMTI0OWVhNDNjliwibmFtZSI6IkkFsZG1pbiBzYS4gZGUgY3YiLCJ0eXBlljoIY29tcGFueSJ9LHsiaX2IkljoIjA0NjVIMWZmYmZlZDE5MjU5YjE0YTRmIiwibmFtZSI6ImdlbmVyYXVwW90b3JzIiwidHlwZSI6ImNvbXBhbmkifV0sImV4cCI6MTY1MzQ5NTc3NX0.kKuIXUuLdGR_T5MbJqf6rfVutGazZNVDjAOffuSfSSQ" }</pre>

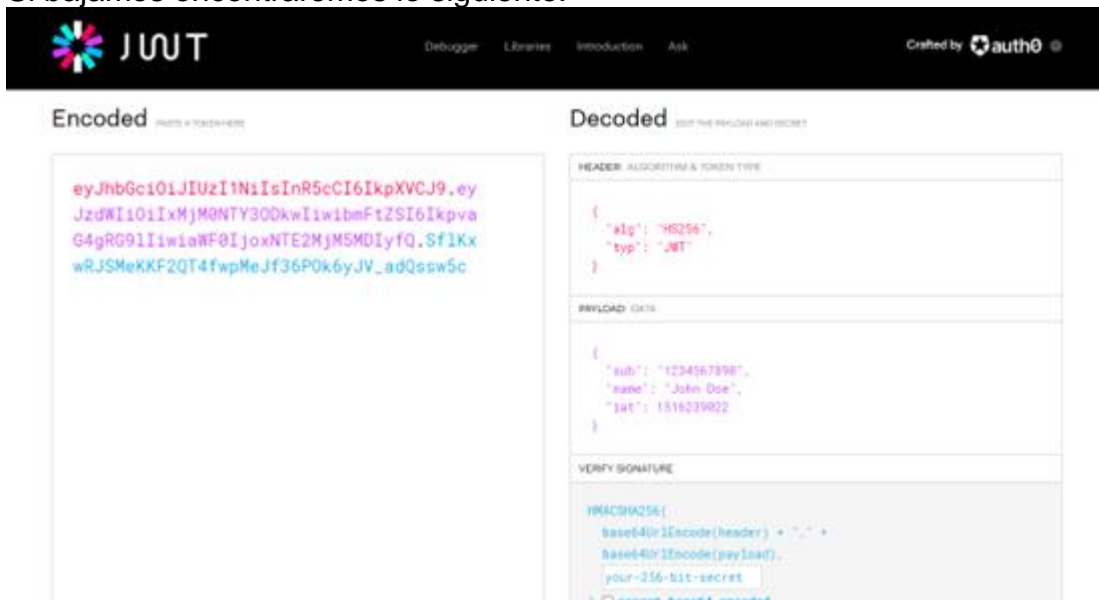
Intermedio

Lo que debemos hacer antes de continuar, es extraer la información que se encuentra en el token. Para esto debemos ir a [JSON Web Tokens - jwt.io](https://jwt.io) y veremos la pantalla principal de jwt.io.



Debugger

Si bajamos encontraremos lo siguiente:



En el cuadro de texto que dice encoded, nosotros borraremos el token por default y luego pegaremos nuestro token recibido.



Con esto tendríamos nuestra cabecera de autorización configurada. Recuerda que tu token tiene una vida útil de 1hr.

Clientes

Method	Path
POST	/clients

El payload del token contiene una lista de clientes, de esa lista agarramos los ids y los enviamos para obtener los datos de cada cliente.

Request Params

```
{
  "clients": [
    "604421c7ed21fe03cb46093a",
    "604424e0ed21fe03cb46093b"
  ]
}
```

Retorna la información de los clientes obtenidos

Response

```
{
  "sucess": true,
  "clients": []
}
```

Companies

Method	Path
POST	/companies

Ids de compañías

Request Params

```
{
  "companies": [
```



```
    "60464b1a8dea36e1249ea43c",  
    "60465e1ffbfed19259b14a4f"  
  ]  
}
```

información de compañías

Response
{ "sucess": true, "companies": [] }

Venta

Method	Path
POST	/venta

El cliente es quien debe realizar los cálculos.

Request Params
{ "cliente_id": "604421c7ed21fe03cb46093a", "vendedor_id": "60442049ed21fe03cb460939", "precio_tortilla": 12, "precio_totopos": 24, "venta": { "tortilla": 5, "totopos": 2 }, "merma": { "tortilla": 2, "totopos": 0 }, "fecha": 20220525, "total": 84 }

Response
{ "sucess": true, "msg": "Sucess operation", "something": "628ec69555578fe26cdd4d0e" }

Agendar

Method	Path
POST	/shedule

Request Params

```
{
  "cliente": {
    "_id": {
      "$oid": "60465e1ffbfed19259b14a4f"
    },
    "tienda": "general motors",
    "type": "company"
  },
  "tortilla": 3,
  "totopos": 2,
  "fecha_entrega": "2021-05-05T19:01:54.000Z",
  "vendedor": {
    "_id": {
      "$oid": "60442049ed21fe03cb460939"
    },
    "name": "cesar_rwr"
  }
}
```

Response

```
{
  "sucess": true,
  "msg": "Sucess operation",
  "something": "628ecc6855578fe26cdd4d0f"
}
```

Obtener pedidos agendados

Method	Path
GET	/agendas

No request params

Regresa los pedidos agendados

Response

```
{
  "sucess": true,
  "agenda": [
    {},
    {}
  ]
}
```

Obtener pedido agendado

El :id en la query debe sustituirse por un id de un pedido agendado. Se debió haber obtenido la lista de agendas primero para obtener un id.

Method	Path
GET	/agenda/schedule/:id

Response
{ "sucess": true, "schedule": {} }

Obtener cliente

Se debe contar con un id de empleado

Method	Path
GET	/agenda/client/:id

Response
{ "sucess": true, "client": {} }

Obtener cliente

Se debe contar con un id de compañía

Method	Path
GET	/agenda/company/:id

Response

```
{
  "sucess": true,
  "company": {}
}
```

Plan de pruebas

Como plan de pruebas se tiene contemplado postman para el api rest, y el complemento wizzler de la tienda de complementos para el navegador.

Forma de ejecución de los contenedores

Para ejecutar los contenedores, nos vamos a docker playground. Y clonamos nuestro repositorio.

```
$ git clone https://github.com/CesarRawr/proxy-test
```

Después nos ubicamos en el directorio

```
[node1] (local)
$ cd proxy-test
[node1] (local)
```

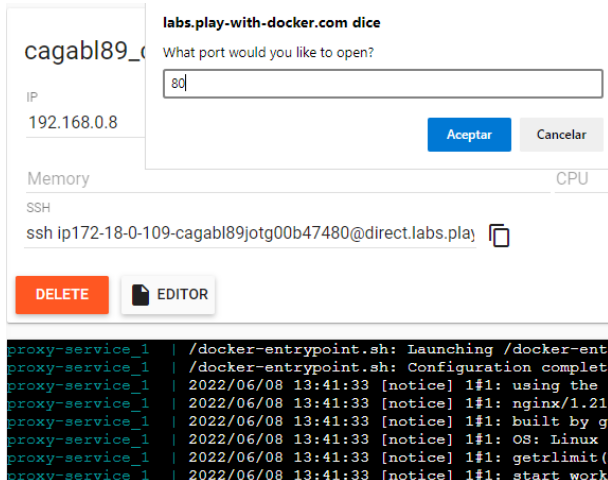
Y ejecutamos el comando

```
$ docker-compose up
```

Ahí comienza a ejecutarse

```
proxy-service_1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
proxy-service_1 | /docker-entrypoint.sh: Configuration complete; ready for start up
proxy-service_1 | 2022/06/08 13:41:33 [notice] 1#1: using the "epoll" event method
proxy-service_1 | 2022/06/08 13:41:33 [notice] 1#1: nginx/1.21.6
proxy-service_1 | 2022/06/08 13:41:33 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
proxy-service_1 | 2022/06/08 13:41:33 [notice] 1#1: OS: Linux 4.4.0-210-generic
proxy-service_1 | 2022/06/08 13:41:33 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
proxy-service_1 | 2022/06/08 13:41:33 [notice] 1#1: start worker processes
proxy-service_1 | 2022/06/08 13:41:33 [notice] 1#1: start worker process 30
proxy-service_1 | 2022/06/08 13:41:33 [notice] 1#1: start worker process 31
proxy-service_1 | 2022/06/08 13:41:33 [notice] 1#1: start worker process 32
proxy-service_1 | 2022/06/08 13:41:33 [notice] 1#1: start worker process 33
proxy-service_1 | 2022/06/08 13:41:33 [notice] 1#1: start worker process 34
proxy-service_1 | 2022/06/08 13:41:33 [notice] 1#1: start worker process 35
proxy-service_1 | 2022/06/08 13:41:33 [notice] 1#1: start worker process 36
proxy-service_1 | 2022/06/08 13:41:33 [notice] 1#1: start worker process 37
```

Despues abrimos el puerto 80



The screenshot shows a cloud console interface. At the top, a dialog box titled "labs.play-with-docker.com dice" asks "What port would you like to open?". The input field contains "80". There are "Aceptar" and "Cancelar" buttons. Below the dialog, the console shows the IP "192.168.0.8", memory and CPU usage bars, and an SSH command: "ssh ip172-18-0-109-cagabl89jotg00b47480@direct.labs.play". At the bottom, there are "DELETE" and "EDITOR" buttons. Below these is a terminal window showing logs for "proxy-service_1" with messages like "/docker-entrypoint.sh: Launching /docker-ent...", "/docker-entrypoint.sh: Configuration complete", and various notices about nginx version and OS.

Para usar el doker debemos usar los paths /rest y /soap para ejecutar las llamadas a sus respectivos servicios

Archivos de Dockerfile y docker-compose.yml

Este es el arhivo de configuración del proxy

```
server {
    listen 80;

    location /rest {
        resolver 8.8.8.8;
        proxy_pass http://tproject4is.herokuapp.com/user;
    }

    location /soap {
        resolver 8.8.8.8;
        proxy_pass http://tproject4is-soap.herokuapp.com/ws/pos.wsdl;
    }
}
```

Este es el dockerfile donde se copia el archivo de configuración para ejecutar el anterior

```
1 from nginx
2 copy ./default.conf /etc/nginx/conf.d/default.conf
```

Y este es el archivo del docker compose, lo único que hace es preparar el servicio de nginx para despues ejecutarlo con las configuraciones dadas.

FOLDERS

proxy

proxy

/* docker-compose.yml

indocker.txt

default.conf

docker-compose.yml

1

2

3

4

5

6

7

8

9

10

version: "3.3"

services:

proxy-service:

image: proxy-service

build:

context: ../proxy

dockerfile: Dockerfile

ports:

- "80:80"