- Carpeta middlewares
- Carpeta clients

Middleware Coolbox

El objetivo de este desarrollo es utilizar API privadas para hacer un consumo de datos correctamente, sin exponer tokens, de acuerdo al requerimiento presentado.

Carpeta middlewares

Cada middleware hace uso de respectivos clientes (Clientes Http que se encargan del consumo de datos)

Tenemos:

- validateCluster.ts
- validateProducts.ts
- validateSeller.ts
- validateSpecifications.ts
- validateIPProps.ts
- validateMarketing.ts, validateMundial.ts, validateGX.ts, validateVCO.ts

validateCluster.ts: El código usado aquí, es para utilizar el cliente Http (realizar consumo de datos) para **clusters** y obtener los datos.

```
const cl = await cluster.getCluster(id.toString())

   ctx.status = 200
   ctx.body = {
     page: cl.Page,
     size: cl.Size,
     result: cl
   }
```

para obtener el id, lo obtenemos así, e indicamos el client a usar:

```
const {
  vtex: {
    route: {
      params: { id },
    },
```

```
    },
    clients: { cluster },
  } = ctx
```

validateProducts.ts: El código usado aquí, es para utilizar el cliente Http (realizar consumo de datos) para **productos** y obtener los datos.

```
  //buscar esto dentro de catalog.d.ts que esta en node modules
    const [targetProduct] = await product.getProduct(id)

 if (!targetProduct) {
     ctx.status = 404

     return
   }

   const result:any = targetProduct
   ctx.status = 200
   ctx.body = {
     // CategoryId: result.CategoryId,
     LinkId: result.LinkId,
     Id: result.Id,
   }
```

para obtener el id del producto a consumir, lo ahcemos de la siguiente manera, indicando además el cliente product

```
  const {
     vtex: {
       route: {
         params: { id },
       },
     },
     clients: { product },
   } = ctx
```

validateSeller.ts

Código para consumir datos de el Seller, enviando el id, se convierte a tipo de datos string.

```
  const targetSeller = await catalog.seller(id.toString())

        ctx.status = 200
        ctx.body = {
```

```
            id: targetSeller.SellerId,
            name: targetSeller.Name,
            link: targetSeller.Description,
        }
```

Especificamos el cliente a usar y además obtenemos el id

```
  const {
        vtex: {
            route: {
                params: { id },
            },
        },
        clients: { catalog },
    } = ctx
```

validateSpecifications.ts

Código para obtener las especificaciones de un producto, con el id. Se usa el cliente **specifications**

```
  const targetSpecifications = await
specifications.getSpecifications(id.toString())

        ctx.status = 200
        ctx.body = {
            specifications: targetSpecifications
        }
        ctx.set('Cache-Control', 'no-cache')
```

Código para especificar el cliente y el id

```
const {
        vtex: {
            route: {
                params: { id },
            },
        },
        clients: { specifications },
    } = ctx
```

validateIPProps.ts

Código para obtener la dirección IP de la request.

```
const { request: { ip } } = ctx;
```

Código para validar que la IP pertenece a un conjunto de IP permitidas

```
const ips = ["200.48.243.162", "179.7.80.146"]
    const ok = ips.includes(ip)
    ctx.status = 200
    ctx.body = {
        ok
    }
```

validateVCO.ts

Este archivo está relacionado con funciones de master data. validateGX, validateMundial,validateMarketing.

Esto es para ocultar también los tokens al momento del consumo de datos.

Código para envíar datos a master data.

Se usa el cliente **vco**, y la función sendData para hacer el envío de los datos

Nota:

Para conseguir esta funcionalidad, se debe incluir esta línea de código a a Policies, de manifest.json

```
{
      "name": "ADMIN_DS"
}
```

```
const doc = await vco.sendData(body)
 // console.log('body ',JSON.stringify(body));

 if (!doc) {
   ctx.status = 404

   return
```

```
  }


  ctx.status = 200
```

Carpeta clients

Los clientes son usados para crear consumos de datos. Tenemos clientes nativos, y también, contamos con clientes custom.

cluster.ts

Se crea la función getCluster (clusterId), ya que se solicita el id como parámetro.

Es importante pasarle la URL a consumir, en este caso, que se trata de un cliente custom.

Así , logramos crear este cliente para nuestros consumos y después usarlos en los middlewares

```
export class ClusterClient extends JanusClient {
  public async getCluster(clusterId: string) {
    const collection = await this.http.get<Cluster>(
      `/api/catalog/pvt/collection/${clusterId}/products?page=1`,
      {
        headers: { VtexIdclientAutCookie: this.context.authToken },
      }
    )

    return collection
  }
}
```

product.ts

Debido a los requerimientos presentados, se han considerado realizar 2 consumos de datos, dentro de ese ciente.

```
// import { JanusClient } from '@vtex/api'
import {  Catalog, Product } from '@vtex/clients'
```

```
export class ProductClient extends Catalog{
  public async getProduct(productId: any) {
    const product = await this.http.get<Product[]>(
      `/api/catalog_system/pvt/sku/stockkeepingunitbyproductid/${productId}`,
      {
        headers: { VtexIdclientAutCookie: this.context.authToken },
      }
    )
    return product
  }
  public async getProductCatalog(productId:any) {
    const product = await this.http.get<Product>(
      `/api/catalog/pvt/product/${productId}`,
      {
        headers: { VtexIdclientAutCookie: this.context.authToken },
      }
    )
    return product
  }
}
```

Igual que en el caso anterior, se agrega la URL para el consumo de datos, incluyendo en los headers, los tokens generados para obetner esta data.

mundial.ts, marketing.ts, gx.ts

Estos clientes son similares, ya que están orientados al consumo de datos en Master Data. La diferencia con los anteriores consumos de datos, es que este cliente, usa el método **put**, el cual puede verse en el código en el **await**.

```
export class GXClient extends JanusClient {
    public async sendData(data: any) {
        // console.log("dataclient:", data);


        const doc = await this.http.put<DocGX>(
            `/api/dataentities/GX/documents`, data,
            {
                headers: { VtexIdclientAutCookie: this.context.authToken },
            }
        )

        return doc
    }
}
```

archivo clients/index.ts

Finalmente, una vez creados los clientes, se agregan en este archivo para pasar a ser usados en los middlewares. Consiste en importar los clientes y asignarles un key, con el cual se van a identificar en los middlewares.

```
  public get catalog() {
    return this.getOrSet('catalog', Catalog)
  }

  public get cluster() {
    return this.getOrSet('cluster', ClusterClient)
  }

  public get product() {
    return this.getOrSet('product', ProductClient)
  }

  public get mundial() {

    return this.getOrSet('mundial', MundialClient)
  }

  public get marketing() {
    return this.getOrSet('marketing', MarketingClient)
  }

  public get gx() {
    return this.getOrSet('gx', GXClient)
  }

  public get specifications() {
    return this.getOrSet('specifications', SpecificationsClient)
  }

  public get vco() {
    return this.getOrSet('vco', VCOClient)
  }
```

Carpeta node, index.js

Definir rutas , con los middlewares respectivos

```
export default new Service({

  clients,

  routes: {

    validateProductInstallments:method({

    GET: [validateProductInstallments]
```

```
}),
validateSeller:method({
GET: [validateSellerProps]
}),
validateCluster:method({
GET: [validateClusterProps]
}),
validateProductAlertShipping:method({
GET: [validateProductAlertShippingProps]
}),
validateProductImpulse:method({
GET: [validateProductImpulseProps]
}),
validateProductSpecifications:method({
GET: [validateSpecificationsProps]
}),
validateMundial:method({
PUT: [validateMundialProps]
}),
validateMarketing:method({
PUT: [validateMarketingProps]
}),
validateVCOProps:method({
PUT: [validateVCOProps]
}),
validateGXProps:method({
PUT:[validateGXProps]
}),
```

```
    validateIPProps:method({

    GET:[validateIPProps]

    })

  },

})
```

service.json

En este archivo, generalmente se establecen la rutas de consumo, para probar los servicios en Postman, junto con los middleware desarrollados.

```
{
  "memory": 256,
  "ttl": 10,
  "timeout": 2,
  "minReplicas": 2,
  "maxReplicas": 4,
  "workers": 1,
  "routes": {
    "validateProductInstallments": {
      "path": "/_v/validate-product-installments/:id",
      "public": true
    },
    "validateCluster": {
      "path": "/_v/validate-cluster/:id",
      "public": true
    },
    "validateProductAlertShipping": {
      "path": "/_v/validate-product-alert-shipping/:id",
      "public": true
    },
    "validateProductImpulse": {
      "path": "/_v/validate-product-impulse/:id",
      "public": true
    },
    "validateSeller": {
      "public": true,
      "path": "/_v/validate-seller/:id"
    },
    "validateMundial": {
      "path": "/_v/validate-mundial-form",
      "public": true
    },
    "validateMarketing": {
      "path": "/_v/validate-marketing-form",
      "public": true
    },
    "validateProductSpecifications":{
```

```
      "path": "/_v/validate-specifications/:id",
      "public":true
    },
    "validateVCOProps":{
      "path": "/_v/validate-vco",
      "public":true
    },
    "validateGXProps":{
      "path": "/_v/validate-garantia-extendida-form",
      "public":true
    },
    "validateIPProps":{
      "path": "/_v/ip",
      "public":true
    }
  }
}
```