TRANSFERENCIAS Y PAGOS (WEB)

Este proyecto integra la versión de Java de escritorio en un proyecto Spring Boot (Java 8). Tiene dos formatos el del **Cuaderno 34_14** y el **ISO 20022**.

Dependencias: Spring Data JDBC, Spring Data JPA, Spring Web, Thymeleaf, MySQL Driver y OpenCSV.

Cuaderno 34_14

Lo primero es crear la clase Pago.java que coincide con los campos de la tabla pagoc34_14 de la base de datos MySQL.

```java
@Entity
@Table(name = "pagoc34_14")
public class Pago {

    @Id
    private String id;
    2 usages
    private String pagos;
    2 usages
    private String normal;
    2 usages
    private String cifOrdenante;
    2 usages
    private String numero;
    2 usages
    private String fechaOperacion;
    2 usages
    private String ibanOrdenante;
    2 usages
    private String concepto;
    2 usages
    private String nombreOrdenante;
    2 usages
    private String ibanBeneficiario;
    2 usages
    private Double importe;
    2 usages
    private String nombreBeneficiario;
    2 usages
    private String gastos;
```

(+ Getters y Setters)

Después he creado una clase @Service llamada CsvService.java en la que se desarrollan los métodos tanto para crear un CSV con los datos de una tabla de MySQL como para insertar datos en la tabla a partir de un archivo CSV.

```java
@Service
public class CsvService {

    @Autowired
    private EntityManager entityManager;

    9 usages
    private static final Logger logger = Logger.getLogger(CsvService.class.getName());
```

Aquí tenemos el método exportDataToCSV(String) que coge los datos de la tabla correspondiente a la clase Pago.java y genera un archivo CSV donde los inserta por filas y sin encabezados, ya que el formato del cuaderno 34_14 es sin encabezados.

```java
@Transactional
public void exportDataToCSV(String filePath) {
    logger.info( msg: "Iniciando exportación de datos a CSV");

    try (CSVWriter writer = new CSVWriter(new FileWriter(filePath), separator: ';', CSVWriter.NO_QUOTE_CHARACTER, CSVWriter.DEFAULT_E
        // Consulta para obtener los datos de la tabla
        Query query = entityManager.createQuery( qlString: "SELECT e.pagos, e.normal, e.cifOrdenante, e.numero, e.fechaOperacion, e.ib
        List<Object[]> resultList = query.getResultList();

        logger.info( msg: "Número de registros obtenidos: " + resultList.size());

        // Escribir los datos en el archivo CSV
        for (Object[] row : resultList) {
            String[] rowData = new String[row.length];
            for (int i = 0; i < row.length; i++) {
                rowData[i] = row[i] != null ? row[i].toString() : "";
            }
            writer.writeNext(rowData);
        }

        logger.info( msg: "Datos exportados exitosamente a " + filePath);
    } catch (IOException ex) {
        logger.severe( msg: "Error al escribir en el archivo CSV: " + ex.getMessage());
        ex.printStackTrace();
    } catch (Exception e) {
        logger.severe( msg: "Error inesperado: " + e.getMessage());
        e.printStackTrace();
    }
}
```

Después tenemos el método importDataFromCSV(String) en el cual se pasa la ruta de un archivo CSV por parámetro, se lee y se insertan los registros en la tabla c34_14 de la base de datos.

```
@Transactional
public void importDataFromCSV(String filePath) {
    logger.info( msg: "Iniciando importación de datos desde CSV");

    try {
        CSVParser parser = new CSVParserBuilder()
                .withSeparator(';')
                .build();

        CSVReader reader = new CSVReaderBuilder(new FileReader(filePath))
                .withCSVParser(parser)
                .build();
        String[] nextLine;
        while ((nextLine = reader.readNext()) != null) {
            Pago pago = new Pago();
            pago.setId(generateId());
            pago.setPagos(nextLine[0]);
            pago.setNormal(nextLine[1]);
            pago.setCifOrdenante(nextLine[2]);
            pago.setNumero(nextLine[3]);
            pago.setFechaOperacion(nextLine[4]);
            pago.setIbanOrdenante(nextLine[5]);
            pago.setConcepto(nextLine[6]);
            pago.setNombreOrdenante(nextLine[7]);
            pago.setIbanBeneficiario(nextLine[8]);
            pago.setImporte(Double.parseDouble(nextLine[9]));
            pago.setNombreBeneficiario(nextLine[10]);
            pago.setGastos(nextLine[11]);

            entityManager.persist(pago);
        }
        logger.info( msg: "Datos importados exitosamente desde " + filePath);
    } catch (IOException ex) {
        logger.severe( msg: "Error al leer el archivo CSV: " + ex.getMessage());
        ex.printStackTrace();
    } catch (Exception e) {
        logger.severe( msg: "Error inesperado: " + e.getMessage());
        e.printStackTrace();
    }
}
```

El método generateId() lo que hace es generar un ID aleatorio para el identificador de pago, que comprenda entre 3 y 5 caracteres formados por números y letras.

```
private String generateId() {
    Random random = new Random();
    String characters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    int minLength = 3;
    int maxLength = 5;
    int length = random.nextInt( bound: maxLength - minLength + 1) + minLength;
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < length; i++) {
        int randomIndex = random.nextInt(characters.length());
        sb.append(characters.charAt(randomIndex));
    }
    return sb.toString();
}
```

Y en la clase Controller primero se declara la clase service:

```
@Autowired
private CsvService csvService;
```

Después los GetMapping para mostrar las páginas en las que accederemos a los métodos anteriores y los GetMapping para ejecutar los métodos:

```
@GetMapping("/c34_14_export")
public String exportCSVPage() { return "export"; }

@GetMapping("/c34_14_import")
public String importCSVPage() { return "import"; }
```

```
@GetMapping("/c34_14_export/export")
public ResponseEntity<String> exportCSV(@RequestParam("saveLocation") String saveLocation) {
    try {
        // Construir la ruta completa del archivo CSV
        String filePath = Paths.get(saveLocation).toAbsolutePath().toString();
        csvService.exportDataToCSV(filePath);
        return ResponseEntity.ok( body: "Archivo CSV creado exitosamente.");
    } catch (Exception e) {
        e.printStackTrace();
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error al crear el archivo CSV.");
    }
}
```

```java
@PostMapping("/c34_14_import/import")
public ResponseEntity<String> handleFileUpload(@RequestParam("file") MultipartFile file) {
    try {
        if (file.isEmpty()) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Archivo vacío.");
        }

        // Definir la ruta donde se guardará el archivo
        String tempDirectory = "C:/path/to/temp/";
        File directory = new File(tempDirectory);

        // Crear la ruta si no existe
        if (!directory.exists()) {
            directory.mkdirs();
        }

        // Guardar el archivo temporalmente
        String filePath = tempDirectory + file.getOriginalFilename();
        File destFile = new File(filePath);
        file.transferTo(destFile);

        // Importar los datos del archivo CSV
        csvService.importDataFromCSV(filePath);

        return ResponseEntity.ok( body: "Datos importados exitosamente desde el archivo CSV.");
    } catch (IOException e) {
        e.printStackTrace();
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error al guardar el archivo.");
    } catch (Exception e) {
        e.printStackTrace();
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error al importar los datos desde
    }
```

ISO 20022

El formato ISO 20022 es en XML y comprende de una cabecera (idMensaje, fecha, hora, numeroOperaciones, controlSuma) y el cuerpo que son los pagos/transferencias. Lo primero es la clase PagoISO20022.java que corresponde a la tabla pagoiso20022 de la base de datos MySQL.

```java
@Entity
@Table(name = "pagoiso20022")
public class PagoISO20022 {

    @Id
    private String identificadorPago;
    2 usages
    private int idMensaje;
    2 usages
    private String medioPago;
    2 usages
    private String indicadorApunteCuenta;
    2 usages
    private String informacionTipoPago;
    2 usages
    private String fecha;
    2 usages
    private String nombreOrdenante;
    2 usages
    private String nifOrdenante;
    2 usages
    private String cuentaOrdenante;
    2 usages
    private String nombreBeneficiario;
    2 usages
    private String nifBeneficiario;
    2 usages
    private String cuentaBeneficiario;
    2 usages
    private Double controlSuma;
```

La clase XMLService.java es la que contiene los dos principales métodos para crear un XML a partir de los datos de una tabla MySQL y para crear el XML a partir de un archivo CSV.

```java
@Service
public class XMLService {

    @Autowired
    private EntityManager entityManager;

    4 usages
    private static final Logger logger = Logger.getLogger(XMLService.class.getName());
```

El primer método es el de exportar datos a XML desde la base de datos.

```java
@Transactional
public void exportDataToXML(String filePath, String idMensaje) {
```

En este método he utilizado un DocumentBuilderFactory para crear el archivo XML. Para los elementos (raíz, cabecera, cuerpo…) he utilizado la clase Element. Se agrega una cabecera con sus datos. Para el NumeroOperaciones se calcula la cantidad de registros que hay y para el controlSuma se suman todos los controlSuma de los pagos. Después viene el cuerpo, donde se leen los pagos registrados en la tabla y los inserta en el XML.

El siguiente método lo que hace es crear un XML con los datos exportados de un CSV.

```java
public void exportToXMLfromCSV(String filePath) {
```

En este método se usa otro método para añadir los campos del cuerpo llamado agregarElemento:

```java
private void agregarElemento(Document document, Element parentElement, String tagName, String textContent) {
    Element element = document.createElement(tagName);
    element.appendChild(document.createTextNode(textContent));
    parentElement.appendChild(element);
}
```

Y en el método anterior se usa así:

```
while ((line = br.readLine()) != null) {
    String[] fields = line.split( regex: ",");

    if (fields.length == 12) {
        Element pagoElement = doc.createElement( tagName: "pago");
        cuerpoElement.appendChild(pagoElement);

        agregarElemento(doc, pagoElement,  tagName: "identificadorPago", fields[0]);
        agregarElemento(doc, pagoElement,  tagName: "nombre", fields[1]);
        agregarElemento(doc, pagoElement,  tagName: "direccion", fields[2]);
        agregarElemento(doc, pagoElement,  tagName: "cp", fields[3]);
        agregarElemento(doc, pagoElement,  tagName: "poblacion", fields[4]);
        agregarElemento(doc, pagoElement,  tagName: "provincia", fields[5]);
        agregarElemento(doc, pagoElement,  tagName: "nif", fields[6]);
        agregarElemento(doc, pagoElement,  tagName: "ccc", fields[7]);
        agregarElemento(doc, pagoElement,  tagName: "iban", fields[8]);
        agregarElemento(doc, pagoElement,  tagName: "bic", fields[9]);
        agregarElemento(doc, pagoElement,  tagName: "controlSuma", fields[10]);

        numeroOperaciones++;
        controlSumaTotal += Double.parseDouble(fields[10].replace( target: ",",  replacement: "."))
    }
}
```

Así está en el Controller:

```
@GetMapping("/genXML")
public String genXMLPage() { return "genXMLfromBD"; }

@GetMapping("/genXMLdesdeCSV")
public String genXMLdesdeCSVPage() { return "genXMLfromCSV"; }
```

```java
@GetMapping("/genXML/generar")
public ResponseEntity<String> generateXMLFromDatabase(@RequestParam("saveLocation") String saveLocation) {
    try {
        String idMensaje = "1";
        String filePath = Paths.get(saveLocation).toAbsolutePath().toString();
        xmlService.exportDataToXML(filePath, idMensaje);
        return ResponseEntity.ok( body: "Archivo XML creado exitosamente desde la base de datos.");
    } catch (Exception e) {
        e.printStackTrace();
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error al crear el archivo XML desde la base de datos.");
    }
}

@GetMapping("/genXMLdesdeCSV/generar")
public ResponseEntity<String> generateXMLFromCSV(@RequestParam("saveLocation") String saveLocation) {
    try {
        String filePath = Paths.get(saveLocation).toAbsolutePath().toString();
        xmlService.exportToXMLfromCSV(saveLocation);
        return ResponseEntity.ok( body: "Archivo XML creado exitosamente a partir del archivo CSV.");
    } catch (Exception e) {
        e.printStackTrace();
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error al crear el archivo XML desde el archivo CSV.");
    }
}
```

Y así es como se ve la web: