

Cuarta Práctica de Aplicaciones Distribuidas

Alumno: Cruz Laparra Jose Julian

Descripción del problema: la empresa LaMejorCotizacion ha decidido iniciar la compra y venta de divisas, asegurando que sus precios son un 10% mejor que las cotizaciones existentes en el momento de una operación. La mencionada empresa basa toda su operación en una aplicación multiplataforma.

Para lograr esto, el arquitecto de la solución ha planteado el siguiente requerimiento para el grupo de desarrollo del Backend.

Se deben crear los servicios web necesarios para realizar lo siguiente:

1. Realizar una investigación de campo para determinar el sitio más confiable que muestre en una página Web las cotizaciones de las principales divisas.
2. Analizar la estructura del Sitio Web para aplicar una técnica de Web Scraping para recuperar el valor de compra y venta de una divisa. El caso de estudio y validación de la solución se hará con la moneda de Libra Esterlina.
3. Los valores recuperados usando Web Scraping serán alterados a la baja con una razón del 10%. Menos el 10% para que un usuario pueda comprar, y un 10% más para que el usuario pueda vender.
4. Se deben definir al menos dos servicios: uno para recuperar el valor de compra, y el otro para recuperar el valor de venta de la divisa.
5. Realizar una serie de pruebas para validar el funcionamiento de los servicios.
6. Documentar la implementación y describir lo necesario para agregar más divisas.

Anexo código en JS:

```
const express = require("express");
const axios = require("axios");
const cheerio = require("cheerio");

const app = express();
const PORT = 3000;

// -----
// Función: obtiene tasa base GBP/MXN
// -----
async function obtenerTasaBaseGBP_MXN() {
  const url = "https://themoneyconverter.com/GBP/MXN";

  const response = await axios.get(url, {
    headers: {
      "User-Agent": "Mozilla/5.0"
    }
  });

  const html = response.data;
  const $ = cheerio.load(html);

  // Extraemos texto plano
  const texto = $("body").text();

  // Buscamos: 1 GBP = XX.XXXX MXN
  const match = texto.match(/1\s*GBP\s*=\s*([0-9.,]+)\s*MXN/i);
  if (!match) return null;

  let valor = match[1];

  // Normalización del número
  valor = valor.replace(/,/g, "");
  const tasa = Number(valor);

  if (!Number.isFinite(tasa)) return null;

  return tasa;
}

// -----
// Endpoint raíz
// -----
app.get("/", (req, res) => {
  res.json({ mensaje: "Servidor activo" });
});

// -----
// Servicio COMPRA (-10%)
// -----
app.get("/compra/gbp-mxn", async (req, res) => {
  try {
    const base = await obtenerTasaBaseGBP_MXN();
    if (!base) {
```

```

        return res.status(500).json({
            error: "No se pudo obtener la tasa base"
        });
    }

    const compra = base * 0.90;

    res.json({
        par: "GBP/MXN",
        tipo: "compra",
        tasa_base: base,
        tasa_compra: Number(compra.toFixed(4)),
        formula: "compra = tasa_base * 0.90"
    });
} catch (error) {
    res.status(500).json({
        error: "Error interno en servicio de compra",
        detalle: error.message
    });
}

// -----
// Servicio VENTA (+10%)
// -----
app.get("/venta/gbp-mxn", async (req, res) => {
    try {
        const base = await obtenerTasaBaseGBP_MXN();
        if (!base) {
            return res.status(500).json({
                error: "No se pudo obtener la tasa base"
            });
        }
    }

    const venta = base * 1.10;

    res.json({
        par: "GBP/MXN",
        tipo: "venta",
        tasa_base: base,
        tasa_venta: Number(venta.toFixed(4)),
        formula: "venta = tasa_base * 1.10"
    });
} catch (error) {
    res.status(500).json({
        error: "Error interno en servicio de venta",
        detalle: error.message
    });
}

// -----
// Arranque del servidor
// -----
app.listen(PORT, () => {
    console.log(`Servidor corriendo en http://localhost:${PORT}`);
}

```

```
});
```

Nota: Se definió un encabezado User-Agent estándar para evitar bloqueos básicos del servidor web, sin afectar el contenido obtenido, ya que la página no depende del navegador para mostrar la cotización.

Pruebas en Postman:

- Servidor activo:

The screenshot shows the Postman interface with a GET request to `http://localhost:3000/`. The response status is `200 OK` with a response time of 31 ms and a body size of 264 B. The response JSON is:

```
1 {  
2   "mensaje": "Servidor activo"  
3 }
```

- Servicio Compra:

The screenshot shows the Postman interface with a POST request to `http://localhost:3000/compra/gbp-mxn`. The response status is `200 OK` with a response time of 594 ms and a body size of 346 B. The response JSON is:

```
1 {  
2   "par": "GBP/MXN",  
3   "tipo": "compra",  
4   "tasa_base": 24.15,  
5   "tasa_compra": 21.735,  
6   "formula": "compra = tasa_base * 0.90"  
7 }
```

- Servicio Venta:

The screenshot shows the Postman application interface. At the top, there's a navigation bar with tabs like 'Search Postman', 'Invite', 'Upgrade', and environment dropdowns. Below the header, a URL bar shows 'http://localhost:3000/venta/gbp-mxn'. The main area has a 'GET' method selected, and the URL is displayed in a blue-bordered input field. Below the URL, there are tabs for 'Docs', 'Params', 'Authorization', 'Headers (7)', 'Body', 'Scripts', and 'Settings'. The 'Body' tab is active, showing options for 'none', 'form-data', 'x-www-form-urlencoded', 'raw' (which is selected), 'binary', 'GraphQL', and 'JSON'. The 'JSON' option is also highlighted with a dropdown arrow. The response section shows a green '200 OK' status with a '390 ms' latency and '343 B' size. The response body is displayed as JSON code:

```
1 {  
2   "par": "GBP/MXN",  
3   "tipo": "venta",  
4   "tasa_base": 24.15,  
5   "tasa_venta": 26.565,  
6   "formula": "venta = tasa_base * 1.10"  
7 }
```