CIFP FRANCESC DE BORJA MOLL

PROYECTO S€PA

Documentación del proyecto SEPA

Presentado por:
Aitor Bauzá Gómez

TABLA DE CONTENIDO

1. RESUMEN EJECUTIVO

1.1. DESCRIPCIÓN GENERAL DEL PROYECTO:

En primer lugar, el proyecto consiste en la implementación de un sistema de pagos SEPA (Single Euro Payments Area) mediante una Iniciativa de Débito Directo (Direct Debit Initiation). Este sistema está diseñado para procesar pagos automatizados de manera eficiente y segura. Además, facilita las transacciones entre las personas obligadas al pago (deudores) y los beneficiarios (acreedores) dentro del marco de las normativas SEPA.

Por otro lado, el sistema del backend se encarga de generar los archivos XML conforma a la normativa pain.008.001.02. Esta normativa es el estándar utilizado para el envió de dinero a través de Débitos Directos SEPA. Asimismo, este archivo tiene el contenido necesario para ejecutar una transacción, incluyendo detalles como:

- El identificador del mandato.
- El importe de la transferencia.
- La cuenta del deudor.
- La cuenta del acreedor.
- La entidad bancaria involucrada.

Finalmente, el frontend está orientado a dar una interfaz de usuario clara y concisa, que permita a los usuarios ingresar y gestionar la información de los pagos de manera sencilla. Asimismo, la interfaz de conecta al backend para generar los archivos XML adecuados y realizar los pagos de forma automatizada, cumpliendo con los requisitos establecidos en el protocolo SEPA.

1.2. ALCANCE Y BENEFICIOS:

1.2.1. ALCANCE:

- Implementación de un sistema de Débito Directo SEPA que permita la creación y gestión de transacciones electrónicas de forma automatizada.
- Desarrollo de una interfaz gráfica que permita la visualización de los pagos a realizar y la gestión de la información necesaria.
- Generar archivos XML conforme a la normativa pain.008.001.02, la cual garantiza la compatibilidad con entidades bancarias y otras instituciones financieras dentro del área SEPA.

1.2.2. BENEFICIOS:

1.2.2.1. EFICIENCIA EN LOS PAGOS:

Automatización de procesos que anteriormente eran manuales, reduciendo el tiempo y esfuerzo necesario para realizar las transacciones.

1.2.2.2. SEGURIDAD EN LAS TRANSCACCIONES:

Cumpliendo con los estándares y regulaciones de SEPA, lo que asegura la fiabilidad y seguridad en la transferencia de fondos.

1.2.2.3. FACILIDAD DE USO:

La interfaz gráfica facilita a los usuarios la interacción con el sistema, permitiendo una experiencia instintiva y eficiente para ingresar y gestionar los pagos.

1.2.2.4. FLEXIBILIDAD:

El sistema es fácilmente adaptable para un número mayor de usuarios y transacciones en el futuro.

1.2.2.5. CUMPLIMIENTO NORMATIVO:

La implementación respeta las normativas SEPA, garantizando que todas las transacciones se realicen conforme a las leyes y regulaciones de la zona europea.

2. ROLES Y PERSONAS EN EL GRUPO:

Roles	Nombres de los integrantes
JEFE DE EQUIPO	Luciano Nicolás Viscio Frank
METATESTER	Elm Ramis Navarro
DOCUMENTADOR	Aitor Bauzá Gómez
DISEÑADOR	Antonio Picornell Perdigón
PROGRAMADOR ENCARGADO DEL FRONT-END	Luciano Nicolás Viscio Frank
PROGRAMADOR ENCARGADO DEL BACK-END	Luciano Nicolás Viscio Frank
ANALISTA	Elm Ramis Navarro

2.1. INFORME DEL TRABAJO:

2.1.1. 4 DE OCTUBRE DE 2024:

TABLA DE CONTROL DE AVANCE DE PROCESOS						
ROLES	NOMBRE	APELLIDOS	FECHA INICIAL	FECHA FINAL	DIAS	PROGRESO
Diseñador de diagramas	Antonio	Picornell Perdigón	**04/oct/2024**	**11/oct/2024**	7	0%
Programador Back-End	Luciano Nicolás	Viscio Frank	**14/oct/2024**	**25/oct/2024**	11	-90,9%
Programador Front-End	Luciano Nicolás	Viscio Frank	**26/oct/2024**	**04/nov/2024**	9	-100,0%
Analista	Elm	Ramis Navarro	**04/nov/2024**	**15/nov/2024**	11	-100,0%
Documentador	Aitor	Bauzá Gómez	**04/oct/2024**	**27/nov/2024**	54	0%
Metatester	Elm	Ramis Navarro	**15/nov/2024**	**22/nov/2024**	7	-100,0%

2.1.2. 7 DE OCTUBRE DE 2024:

TABLA DE CONTROL DE AVANCE DE PROCESOS						
ROLES	NOMBRE	APELLIDOS	FECHA INICIAL	FECHA FINAL	DIAS	PROGRESO
Diseñador de diagramas	Antonio	Picornell Perdigón	**04/oct/2024**	**11/oct/2024**	7	42,9%
Programador Back-End	Luciano Nicolás	Viscio Frank	**14/oct/2024**	**25/oct/2024**	11	-63,6%
Programador Front-End	Luciano Nicolás	Viscio Frank	**26/oct/2024**	**04/nov/2024**	9	-100,0%
Analista	Elm	Ramis Navarro	**04/nov/2024**	**15/nov/2024**	11	-100,0%
Documentador	Aitor	Bauzá Gómez	**04/oct/2024**	**27/nov/2024**	54	5,6%
Metatester	Elm	Ramis Navarro	**15/nov/2024**	**22/nov/2024**	7	-100,0%

2.1.3. 14 DE OCTUBRE DE 2024:

TABLA DE CONTROL DE AVANCE DE PROCESOS						
ROLES	NOMBRE	APELLIDOS	FECHA INICIAL	FECHA FINAL	DIAS	PROGRESO
Diseñador de diagramas	Antonio	Picornell Perdigón	**04/oct/2024**	**11/oct/2024**	7	100,0%
Programador Back-End	Luciano Nicolás	Viscio Frank	**14/oct/2024**	**25/oct/2024**	11	0%
Programador Front-End	Luciano Nicolás	Viscio Frank	**26/oct/2024**	**04/nov/2024**	9	-100,0%
Analista	Elm	Ramis Navarro	**04/nov/2024**	**15/nov/2024**	11	-100,0%
Documentador	Aitor	Bauzá Gómez	**04/oct/2024**	**27/nov/2024**	54	18,5%
Metatester	Elm	Ramis Navarro	**15/nov/2024**	**22/nov/2024**	7	-100,0%

2.1.4. 21 DE OCTUBRE DE 2024:

TABLA DE CONTROL DE AVANCE DE PROCESOS						
ROLES	NOMBRE	APELLIDOS	FECHA INICIAL	FECHA FINAL	DIAS	PROGRESO
Diseñador de diagramas	Antonio	Picornell Perdigón	**04/oct/2024**	**11/oct/2024**	7	100,0%
Programador Back-End	Luciano Nicolás	Viscio Frank	**14/oct/2024**	**25/oct/2024**	11	63,6%
Programador Front-End	Luciano Nicolás	Viscio Frank	**26/oct/2024**	**04/nov/2024**	9	-55,6%
Analista	Elm	Ramis Navarro	**04/nov/2024**	**15/nov/2024**	11	-100,0%
Documentador	Aitor	Bauzá Gómez	**04/oct/2024**	**27/nov/2024**	54	31,5%
Metatester	Elm	Ramis Navarro	**15/nov/2024**	**22/nov/2024**	7	-100,0%

2.1.5. 25 DE OCTUBRE DE 2024:

TABLA DE CONTROL DE AVANCE DE PROCESOS						
ROLES	NOMBRE	APELLIDOS	FECHA INICIAL	FECHA FINAL	DIAS	PROGRESO
Diseñador de diagramas	Antonio	Picornell Perdigón	**04/oct/2024**	**11/oct/2024**	7	100,0%
Programador Back-End	Luciano Nicolás	Viscio Frank	**14/oct/2024**	**25/oct/2024**	11	100,0%
Programador Front-End	Luciano Nicolás	Viscio Frank	**26/oct/2024**	**04/nov/2024**	9	-11,1%
Analista	Elm	Ramis Navarro	**04/nov/2024**	**15/nov/2024**	11	-90,9%
Documentador	Aitor	Bauzá Gómez	**04/oct/2024**	**27/nov/2024**	54	38,9%
Metatester	Elm	Ramis Navarro	**15/nov/2024**	**22/nov/2024**	7	-100,0%

2.1.6. 4 DE NOVIEMBRE DE 2024:

TABLA DE CONTROL DE AVANCE DE PROCESOS						
ROLES	NOMBRE	APELLIDOS	FECHA INICIAL	FECHA FINAL	DIAS	PROGRESO
Diseñador de diagramas	Antonio	Picornell Perdigón	**04/oct/2024**	**11/oct/2024**	7	100,0%
Programador Back-End	Luciano Nicolás	Viscio Frank	**14/oct/2024**	**25/oct/2024**	11	100,0%
Programador Front-End	Luciano Nicolás	Viscio Frank	**26/oct/2024**	**04/nov/2024**	9	100,0%
Analista	Elm	Ramis Navarro	**04/nov/2024**	**15/nov/2024**	11	0,0%
Documentador	Aitor	Bauzá Gómez	**04/oct/2024**	**27/nov/2024**	54	57,4%
Metatester	Elm	Ramis Navarro	**15/nov/2024**	**22/nov/2024**	7	-100,0%

2.1.7. 7 DE NOVIEMBRE DE 2024:

TABLA DE CONTROL DE AVANCE DE PROCESOS						
ROLES	NOMBRE	APELLIDOS	FECHA INICIAL	FECHA FINAL	DIAS	PROGRESO
Diseñador de diagramas	Antonio	Picornell Perdigón	**04/oct/2024**	**11/oct/2024**	7	100,0%
Programador Back-End	Luciano Nicolás	Viscio Frank	**14/oct/2024**	**25/oct/2024**	11	100,0%
Programador Front-End	Luciano Nicolás	Viscio Frank	**26/oct/2024**	**04/nov/2024**	9	100,0%
Analista	Elm	Ramis Navarro	**04/nov/2024**	**15/nov/2024**	11	27,3%
Documentador	Aitor	Bauzá Gómez	**04/oct/2024**	**27/nov/2024**	54	63,0%
Metatester	Elm	Ramis Navarro	**15/nov/2024**	**22/nov/2024**	7	-100,0%

2.1.8. 15 DE NOVIEMBRE DE 2024:

TABLA DE CONTROL DE AVANCE DE PROCESOS						
ROLES	NOMBRE	APELLIDOS	FECHA INICIAL	FECHA FINAL	DIAS	PROGRESO
Diseñador de diagramas	Antonio	Picornell Perdigón	**04/oct/2024**	**11/oct/2024**	7	100,0%
Programador Back-End	Luciano Nicolás	Viscio Frank	**14/oct/2024**	**25/oct/2024**	11	100,0%
Programador Front-End	Luciano Nicolás	Viscio Frank	**26/oct/2024**	**04/nov/2024**	9	100,0%
Analista	Elm	Ramis Navarro	**04/nov/2024**	**15/nov/2024**	11	100,0%
Documentador	Aitor	Bauzá Gómez	**04/oct/2024**	**27/nov/2024**	54	77,8%
Metatester	Elm	Ramis Navarro	**15/nov/2024**	**22/nov/2024**	7	0,0%

2.1.9. 22 DE NOVIEMBRE DE 2024:

TABLA DE CONTROL DE AVANCE DE PROCESOS						
ROLES	NOMBRE	APELLIDOS	FECHA INICIAL	FECHA FINAL	DIAS	PROGRESO
Diseñador de diagramas	Antonio	Picornell Perdigón	**04/oct/2024**	**11/oct/2024**	7	100,0%
Programador Back-End	Luciano Nicolás	Viscio Frank	**14/oct/2024**	**25/oct/2024**	11	100,0%
Programador Front-End	Luciano Nicolás	Viscio Frank	**26/oct/2024**	**04/nov/2024**	9	100,0%
Analista	Elm	Ramis Navarro	**04/nov/2024**	**15/nov/2024**	11	100,0%
Documentador	Aitor	Bauzá Gómez	**04/oct/2024**	**27/nov/2024**	54	90,7%
Metatester	Elm	Ramis Navarro	**15/nov/2024**	**22/nov/2024**	7	100,0%

2.1.10.27 DE NOVIEMBRE DE 2024:

TABLA DE CONTROL DE AVANCE DE PROCESOS						
ROLES	NOMBRE	APELLIDOS	FECHA INICIAL	FECHA FINAL	DIAS	PROGRESO
Diseñador de diagramas	Antonio	Picornell Perdigón	**04/oct/2024**	**11/oct/2024**	7	100,0%
Programador Back-End	Luciano Nicolás	Viscio Frank	**14/oct/2024**	**25/oct/2024**	11	100,0%
Programador Front-End	Luciano Nicolás	Viscio Frank	**26/oct/2024**	**04/nov/2024**	9	100,0%
Analista	Elm	Ramis Navarro	**04/nov/2024**	**15/nov/2024**	11	100,0%
Documentador	Aitor	Bauzá Gómez	**04/oct/2024**	**27/nov/2024**	54	100,0%
Metatester	Elm	Ramis Navarro	**15/nov/2024**	**22/nov/2024**	7	100,0%

3. GUÍA DE CONTRIBUCIÓN

3.1. PROTOCOLO PARA CONTRIBUIR AL PROYECTO:

Para contribuir al proyecto, se deben seguir los siguientes puntos:

3.1.1. CLONAR EL REPOSITORIO GITHUB:

- Los colaboradores deben clonar el repositorio oficial (https://github.com/eramis152/proyecto-SEPA) en el entorno local.
- Comando: git close https://github.com/eramis152/proyecto-SEPA

3.1.2. CREAR UNA RAMA PARA LA FUNCIONALIDAD:

- Antes de realizar cualquier cambio, hay que crear una rama nueva que describa brevemente la funcionalidad que se va a implementar:
- Comando: git checkout – b nombre_de_rama

3.1.3. IMPLEMENTAR CAMBIOS Y ESCRIBIR CÓDIGO:

- Realizar las modificaciones necesarias.
- Asegurarse de que todo el código se adapte a los estándares establecidos.

3.1.4. PRUEBAS LOCALES:

- Antes de hacer un *commit*, hay que asegurarse de haber ejecutado las pruebas necesarias para verificar que los cambios no introducen errores y que el sistema funciona correctamente.

3.1.5. HACER COMMIT DE LOS CAMBIOS:

- Hay que escribir mensajes de *commit* claros y descriptivos que expliquen los cambios hechos:
- Comando: git commit -m “Mensaje”

3.1.6. SINCRONIZAR Y RESOLVER CONFLICTOS:

- Antes de hacer un *push*, tenemos que sincronizar la rama con la rama principal para poder resolver posibles conflictos:
- Primer comando: git fetch origin
- Segundo comando: git merge origin/main

3.1.7. PUSH DE LOS CAMBIOS A GITHUB:

- Hay que enviar los cambios al repositorio en GitHub:
- Comando: git push origin nombre_de_rama

3.1.8. CREAR UN PULL REQUEST (PR):

- Una vez que los cambios estén subidos al repositorio, abrimos un *Pull Request* en el repositorio principal.
- En la descripción del *Pull Request*, tenemos que explicar claramente el que hemos cambiado, el por qué, y si se necesita alguna revisión o prueba específica.

3.1.9. REVISIÓN DE CÓDIGO Y APROBACIÓN:

- Hay que realizar los ajustes necesarios en función de los comentarios recibidos.

3.2. ESTÁNDARES DE CALIDAD Y REVISIÓN DE CÓDIGO:

Para mantener los estándares de calidad hay que seguir la siguiente guía:

3.2.1. ESTRUCTURA Y FORMATO DEL CÓDIGO:

- El código XML debe ser legible y estructurado de manera clara, usando indentación.
- Las etiquetas XML han de cumplir con los estándares SEPA y estar correctamente indentadas.
- Debe haber un uso de comentarios cuando sea necesario para explicar las partes complejas o específicas del código.

3.2.2. USO DE LINTERS Y FORMATEADORES:

- Asegurarse de usar herramientas de *linters* para garantizar la consistencia del código.
- Utilizar formateadores automáticos para mantener un estilo idéntico.

3.2.3. PRUEBAS DE VALIDACIÓN:

- Realiza pruebas exhaustivas para validar la generación de ficheros XML y su conformidad con el estándar **pain.008.001.02**.
- Comprobar los datos obligatorios que se validarán correctamente y que el sistema maneje errores de forma adecuada.

3.2.4. REVISIÓN DE CÓDIGO:

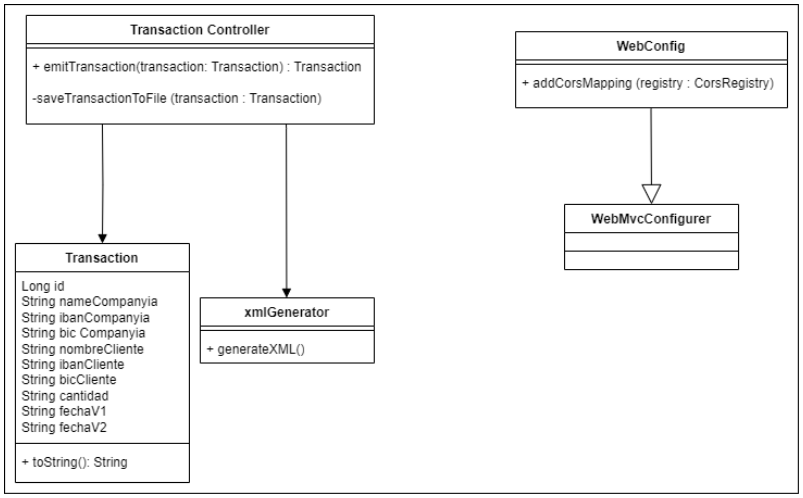
- Todo *Pull Request* ha de ser revisado por al menos otro colaborador del proyecto.
- Se recomienda verificar los cambios que no introducen problemas de rendimiento, fallos de seguridad o errores funcionales.

3.2.5. DOCUMENTACIÓN:

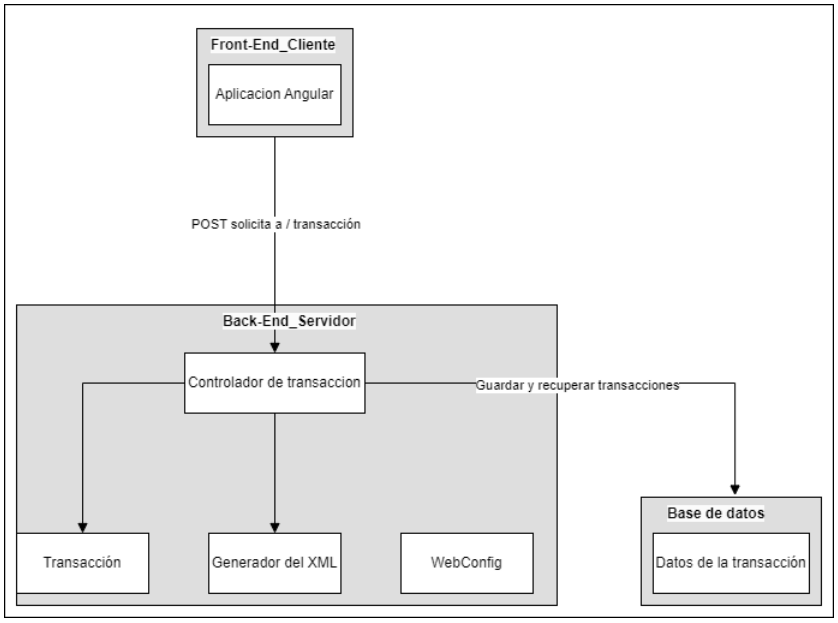
- Cualquier nueva funcionalidad o cambio significativo ha de estar bien documentado en el repositorio.
- Actualizar los archivos README.md y cualquier otra documentación importante si es necesario.

4. DIAGRAMAS UML:

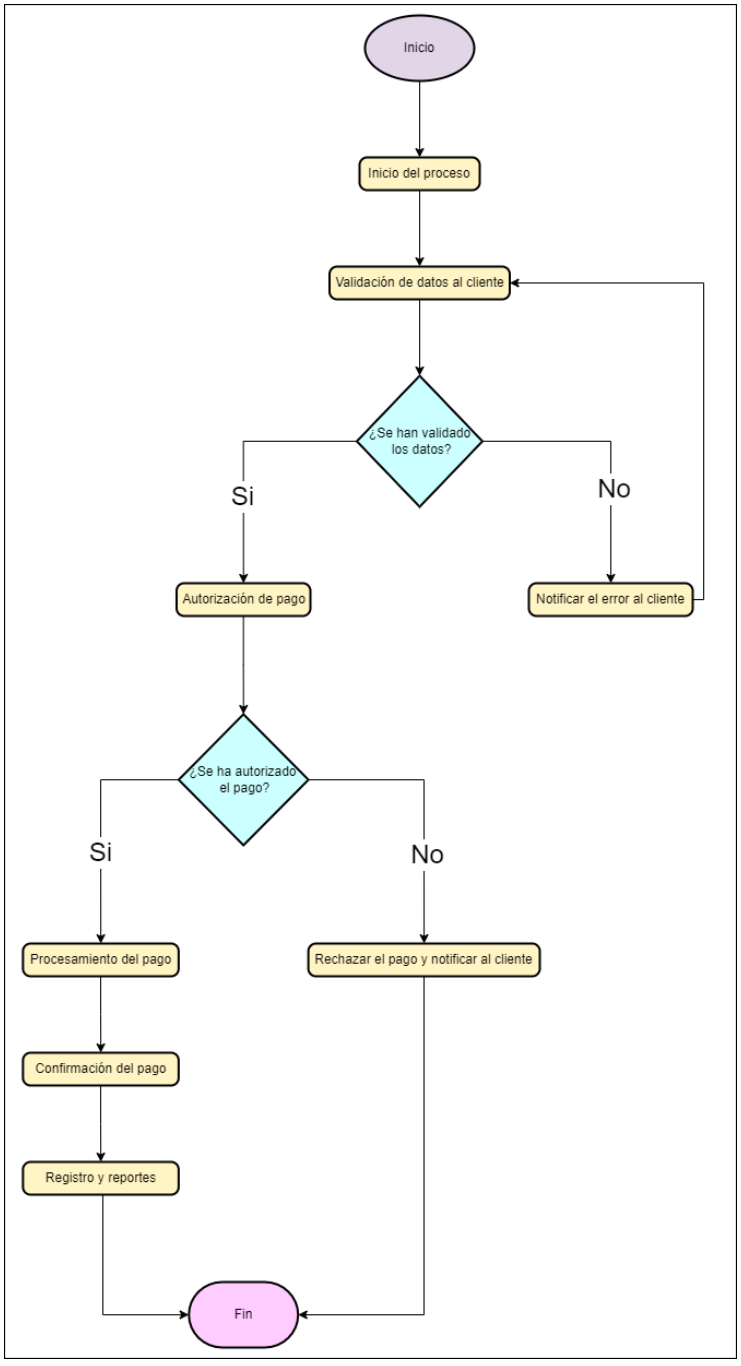
4.1. DIAGRAMA DE ARQUITECTURA GENERAL



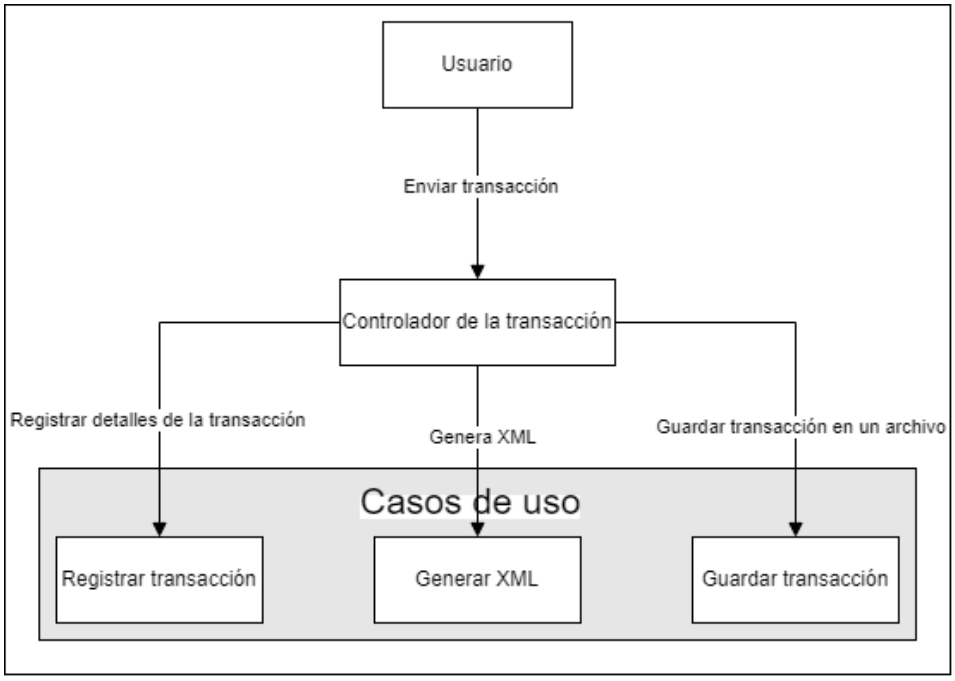
4.2. DIAGRAMA DE DESPLIEGE



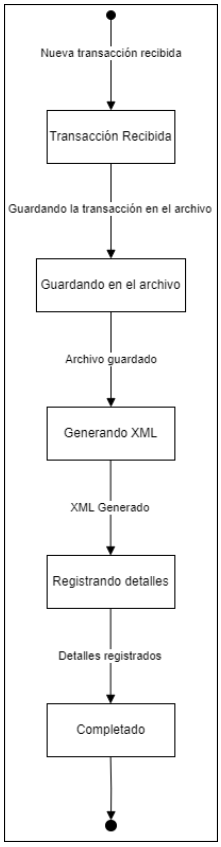
4.3. DIAGRAMA DE FLUJO DE PROCESOS (BPMN):



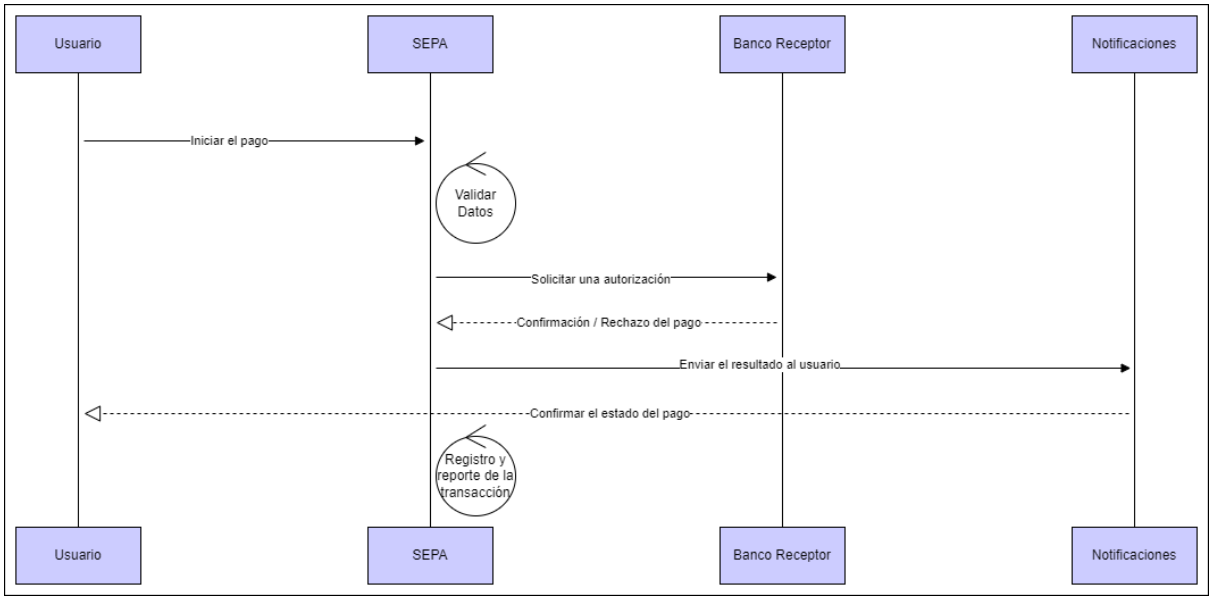
4.4. DIAGRAMA DE CASOS DE USO:



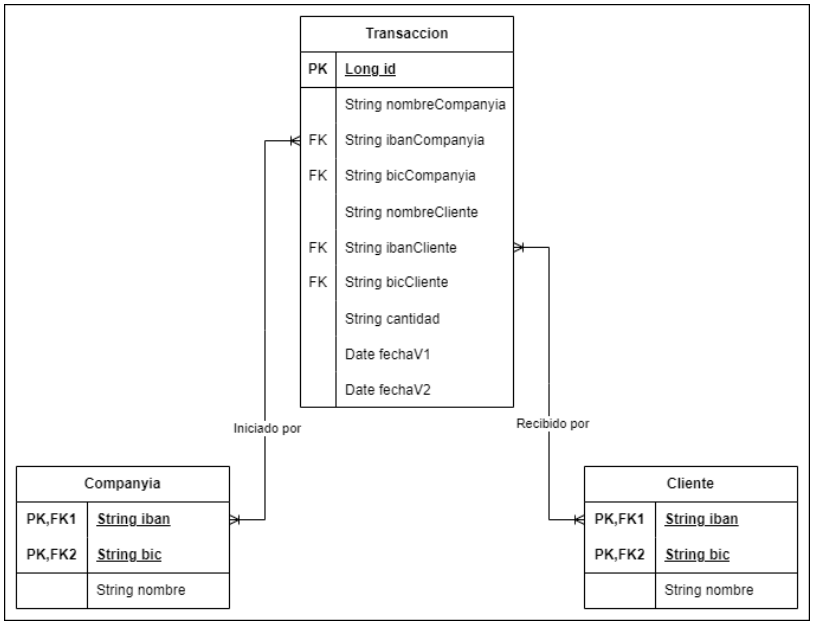
4.5. DIAGRAMA DE ESTADO:



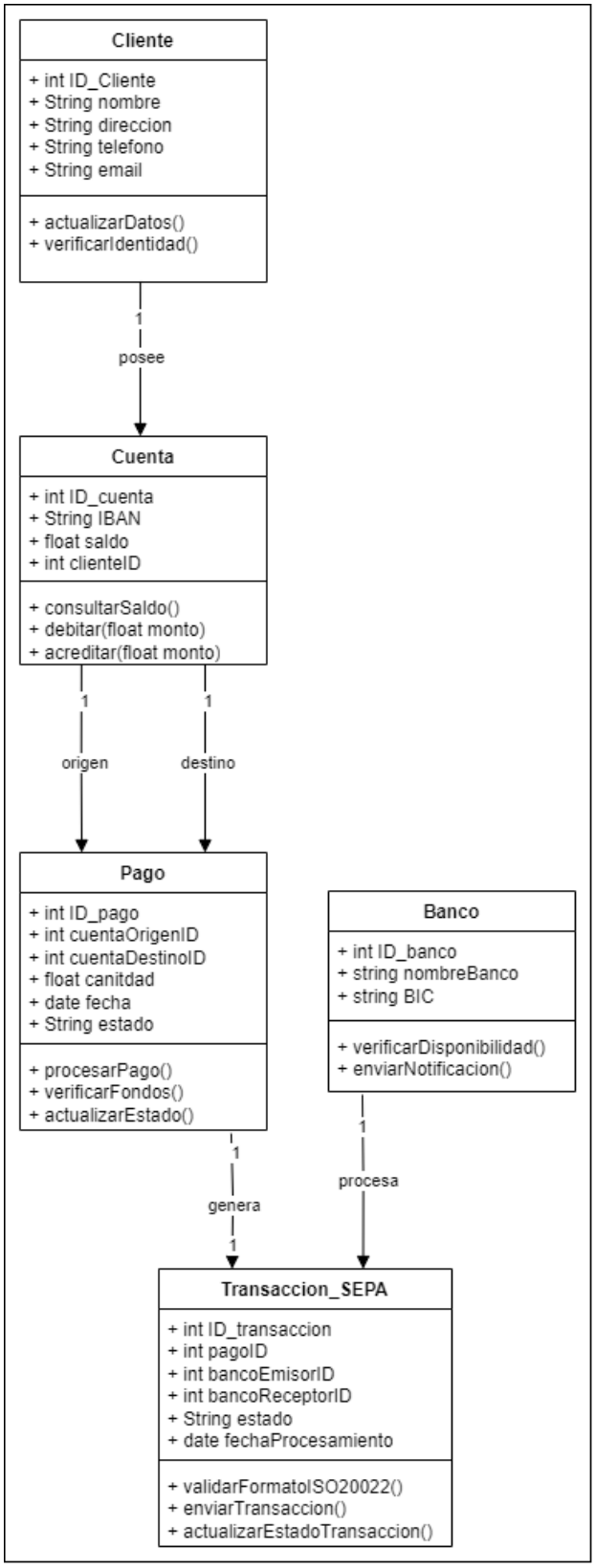
4.6. DIAGRAMA DE SECUENCIA:



4.7. DIAGRAMA DE ENTIDAD – RELACIÓN (E-R):



4.8. DIAGRAMA DE CLASES:



5. GUÍA DE INSTALACIÓN Y CONFIGURACIÓN

5.1. REQUISITOS DEL SISTEMA:

Para configurar el entorno de desarrollo y producción, le recomendamos que su sistema cumpla con los siguientes requisitos mínimos:

- **Sistema operativo**: Windows 10/11, macOS 10.15 o versiones superiores, o distribuciones de Linux (Ubuntu 20.04+).
- **Procesador**: Intel i5 o AMD Ryzen 5 o procesadores superiores.
- **Memoria RAM**: 8 GB mínimo (16 GB recomendado).
- **Espacio en disco**: 50 GB de espacio disponible.

5.1.1. SOFTWARE NECESARIO:

- **Java Development Kit (JDK)**: Versión 17 o superior.
- **IDE recomendado**: Visual Studio Code.
- **Herramienta de control de versiones**: Git.
- **Framework**: Spring Boot.
- **Dependencias adicionales**: Node.js y npm.

5.2. INSTALACIÓN Y CONFIGURACIÓN DEL ENTORNO DE DESARROLLO:

5.2.1. INSTALACIÓN DEL JDK:

1. Descargue la última versión del JDK de Oracle: https://www.oracle.com/java/technologies/javase-jdk17-downloads.html
2. Instale el JDK siguiendo las instrucciones de su sistema operativo.
3. Configure la variable de entorno ***Java_Home*** apuntando a la carpeta de instalación del JDK.

5.2.2. INSTALACIÓN DEL IDE:

1. Descargue e instale el ID que usted prefiera: https://code.visualstudio.com/
2. Configure el IDE para que utilice el JDK instalado.

5.2.3. CONFIGURACIÓN DEL CONTROL DE VERSIONES:

1. Descargue el Git y sigua las instrucciones: https://git-scm.com/
2. Configure su nombre de usuario y correo electrónico de Git ejecutando estos dos comandos:
   git config --global user.name “Nombre a elegir”
   git config --global user.email “su_nombre_de_correo@su_dominio_del_correo.com”

5.3. DESPLIEGUE INICIAL:

1. Clone el repositorio de su aplicación en el servidor de producción.
2. Compile y ejecute la aplicación, asegurándose de que las variables de entornos estén correctamente configuradas.
3. Verifique que todos los servidores funcionen según lo esperado y realice las pruebas iniciales.

6. GUÍA DE DESPLIEGUE

6.1. PROCEDIMIENTOS PARA DESPLEGAR EL SISTEMA EN ENTORNOS DE PRODUCCIÓN:

6.1.1. PREPARACIÓN DEL ENTORNO DE PRODUCCIÓN:

Antes de comenzar con el despliegue, asegúrese de que el entorno de producción esté listo y cumpla con los siguientes requisitos:

1. **Sistema Operativo**: Asegúrese de que el servidor de producción esté ejecutando una versión estable y actualizada del sistema operativo.
2. **Red**: Verifique que las conexiones de red sean seguras.
3. **Seguridad**: Confirme que los sistemas de seguridad como firewalls y controles de acceso estén configurados correctamente.

7. GUÍA DE ESTILO DE CÓDIGO

El código del proyecto sigue un conjunto de normas y convenciones de estilo para asegurar la legibilidad, consistencia y facilidad de mantenimiento. Estas reglas se aplican tanto al frontend como al backend, y aparecen nombramiento de variables, organización del código y formato de los archivos.

7.1. NORMAS Y CONVENCIONES PARA EL CÓDIGO:

7.1.1. ESTILO DE CÓDIGO GENERAL:

- Indentación: utilizar espacios por nivel de indentación.
- Longitud de línea: mantener el ancho máximo de línea en 100 caracteres.
- Comentarios: Incluir comentarios claros y concisos para explicar secciones complejas del código.

7.1.2. CONVENCIONES DE NOMENCLATURA:

7.1.2.1. VARIABLES Y MÉTODOS:

- Utilizar **camelCase** para nombres de variables y métodos en Java.
- Utilizar **snake_case** para nombres de variables en scripts o archivos de configuración.

7.1.2.2. CLASES:

- Usar **PascalCase** para nombres de clases y tipos.

7.1.2.3. CONSTANTES:

- Escribir constantes en mayúsculas con guiones bajos.

7.1.3. ORGANIZACIÓN DE ARCHIVOS:

7.1.3.1. ESTRUCTURA DEL PROYECTO:

Mantener una estructura organizada, separando las diferentes capas de la arquitectura en carpetas.

7.1.3.2. IMPORTS:

Colocar las importaciones al inicio del archivo, ordenadas por:

1. Importaciones de bibliotecas estándar.
2. Importaciones de bibliotecas externas.
3. Importaciones de módulos internos.

7.1.3.3. ESPACIOS EN BLANCO:

Usar espacios en blanco para mejorar la legibilidad, especialmente alrededor de operadores y después de comas.

7.1.4. ESPECÍFICOS DE BACKEND:

7.1.4.1. MANEJO DE EXCEPCIONES:

Utilizamos bloques *try – catch* para capturar y manejar las excepciones y asegurarse de no dejar excepciones sin manejar.

7.1.4.2. FORMATO XML:

Asegurarse de que los ficheros XML sigan el esquema de validación correspondiente.

7.1.4.3. PRUEBAS UNITARIAS:

Mantener un alto porcentaje de cobertura de pruebas al utilizar frameworks como **JUnit**.

7.1.5. ESPECÍFICOS DE FRONTED:

7.1.5.1. FORMATO XML:

Asegurarse de que los archivos de diseño XML estén correctamente formateados y validados. Utilizar atributos significativos y evitar duplicación de estilos.

7.1.5.2. IDS DE RECURSOS:

Usar nombres descriptivos y específicos para los recursos de la interfaz.

7.1.5.3. STRINGS Y DIMENSIONES:

Centralizar las cadenas de texto y las dimensiones en los archivos ***strings.xml*** y ***dimens.xml*** respectivamente, en lugar de escribirlos directamente en el diseño XML.

7.1.6. PRÁCTICAS ADICIONALES:

7.1.6.1. REVISIONES DE CÓDIGO:

El código debe estar revisado mediante el sistema de control de versiones **GitHub**.

7.1.6.2. OPTIMIZACIÓN:

Realizar comprobaciones periódicas para optimizar el código y reducir la dificultad.

7.1.6.3. DOCUMENTACIÓN:

Tenemos que mantener la documentación del código actualizada, sobre todo cuando se hacen cambios significativos en el proyecto.

8. ESPECIFICACIONES FUNCIONALES

8.1. DESCRIPCIÓN DE LAS FUNCIONALIDADES DEL SISTEMA:

El sistema de gestión de Débito Directo SEPA implementa una serie de funcionalidades esenciales para facilitar y automatizar el proceso de transacciones electrónicas, cumpliendo con las normativas SEPA.

Esta serie de funcionalidades esenciales incluyen:

8.1.1. GESTIÓN DE PAGOS:

La gestión de pagos permite la creación, modificación y cancelación de órdenes de débitos directo. También, los usuarios puedes especificar destalles como:

- El importe.
- La cuenta del deudor.
- La cuenta del acreedor.
- Las fechas relevantes.

8.1.2. GENERACIÓN DE FICHEROS XML SEPA:

El sistema genera automáticamente archivos XML en el formato **pain.008.001.02**, que contiene todos los detalles necesarios para la ejecución de los pagos, garantizando la compatibilidad con las entidades bancarias.

8.1.3. VALIDACIÓN DE DATOS:

Antes de la generación del fichero XML, el sistema valida que todos los campos obligatorios del formulario estén completos y cumplan con los formatos requeridos.

8.1.4. INTERFAZ DE USUARIO INTUITIVA:

La interfaz gráfica facilita la introducción de datos de manera simple, incluyendo secciones para ver, editar y gestionar las instrucciones de pago.

8.1.5. CONTROL DE MANDATOS:

El control de mandatos sirve para registrar y gestionar la información de los mandatos de débito, conteniendo el identificador del mandato y la fecha de la firma.

8.1.6. RESUMEN Y CONTROL DE TRANSACCIONES:

El control de transacciones permite a los usuarios visualizar un resumen de las transacciones realizadas y programadas, junto con el número total de las transacciones y la suma total del importe a pagar.

8.1.7. NOTIFICACIONES Y CONFIRMACIONES:

Sirve para el envío de notificaciones sobre el estado de las transacciones y confirmaciones de pago.

8.2. REQUERIMIENTOS FUNCIONALES Y NO FUNCIONALES

8.2.1. REQUERIMIENTOS FUNCIONALES:

8.2.1.1. CREACIÓN DE ORDENES DE DÉBITO DIRECTO:

- En primer lugar, el sistema tiene que permitir a los usuarios crear una nueva orden de débito directo, especificando los detalles necesarios, como: IBAN, importe, acreedor…
- En segundo lugar, el sistema debe validar la información ingresada y mostrar mensajes de error si algún campo es incorrecto o este incompleto.

8.2.1.2. GENERACIÓN AUTOMÁTICA DE ARCHIVOS XML:

- El sistema debe generar archivos XML respetando la normativa **pain.008.001.02** con los detalles de las transacciones.
- Los archivos deben estar listos para ser enviados a las diferentes entidades bancarias permitentes.

8.2.1.3. GESTIÓN DE MANDATOS DE DÉBITO:

- Los usuarios deben poder registrar y gestionar los mandatos asociados a los pagos, incluyendo la fecha de firma y el identificador del mandato.

8.2.1.4. VISUALIZACIÓN Y EDICIÓN DE TRANSACCIONES:

- El sistema debe permitir la visualización de todas las transacciones registradas, junto con la opción de editar o eliminar las órdenes antes de que se procesen.

8.2.1.5. RESUMEN DE TRANSACCIONES:

- El sistema proporciona un resumen que muestra el número de transacciones y la suma total del importe a cargar.

8.2.1.6. NOTIFICACIONES DE ESTADO:

- Se deben enviar notificaciones a los usuarios, que están utilizando el sistema, indicando el estado de las transacciones:
    - Pendiente.
    - Completada.
    - Fallida.

8.2.2. REQUERIMIENTOS **NO** FUNCIONALES:

8.2.2.1. SEGURIDAD:

- El sistema debe proteger la información sensible de los usuarios a través de una encriptación y protocolos seguros.
- Debe a ver autentificación de usuarios y control de acceso para proteger los datos bancarios.

8.2.2.2. RENDIMIENTO:

- El sistema debe estar capacitado para manejar múltiples transacciones a la vez sin afectar al rendimiento.
- La generación XML tiene que crearse en menos de 2 segundos para transacciones de un tamaño mediano.

8.2.2.3. AMPLIABILIDAD:

- El sistema debe ser ampliable para soportar un aumento del número de usuarios y transacciones en un futuro.

8.2.2.4. DISPONIBILIDAD:

- El proyecto garantiza una disponibilidad del 99% y proporciona mecanismos de recuperación ante fallos.

8.2.2.5. USABILIDAD:

- La interfaz es intuitiva y sencilla de utilizar, minimizando el aprendizaje para los usuarios finales.

8.2.2.6. COMPATIBILIDAD:

- El sistema es compatible con los principales navegadores web de distintos dispositivos.
- Cumple con los estándares de SEPA para asegurar la compatibilidad con las entidades bancarias.

9. PRUEBAS Y COBERTURA DEL CÓDIGO:

9.1. ESTRATEGIA DE PRUEBAS Y TIPOS DE PRUEBAS A REALIZAR

9.1.1. ESTRATEGIA DE PRUEBAS:

La estrategia de pruebas es el enfoque general que guiará la ejecución de las pruebas en el sistema, asegurando que se cumplan todos los requisitos de calidad, seguridad y rendimiento. La estrategia de pruebas tiene los siguientes puntos:

1. **Cobertura de Requisitos**, donde todas las funcionalidades del sistema sean probadas. Esto incluye pruebas funcionales, de integración y de rendimiento.
2. **Automatización de Pruebas**, donde se automatizarán las pruebas de regresión para agilizar el proceso de pruebas y reducir los posibles errores humanos.
3. **Entorno de Pruebas**, donde las pruebas se ejecutarán en un entorno similar al entorno de producción.
4. **Pruebas de Seguridad y Rendimiento**, donde a las pruebas de seguridad, se le intentará detectar vulnerabilidades, así como a las pruebas de rendimiento para garantizar que el sistema tenga un buen rendimiento.
5. **Ciclo de Vida de las Pruebas**:
    - **Pruebas de Desarrollo**: Para detectar errores temprano.
    - **Pruebas de Integración**: Para asegurar que los diferentes módulos y servicios funcionen correctamente juntos.
    - **Pruebas de Aceptación**: Para validar que el sistema cumple con los requisitos del cliente.
    - **Pruebas de Regresión**: Para asegurarse de que los cambios en el código no generen errores.

9.1.2. TIPOS DE PRUEBA A REALIZAR:

Las diferentes pruebas a realizar son las siguientes:

9.1.2.1. PRUEBAS UNITARIAS:

Pruebas que verifican las funciones del código en aislamiento, es decir, cada componente del sistema es probado de manera independiente para garantizar su correcto funcionamiento con seguridad.

9.1.2.2. PRUEBAS DE INTEGRACIÓN:

Pruebas que verifican que las diferentes partes del sistema funcionen correctamente entre sí. Se realizan después de las pruebas unitarias.

9.1.2.3. PRUEBAS DE ACEPTACIÓN:

Pruebas realizadas para verificar que el sistema cumple con los requisitos funcionales.

9.1.2.4. PRUEBAS DE REGRESIÓN:

Pruebas que garantizan que el código no afecte al funcionamiento de las características ya existentes. Estas pruebas deben ejecutarse en cada ciclo de desarrollo, especialmente después de cambios significativos en el código.

9.1.2.5. PRUEBAS DE SEGURIDAD:

Pruebas realizadas para identificar posibles vulnerabilidades.

9.1.2.6. PRUEBAS DE RENDIMIENTO:

Pruebas que evalúan el comportamiento del sistema bajo condiciones de carga pesada, como un alto número de usuarios simultáneos o grandes volúmenes de datos.

9.1.2.7. PRUEBAS DE USABILIDAD:

Pruebas que evalúan la interfaz de usuario y la experiencia de usuario del sistema, con el fin de asegurar que la aplicación sea fácil de usar.

9.2. HERRAMIENTAS Y MÉTODOS DE COBERTURA DE CÓDIGO:

9.2.1. HERRAMIENTAS:

Las herramientas de cobertura de código son esenciales para medir el alcance de las pruebas y asegurar que se cubren todos los aspectos del código durante las pruebas.

Las diferentes herramientas de cobertura de código utilizadas son:

9.2.1.1. JACOCO (JAVA CODE COVERAGE):

Herramienta que proporciona métricas detalladas sobre qué partes del código se han ejecutado durante las pruebas unitarias.

9.2.1.2. JEST:

Framework de pruebas para JavaScript que incluye soporte integrado para cobertura de código. Se utiliza en aplicaciones React y Node.js.

9.2.2. MÉTODOS:

Los diferentes métodos para la cobertura de código son los siguientes:

9.2.2.1. COBERTURA DE LÍNEA DE CÓDIGO:

Método que mide la cantidad de líneas de código ejecutadas durante las pruebas, proporciona una visión general de la cobertura.

9.2.2.2. COBERTURA DE RAMA:

Método que mide la cantidad de rutas de ejecución tomadas en las estructuras de control (condicionales…).

9.2.2.3. COBERTURA DE CAMINO:

Método que mide la cantidad de rutas únicas (caminos) que el código puede tomar. Las pruebas tienen que cubrir todas las combinaciones posibles de ramas y condiciones.s posibles de ramas y condiciones.

9.2.2.4. COBERTURA DE CONDICIÓN:

Método que verifica que todas las posibles evaluaciones de una condición se prueben durante las pruebas.

9.2.2.5. COBERTURA DE FUNCIÓN / MÉTODO:

Método que mide si cada función o método del código ha sido llamado durante las pruebas.

9.2.2.6. COBERTURA DE DATOS:

Método que asegura que todas las variables se prueben con diferentes valores, detectando casos no cubiertos por las pruebas de funcionalidad.

10. REGISTRO DE CAMBIOS (CHANGELOG).

10.1. HISTORIAL DE CAMBIOS:

10.1.1. MEJORAS REALIZADAS:

10.1.1.1. VERSIÓN 1.0.0 – 04/10/2024:

- **Lanzamiento inicial**: Primera versión del sistema SEPA implementada con todas las funcionalidades básicas.
- **Interfaz de usuario**: Se ha optimizado la interfaz para mejorar la experiencia de usuario y facilitar la navegación entre las diferentes secciones del sistema, por ahora esta interfaz solo es visual, aún no es funcional.

- **Documentación**: Documentación iniciada, incluida para facilitar la implementación y configuración del sistema.

10.1.1.2. VERSIÓN 1.1.0 – 23/10/2024:

- **Mejora de la seguridad**: Implementación de autenticación en el formulario.
- **Automatización de pruebas**: Introducción de pruebas unitarias y de integración.
- **Documentación**: Documentación poco avanzada, explicando aspectos generales.

10.1.1.3. VERSIÓN 1.2.0 – 07/11/24:

- **Conexión**: Conexión correcta entre el front-end y el back-end.
- **Cobertura de código**: Integración de herramientas de cobertura de código (JaCoCo) para evaluar el rendimiento de las pruebas.
- **Documentación**: Documentación avanzada, añadiendo aspectos más técnicos de forma precisa.

10.1.1.4. VERSIÓN 1.3.0. – 21/11/24:

- **Conexión**: Conexión correcta entre el front-end y el back-end.
- **Cobertura de código**: Integración de herramientas de cobertura de código (JaCoCo) para evaluar el rendimiento de las pruebas.
- **Documentación**: Documentación avanzada, añadiendo aspectos más técnicos de forma precisa.

10.2. VERSIONES DEL SOFTWARE:

- **Versión 1.0.0**: Primer lanzamiento del sistema SEPA.
- **Versión 1.1.0**: Mejora de seguridad e introducción de pruebas.
- **Versión 1.2.0**: Conexión realizada y cobertura de código.
- **Versión 1.3.0**: Versión final.

10.3. FECHAS DE LANZAMIENTO:

- **Versión 1.0.0**: 04-10-2024
- **Versión 1.1.0**: 23-10-2024
- **Versión 1.2.0**: 07-11-2024
- **Versión 1.3.0**: 21-11-2024

11. DOCUMENTACIÓN DEL CÓDIGO (JAVADOC)

Javadoc es la herramienta estándar de Java para generar documentación técnica de forma automática, basada en comentarios dentro del código fuente. Utiliza anotaciones específicas para describir clases, métodos y atributos, creando archivos HTML que facilitan la comprensión y el uso de las APIs.

11.1. INSTRUCCIONES SOBRE CÓMO GENERAR Y MANTENER LA DOCUMENTACIÓN DEL CÓDIGO

11.1.1. ESTRUCTURA DE COMENTARIOS:

Cada clase, método o atributo debe ser documentado utilizando comentarios Javadoc. Estos comentarios generan la documentación de manera automática.

11.1.2. GENERAR UN JAVADOC DESDE LA LÍNEA DE COMANDOS:

Para generar dicha documentación se puede usar la herramienta javadoc incluida en el JDK, ejecutando este comando:

javadoc -d doc -sourcepath src/main/java com.sepa.back_end

11.1.3. MANTENIMIENTO DE LA DOCUMENTACIÓN:

- **Actualizar regularmente**: La documentación debe actualizarse cada vez que se añadan nuevas funcionalidades o haya algún cambio en los métodos y clases existentes.
- **Verificar la consistencia**: Los comentarios Javadoc deben ser claros, completos y consistentes con el código.
- **Automatizar el proceso de generación**: Se puede integrar la generación de Javadoc en el proceso de construcción del proyecto, utilizando herramientas como Maven (mvn javadoc:javadoc).

11.2. ESTÁNDARES PARA COMENTARIOS EN EL CÓDIGO:

Cabe destacar que los comentarios en el código deben seguir un conjunto de estándares que faciliten su lectura y comprensión.

11.2.1. REGLAS GENERALES PARA LOS COMENTARIOS:

- **Claridad y concisión**: Comentarios claros, precisos y directos, evitando redundancias y obviedades.

- **Comentarios en cuerpo del código**:
  1. Comentarios dentro del código para explicar bloques de código complejos.
  2. También tenemos los comentarios en línea que deben comenzar con // y colocarse al final de la línea de código. Esto es un ejemplo sencillo de comentario en línea sacado de internet:
     int resultado = a + b // Suma de los dos números

11.2.2. ESTILO DE COMENTARIOS JAVADOC:

- **Comentarios de clase**: Deben comenzar con una descripción general de la clase, seguido con detalles más precisos sobre el propósito de la clase.
- **Comentarios de métodos**: Describen el propósito del método, los parámetros y el valor de retorno.
- **Comentarios de bloques complejos**: A aquellos bloques de códigos que han sido considerados como bloques complejos, incluimos comentarios descriptivos antes del bloque para poder entender, a grandes rasgos, el funcionamiento del bloque.

12. DOCUMENTACIÓN DE LA ARQUITECTURA.

12.1. DESCRIPCIÓN DE LA ARQUITECTURA DEL SISTEMA:

La arquitectura del sistema está basa en un enfoque modular y adaptable, permitiendo la integración de las funcionalidades SEPA de **Débito Directo** dentro de una plataforma bancaria o de pagos. La solución está compuesta por varias capas que interactúan de manera sencilla para garantizar la correcta creación, validación y envío de los archivos XML conforme al estándar ISO 20022.

A continuación, explicaré los siguientes componentes clave de la arquitectura:

12.1.1. CAPA DE PRESENTACIÓN (FRONTEND):

La interfaz gráfica del usuario (GUI) permite al usuario interactuar con el sistema. Esta capa está formada por formularios donde se introducen los datos de la transacción. El sistema permite generar el archivo XML SEPA desde los datos introducidos.

12.1.1.1. ARCHIVOS Y CARPETAS PERTENECIENTES AL FRONTEND:

En este apartado encontramos las dependencias para iniciar Angular, Spring Boot, los archivos del HTML, CSS y JavaScript.

12.1.2.CAPA DE SERVICIO (BACKEND):

Esta capa se encarga de la lógica de negocio y las operaciones necesarias para manejar la información financiera y generar los archivos XML a raíz del estándar SEPA.

Utiliza bibliotecas como **JAXB** para la conversión entre objetos Java y XML, y tu Junit para asegurar la calidad del código mediante pruebas unitarias.

La arquitectura del backend está diseñada con principios de Microservicios, lo que permite escalar componentes individuales según la demanda y facilita la integración con otros sistemas o servicios con entidades bancarias.

12.1.2.1. ARCHIVOS Y CARPETAS PERTENECIENTES AL BACKEND:

Dentro de la carpeta backend, encontramos dos archivos y cuatro carpetas:

- BackEndApplication.java
- package-lock.json
- Carpeta Controllers.
    - TransactionController.java
- Carpeta Cors.
    - WebConfig.java
- Carpeta Entities.
    - Transaction.java
- Carpeta Util.
    - XmlGenerator.java
    - Util.java

12.1.2.1.1. BACKENDAPPLICATION.JAVA

Este archivo es la clase principal, que se encarga de iniciar la aplicación backend basado en Spring Boot.

12.1.2.1.2. PACKAGE-LOCK.JSON

Este fichero asegura que las dependencias del proyecto Node.js se instalen con las mismas versiones en cualquier dispositivo, evitando discrepancias.

12.1.2.1.3. CARPETA CONTROLLERS:

A. TRANSACTIONCONTROLLER.JAVA

Esta clase es un controlador desarrollado con Spring Boot, cuyo propósito principal es procesar las transacciones financieras enviadas por un cliente. Esta clase hace las siguientes funciones:

- Recibe la transacción.
- Guarda los datos en un archivo.
- Genera un archivo XML.
- Muestra información en la consola.
- Devuelve la transacción.

12.1.2.1.4. CARPETA CORS:

A. WEBCONFIG.JAVA

Este archivo configura las reglas de CORS (Cross – Origin Resource Sharing). Asimismo, el fichero permite que la aplicación backend sea accesible desde la aplicación frontend.

12.1.2.1.5. CARPETA ENTITIES:

A. TRANSACTION.JAVA

Este archivo está diseñado para mapear los datos relacionados con transacciones financieras a una tabla de base de datos mediante JPA (Java Persistence API).

12.1.2.1.6. CARPETA UTIL:

A. XMLGENERATOR.JAVA

La clase XmlGenerator sirve para crear archivos XML compatibles con el estándar SEPA. Su propósito principal es automatizar la generación de estos archivos basándose en los datos de una transacción.

El funcionamiento principal de esta clase es:

- Recibir una transacción como entrada: nombres, IBAN, BIC, montos y fechas.
- Procesar y formatear los datos para ajustarlos al estándar SEPA
- Crear un archivo XML con la información de la transacción.

B. UTIL.JAVA

Su propósito es facilitar la creación de ciertos archivos necesarios para el proyecto.

12.1.3.CAPA DE PERSISTENCIA:

El sistema almacena la información relaciona con las transacciones y datos de clientes de forma segura en bases de datos. Se pueden usar bases de datos SQL o NoSQL. También, se implementan servicios de almacenamiento que gestionan los datos de las transacciones SEPA y la historia de las operaciones realizadas.

13.1.4. CAPA DE INTEGRACIÓN:

La capa de integración se encarga de la comunicación con los sistemas externos para enviar los archivos XML generados y recibir los estados de las transacciones. También, implementa el protocolo estándar SEPA para asegurar que las transacciones se envíen correctamente.

12.2. DECISIONES DE DISEÑO TOMADAS:

Las decisiones de diseño tomadas se eligieron con el fin de garantizar la adaptabilidad, seguridad y facilidad de mantenimiento del sistema.

12.2.1.PRINCIPALES DECISIONES.

12.2.1.1. ELECCIÓN DEL ESTÁNDAR SEPA Y XML:

La implementación del estándar **ISO 20022** con el formato **pain.008.001.02** fue una decisión tomada debido a su implementación a gran escala por parte de las entidades bancarias de la zona europea. Esto asegura la compatibilidad con los sistemas de pago SEPA.

12.2.1.2. USO DE MICROSERVICIOS:

La arquitectura de microservicios fue adoptada para dividir el sistema en componentes modulares e independientes, lo que facilita adaptabilidad, la reutilización de servicios y una mejor gestión de los errores y las actualizaciones de componentes.

12.2.1.3. VALIDACIÓN DE ARCHIVOS XML:

La validación de los archivos XML SEPA se realiza mediante herramientas automáticas y reglas específicas definidas para el formato **pain.008.001.02**. Esto garantiza que los archivos XML generados sean válidos antes de su envío a las entidades bancarias.

12.2.1.4. SEGURIDAD EN EL MANEJO DE DATOS:

Hemos implementado prácticas de seguridad robustas, como la encriptación de datos sensibles (IBAN o BIC) y el uso de SST/TLS para proteger la transmisión de datos hacia los bancos o otras entidades financieras.

12.2.1.5. COMPATIBILIDAD CON OTRAS PLATAFORMAS:

El sistema se diseñó para permitir la integración con otras plataformas de pagos y servicios bancarios, lo que facilita la compatibilidad en un moderno mundo bancario.

12.3. DIAGRAMAS RELEVANTES:

12.3.1. DIAGRAMA DE ARQUITECTURA GENERAL:

Este diagrama ilustra las distintas capas del sistema y como interactúan entre ellas.

12.3.1.1. COMPONENTES CLAVE:

- Frontend.
- Backend.
- Base de datos.
- Servicios de integración externa (banco).

12.3.1.2. FLUJO DE DATOS:

El flujo de datos funciona de la siguiente manera: la información es introducida en el frontend, procesada en el backend, validada y almacenada en la base de datos, y finalmente enviada al banco a través del servicio de integración.

12.3.2. DIAGRAMA DE FLUJO DE PROCESOS (BPMN):

Este diagrama describe cómo se lleva a cabo el proceso de creación de una transacción SEPA desde la recepción de datos hasta la validación y envió de los archivos XML.

12.3.3. DIAGRAMA DE DESPLIEGUE:

Este diagrama muestra cómo se distribuyen los componentes del sistema en los servidores y máquinas, incluyendo las interacciones entre los diferentes servicios, bases de datos y sistemas externos.

12.3.4. DIAGRAMA DE SECUENCIA:

Este diagrama representa cómo funciona la comunicación secuencial entre los diferentes componentes del sistema cuando un usuario genera una transacción SEPA. Por añadir, detalla los pasos involucrados en la creación, validación y envió del archivo XML.

13. DOCUMENTACIÓN DE DEPENDECIAS

13.1. LISTADO DE BIBLIOTECAS UTILIZADAS:

13.1.1. JAVA XML LIBRARIES:

En este proyecto utilizamos la biblioteca estándar: ***javax.xml***. Esta biblioteca sirve para la manipulación y validación de archivos XML. Asimismo, es necesaria para trabajar con los formatos SEPA y garantizar que el archivo XML generado cumpla con el estándar **pain.008.001.02**.

13.1.2. JAXB:

JAXB (Java Architecture for XML Binding) sirve para convertir objetos Java en XML. Es útil para la creación de objetos de la estructura SEPA y la conversión a formato XML.

13.1.3. SEPA XML VALIDATION TOOLS:

Esta herramienta se utiliza para validar los ficheros XML generados sigan el esquema de **paing.008.001.02**. Estas validaciones aseguran que el contenido del XML esté bien formado y sea compatible con los requisitos del sistema bancario SEPA.

13.1.4. JUNIT:

Junit sirve para las pruebas de validación y se usa para realizar pruebas unitarias y funcionales del código, asegurando que las funcionalidades de validación y generación de archivos XML SEPA trabajen de manera precisa.

13.1.5. GITHUB ACTIONS:

GitHub Actions se utiliza para crear flujos de trabajo que validen la creación de los ficheros XML y las pruebas de integración en cada *push* o *pull request*.

13.1.6. MAVEN:

Maven sirve como herramienta de gestión y creación de dependencias para facilitar la integración de las bibliotecas necesarias y el despliegue del proyecto.

13.2. DEPENDENCIAS UTILIZADAS:

- Archivo de configuración del **javax.xml**.
- Archivo **pom.xml** de Maven.
- Configuración del archivo .**github/workflows/ci.yml**.
- **Node**: Entorno de ejecución que permite ejecutar código JavaScript. Sirve para procesar transacciones.
- **Angular**: Framework para el desarrollo de aplicaciones web. Sirve para crear una interfaz de usuario interactiva que permita a los usuarios enviar y gestionar pagos SEPA.
- **Spring Boot**: Spring Boot es un framework para el desarrollo de aplicaciones backend en Java. Sirve para conectar el front-end con el back-end.

14. MANUAL DEL USUARIO

14.1. INTRODUCCIÓN:

La aplicación Proyecto – SEPA está diseñada para convertir datos adquiridos desde una página web externa a un XML generado desde una aplicación Java.

Se recomienda encarecidamente la instalación de todas las dependías y el seguimiento paso por paso para el correcto funcionamiento de la aplicación.

14.2. REQUISITOS DEL SISTEMA:

14.2.1. SISTEMA OPERATIVO:

Windows: Windows 10 o superior.

MacOS: MacOS 10.15 o superior.

Linux.

14.2.2. ESPACIO EN DISCO:

El espacio necesario en disco es de **301 MB**.

14.2.3. JAVA RUNTIME ENVIRONMENT (JRE):

La versión de Java Runtime Environment (JRE) es la **versión 17 o superior**.

14.3. INSTALACIÓN:

1. Descargue el proyecto desde el link de GitHub: https://github.com/eramis152/proyecto-SEPA
2. Instale las dependencias de Node: archivo ***z_node_instaltion***.

- Simplemente pulse *siguiente* en todos los pasos del instalador.

3. Instale las dependencias de SpringBoot: archivo ***z_install_dependences***.
4. Ejecute el Back End: archivo ***z_start_back***.
   - Procure **no cerrar la terminal**, ya que de hacerlo **finalizará el proceso**.
5. Ejecute el Front End: archivo ***z_start_front***.
   - Procure **no cerrar la terminal**, ya que de hacerlo **finalizará el proceso**.

14.4. NAVEGACIÓN Y UTILACIÓN DE LA APLICACIÓN:

La aplicación cuenta con una página principal, la cual pedirá al usuario los diferentes campos a completar:

Nombre de la compañía	Nombre al cual se ingresará la cantidad del importe.
IBAN Empresa	Código de identificación del número de cuenta de la empresa a abonar.
BIC Empresa	Identificador del banco de la empresa a abonar.
Nombre Cliente	Nombre completo del pagador.
IBAN Cliente	Código de identificación del número de cuenta del pagador.
BIC Cliente	Identificador del banco del pagador.
Importe	Cantidad de dinero abonada.

14.5. FUNCIONALIDADES:

14.5.1. CREACIÓN DE DOCUMENTO XML:

La aplicación Java generará un fichero XML con la estructura de un documento SEPA para su posterior validación. La aplicación recogerá el objeto generado por la aplicación web e introducirá los datos adquiridos en el XML generado, el cual guardará en la carpeta XML.

14.6. CONFIGURACIÓN:

La aplicación **no requiere configuración previa**.

14.7. DESISTALACIÓN DE DEPENDENCIAS:

Para la desinstalación de manera definitiva de todas las dependencias instaladas previamente para la correcta ejecución del proyecto seguiremos los siguientes pasos:

14.7.1. NODE JS:

Para desinstalar las dependencias acceda a: **Aplicaciones > Aplicaciones instaladas**.

Dentro de esta página, busque en el navegador **NodeJS** y, posteriormente, desinstálelo. Esto tendría que borrar todas las dependencias instaladas del mismo.

14.7.2. OTRAS DEPENDENCIAS:

Al borrar la carpeta del proyecto, se llevará consigo todas las dependencias instaladas, ya que se instalan de manera local en el proyecto.

15. PLAN DE MANTEMIENTO

El plan de mantenimiento sirve para garantizar que el sistema esté operativo, seguro y actualizado durante todo su ciclo de vida.

15.1. ESTRATEGIA PARA EL MANTENIMIENTO DEL SISTEMA:

Las estrategias de mantenimiento están diseñadas para garantizar que los sistemas continúen funcionando de manera eficiente y sin interrupciones importantes.
Se deben establecer procesos para detectar y corregir errores rápidamente, mejorar el rendimiento y garantizar la estabilidad a largo plazo del sistema.

15.1.1. TIPOS DE MANTENIMIENTO:

15.1.1.1. MANTENIMIENTO CORRECTIVO:

Se enfoca en corregir fallos o errores detectados en el sistema durante su funcionamiento.

15.1.1.2. MANTENIMIENTO PREVENTIVO:

Implica implementar medidas preventivas para solucionar los problemas antes de que ocurran, como actualizar software y hardware, mejorar el rendimiento, optimizar recursos, etc.

15.1.1.3. MANTENIMIENTO ADAPTATIVO:

Se realiza cuando el sistema necesita ser modificado debido a cambios en el entorno externo, como la actualización de bibliotecas o cambios en el sistema operativo que afectan al funcionamiento del sistema.

15.1.1.4. MANTENIMIENTO EVOLUTIVO:

Implica agregar nuevas funciones y mejoras para garantizar que el sistema permanezca alineado con los objetivos del negocio y necesidades del usuario.

15.1.2. ESTRATEGIA DE MONITOREO:

- **Monitoreo continuo del sistema**: Se implementan herramientas de monitoreo para detectar problemas de rendimiento, errores y fallos en tiempo real.
- **Alertas y notificaciones**: Se configuran alertas automáticas para notificar al equipo de mantenimiento sobre incidentes o caídas del sistema.
- **Revisión de logs y auditoría**: Se revisa regularmente los registros de actividad y errores del sistema para identificar tendencias o posibles problemas.

15.2. PLAN DE ACTUALIZACIONES Y GESTIÓN DE INCIDENCIAS

El plan de actualización y gestión de incidencias está diseñado para garantizar que las actualizaciones del sistema se realicen de forma controlada y las incidencias se gestionen de forma eficaz para minimizar el impacto en los usuarios.

15.2.1. PLAN DE ACTUALIZACIONES:

Las actualizaciones del sistema deben planificarse cuidadosamente para evitar interrupciones en el servicio. Estas actualizaciones incluyen nuevas funciones, mejoras de rendimiento y “parches” de seguridad.

A continuación, se presenta el procedimiento para las actualizaciones:

15.2.1.1. IDENTIFICACIÓN DE LAS NECESIDADES DE ACTUALIZACIÓN:

Se deben identificar las dependencias del sistema que requieren actualizaciones periódicas para mantener la compatibilidad y seguridad del sistema.

15.2.1.2. PRUEBAS DE ACTUALIZACIÓN:

- Antes de implementar cualquier actualización en producción, debe probarse en un entorno de prueba.
- En segundo lugar, se debe validar la compatibilidad de la actualización con el resto del sistema.
- Por último, se debe realizar pruebas de regresión para asegurarse de que las actualizaciones no rompan las funcionalidades ya existentes.

15.2.1.3. DESPLIEGUE DE ACTUALIZACIÓN:

- Se debe planificar el tiempo de la actualización para que cause el menor impacto posible en los usuarios.
- Se debe implementar un proceso automatizado de despliegue para asegurar consistencia y confiabilidad.
- Importante realizar copias de seguridad del sistema antes de proceder con la actualización.

- Si es posible, realizar la actualización en fases para monitorear cualquier problema que pueda surgir.

15.2.1.4. DOCUMENTACIÓN DE ACTUALIZACIÓN:

- Cada actualización debe estar documentada detalladamente, indicándolos cambios realizados, desde los errores corregidos hasta las mejoras realizadas.
- Actualizar el registro de cambios (Changelog) para reflejar la nueva versión y los detalles pertinentes.

15.2.1.5. REVISIÓN POST – ACTUALIZACIÓN:

- Después de cada actualización, es importante realizar una revisión para detectar posibles fallos o comportamientos inesperados.

15.2.2. GESTIÓN DE INCIDENCIAS:

La gestión de incidencias es un proceso básico que garantiza que cualquier error o fallo que se produzca en el sistema se resuelva de forma rápida y eficaz.

A continuación, se presenta el procedimiento para la gestión de incidencias:

15.2.2.1. REGISTRO DE INCIDENCIAS:

Se crea un sistema centralizado donde se registran todas las incidencias que ocurren en el sistema. Cabe destacar que, este sistema, debe permitir registrar los detalles de la incidencia (descripción, fecha y hora…)

15.2.2.2. CLASIFICACIÓN DE INCIDENCIAS:

- **Incidencias críticas**: Aquellas que afectan gravemente al funcionamiento del sistema, como caídas de servidores o problemas de seguridad. Deben ser atendidas inmediatamente.
- **Incidencias de alta prioridad**: Problemas que afectan a los usuarios, pero no impiden el funcionamiento del sistema, como errores en la interfaz de usuario o en la funcionalidad de un módulo específico, también conocidos como bugs.
- **Incidencias de baja prioridad**: Errores menores que no afectan la funcionalidad principal del sistema y pueden resolverse a mediano plazo.

15.2.2.3. ASIGNACIÓN DE RESPONSABILIDADES:

- Se debe asignar cada incidencia a un miembro del equipo con la habilidad adecuada para resolverla.

15.2.2.4. RESOLUCIÓN Y SEGUIMIENTO:

- La resolución de las incidencias debe ser lo más rápido posible, manteniendo al equipo y a los usuarios informados sobre el progreso.
- Debe haber un seguimiento continuo hasta que la incidencia se haya resuelto completamente.

15.2.2.5. ANÁLISIS DE CAUSA RAÍZ:

- Después de resolver una incidencia, hay que identificar la raíz del problema y evitar que se repita.

15.2.2.6. CIERRE DE INCIDENCIA:

- Una vez que la incidencia se ha resuelto, se detalla sobre cómo se ha solucionado el problema.
- Verificaremos que el sistema esté funcionando correctamente y sin consecuencias de la incidencia antes de cerrar la incidencia.

15.2.2.7. COMUNICACIÓN CON LOS USUARIOS:

- Se les notifica a los usuarios sobre la resolución de las incidencias importantes y proporcionar información relevante sobre los cambios realizados.

15.3. MONITOREO Y REPORTES DE INCIDENCIAS

Es importante que los equipos de mantenimiento supervisen activamente el rendimiento del sistema y se aseguren de que cualquier incidente se detecte con prontitud. La clave es monitorear el sistema en tiempo real y establecer alertas para detectar fallas del sistema.

La estrategia de mantenimiento debe ser ágil y flexible para adaptarse a las necesidades cambiantes del proyecto y los usuarios, asegurando un ciclo de vida del sistema largo y estable.

15.3.1. HERRAMIENTAS DE MONITOREO RECOMENDADAS:

- **Nagios**: Para monitoreo de servidores y aplicaciones.
- **New Relic**: Para monitoreo de rendimiento de aplicaciones y bases de datos.
- **Prometheus + Grafana**: Para monitoreo de métricas y generación de dashboards.

16. LICENCIAS

16.1. OBJETIVO:

El objetivo de SEPA es mejorar y facilitar los procesos de pagos en euros entre distintos países fuera de la frontera y que sean igual de sencillos que los pagos nacionales. Esto se logra normalizando las transferencias y domiciliaciones en euros.

16.2. NORMATIVA Y REGULACIÓN:

La regulación de SEPA está dirigida por la normativa de la UE (Unión Europea) y se basa en normas como el Reglamento (UE) nº 260/2012, es establece los siguientes requisitos para los pagos en euros:

- Identificación de cuentas.
- Formatos de mensajes.
- Fecha límite de implementación.
- Compatibilidad.
- Accesibilidad.

16.2.1. REQUISITOS GENERALES:

- Dentro del Espacio Económico Europeo, se aplica a transferencias y adeudos domiciliados en euros.
- Establece disposiciones comunes para la ejecución de estas operaciones de pago.

16.2.2. IDENTIFICACIÓN DE CUENTAS:

- Uso obligatorio del IBAN (International Bank Account Number) como identificador de cuenta, incluso para operaciones domésticas.

16.2.3. FORMATOS DE MENSAJES:

- Utilización obligatoria del formato XML basado en estándares UNIFI ISO 20022 para no consumidores.

16.2.4. FECHA LÍMITE DE IMPLEMENTACIÓN:

- 1 de febrero de 2014: fecha límite general para reemplazar los instrumentos de pago nacionales por los instrumentos SEPA.
- 1 de febrero de 2016: fecha límite extendida para ciertos productos nicho y situaciones específicas en España.

16.2.5. COMPATIBILIDAD:

- Los sistemas de pago deben ser técnicamente compatibles con otros sistemas de pago minoristas en la UE.

16.2.6. ACCESIBILIDAD:

- Las entidades financieras que ofrecen servicios de transferencias o adeudos nacionales deben procesar también transferencias SEPA y adeudos directos del esquema básico SEPA.

16.3. TIPOS DE PAGOS EN SEPA:

Los diferentes tipos de pago en SEPA son:

- Transferencias SEPA.
- Domiciliaciones SEPA.
- Pagos con tarjeta.

16.3.1. TRANSFERENCIAS EN SEPA:

Hacemos referencia a transferencias de fondos entre cuentas bancarias dentro de la zona SEPA.

16.3.2. DOMICIALICIONES SEPA:

Las domiciliaciones SEPA permites a un acreedor retirar dinero de la cuenta del deudor, siempre con su autorización.

16.3.3. PAGOS CON TARJETA:

Hacemos referencia a los pagos mediante tarjetas bancarias dentro de la zona SEPA.

16.4. DERECHOS DE USO:

Los derechos de uso en SEPA no son derechos de licencia en sentido tradicional, sino que SEPA se basa en la compatibilidad y el acceso a las infraestructuras de pago:

1. Acceso a servicios.
2. Protección de datos.
3. Costes.

16.4.1. ACCESO A SERVICIOS:

Las entidades que pretendan participar en SEPA, ya sean bancos y proveedores de servicios de pago, deben cumplir los requisitos establecidos por las normas SEPA. Por ejemplo, esto incluye cumplir con estándares técnicos y de seguridad.

16.4.2. PROTECCIÓN DE DATOS:

Las transacciones realizadas bajo SEPA están protegidas bajo las normativas de protección de datos (Reglamento General de Protección de Datos en la Unión Europea). Esto significa, que las entidades deben manejar la información personal de manera responsable y con el consentimiento del usuario.

16.4.3. COSTES:

Los usuarios de servicios SEPA (particulares y/o empresas) pueden estar sujetos a tarifas de sus bancos por el uso y disfrute de estos servicios, aunque no debería haber grandes diferencias en los costes entre pagos naciones y pagos internacionales dentro de la zona SEPA.

16.5. RECURSOS ADICIONALES.

16.5.1. GUÍAS OFICIALES:

La EPC (European Payments Council) publica regularmente actualizaciones y guías detalladas sobre SEPA.

16.5.1.1. DATO INTERESANTE:

En la pagina web de EPC aparece documentación técnica específica:

- https://www.europeanpaymentscouncil.eu/

16.5.2. REGULACIONES DEL BCE:

El Banco Central Europeo (BCE) publica normativas relaciones con los pagos europeos.

17. BIBLIOGRAFÍA / FUENTES:

https://www.sepaesp.es/sepa/es/faqs/reglamento/

https://getquipu.com/blog/normativa-sepa/

https://www.bde.es/wbe/es/areas-actuacion/sistemas-pago/los-sistemas-pago-espana/zona-unica-pagos-euros-sepa/

https://www.boe.es/buscar/doc.php?id=DOUE-L-2012-80471

https://eur-lex.europa.eu/ES/legal-content/summary/single-euro-payments-area-regulation.html

https://www.uria.com/documentos/publicaciones/3488/documento/foro06.pdf?id=4272

https://www.europeanpaymentscouncil.eu/