# **Capítulo 2: Solución propuesta**

## **2.1- Introducción**

Para comprender el contexto en el que se desarrolla un proceso específico, es fundamental identificar las entidades gestionadas dentro del negocio, los actores que las manipulan, sus responsabilidades y qué información resulta relevante. En este capítulo, se presenta una descripción del funcionamiento actual del sistema, el modelo de dominio correspondiente, así como el diagrama de casos de uso que representa las interacciones principales. Además, se define la descripción detallada de los casos de uso más relevantes, con énfasis en la funcionalidad del sistema y sus requisitos funcionales y no funcionales. También se ahondará en la seguridad del sistema y como varios mecanismos en influyen de manera que la solución propuesta pueda enfrentarse a varias de las ciber amenazas que abundan en el mercado actual.

## **2.2- Descripción del negocio**

El sistema en desarrollo es un sistema de gestión geográfico que pretende mejorar la manera en que se analizan, procesar y visualizan datos geoespaciales, permitiendo al usuario final extraer de manera más sencilla los datos almacenados en la aplicación. El sistema está orientado para facilitar la migración de datos almacenados en fuentes externas: archivos Excel, con el objetivo de proporcionar una plataforma visual interactiva que permita a los usuarios analizar los datos y visualizar reportes.

La solución permite a los usuarios cargar, procesar y gestionar grandes volúmenes de datos geográficos, optimizando así la toma de decisiones basadas en la ubicación. Esto se logra a través de una interfaz web intuitiva, accesible desde cualquier dispositivo, que garantiza una experiencia fluida y segura.

El negocio comienza cuando el usuario, se autentifica en el sistema para comprobar su identidad. Si el proceso anteriormente explicado es satisfactorio, el usuario puede importar desde su almacenamiento local, un fichero Excel (extensión xls o xlsx) para que sistema extraiga toda la información almacenada y la inserte en un base de datos. Desde este punto, cualquier usuario de la aplicación puede analizar y visualizar los datos en un mapa de coordenada cartesianas.

## **2.3- Descripción del sistema**

El sistema está diseñado para extraer datos almacenados en archivos Excel, analizar su contenido, y filtrar aquellos que cumplan con los requisitos establecidos. Los datos que pasen las validaciones serán ingresados automáticamente en la base de datos del sistema. Además, el sistema se integra con un servicio externo para procesar y extraer información geográfica de las direcciones almacenadas, permitiendo enriquecer los datos con coordenadas y otros atributos geoespaciales.

Una vez procesados, el sistema permite la visualización de estos datos geográficos en un mapa interactivo, proporcionando una representación clara y dinámica de la información. También ofrece la funcionalidad de generar e imprimir reportes sobre diversos aspectos de los datos almacenados, facilitando así la toma de decisiones y el análisis de información.

## **2.7- Seguridad**

La seguridad web o seguridad de sitios web es la práctica de salvaguardar las redes, las comunicaciones en línea, el hardware y el software para que no sean manipulados o utilizados con fines maliciosos. En esta era creciente de ciber amenazas y vulnerabilidades, los principales objetivos son los sitios web[1]

### **2.7.1- Autenticación y Autorización**

En las aplicaciones web, es imprescindible garantizar que solamente los usuarios autorizados y autentificados puedan acceder a las funciones y datos críticos del sistema, evitando así que visitantes no autorizados pueda acceder y hacer un mal uso de dicha información.

La autenticación es el proceso de identificar a los usuarios y garantizar que los mismos sean quienes dicen ser. Esto evita que cualquiera pueda entrar en un determinado sistema o iniciar sesión en alguna plataforma de forma indebida, sin que realmente sea el usuario legítimo que tiene el poder para hacerlo.[2]

La autorización es lo que define a qué recursos de sistema el usuario autenticado podrá acceder. Que haya logrado pasar la instancia de la autenticación, no significa que podrá utilizar el sistema por completo como super administrador. De acuerdo a una serie de reglas, normas y regulaciones propias de cada red interna, se determina que el usuario A tendrá acceso a los recursos X e Y. Sin embargo, el usuario B sólo podrá acceder al recurso Z.[2]

Ejemplos de métodos de autenticación:

* **Sin contraseña o Passwordless.** Este es uno de los métodos modernos más prácticos. Un ejemplo de aplicación es el uso de un enlace mágico (*magic link*). Consiste en que, cada vez que quieras iniciar sesión a un recurso o servicio, se enviará a tu correo electrónico un enlace que te permitirá acceder sin necesidad de contraseña. Este es un método recomendable, ya que se necesita del acceso al correo electrónico y, por ende, hay más garantías de asegurar que es el propio usuario quien está accediendo.[3]
* **Por redes sociales.**Sin duda, ya habrás utilizado este método. Varias aplicaciones y servicios te dan como opción iniciar sesión directamente con alguna **cuenta social**. La ventaja principal es que no hace falta crear una cuenta aparte de forma manual, directamente los datos de esa cuenta social hacen ese paso al iniciar la sesión. Las plataformas sociales más utilizadas son Facebook, Twitter y la cuenta Google. De esta forma podremos iniciar sesión en programas o páginas de forma más rápida, sin tener que registrarnos.[3]
* **Autenticación API.** Este es el proceso de certificar la identidad de un usuario que quiera acceder a recursos y/o servicios en el servidor. Para tener en cuenta.[3]
* **Autenticación Biométrica.**Se vale de las huellas dactilares para validar la identidad del usuario. El caso de uso más popular es en los lugares de trabajo, en donde tanto para registrar la entrada como salida, se posa el dedo para validar la huella dactilar. Esa huella es validada mediante un previo registro de la misma que se almacena en la base de datos. Es cada vez más utilizado este método también en dispositivos móviles, para iniciar sesión, realizar un pago, etc.[3]

### **2.7.2- Spring Security**

La seguridad de la aplicación se gestiona con Spring Security, un marco integral que proporciona un conjunto de herramientas para implementar autenticación, autorización, y protección de acceso a aplicaciones basadas en Java. Spring Security permite configurar políticas de seguridad a nivel de aplicación de manera flexible, adaptándose a diferentes necesidades y escenarios.

1. **Configuración General**: Spring Security intercepta todas las solicitudes HTTP entrantes y aplica reglas de autenticación y autorización. Permite definir qué rutas de la aplicación requieren autenticación, qué tipos de autenticación se utilizan, y qué acciones están permitidas para cada tipo de usuario.
2. **Filtros de Seguridad**: Spring Security proporciona una serie de filtros de seguridad que se ejecutan en cada solicitud para autenticar y autorizar usuarios. Estos filtros pueden ser personalizados para adaptarse a necesidades específicas, como la validación de tokens JWT.

### **2.7.3- Autenticación Sin Estado (Stateless)**

Una de las principales características de una API REST es que el servicio no tiene estado (*stateless*), lo que implica que cada vez que se recurra a él será necesario recordarle nuestros datos, ya sean credenciales de usuario o cualquier otro tipo de información. Lo que por una parte podría parecer una desventaja, al implicar la tediosa tarea de repetir datos, es en realidad una de sus fortalezas: al no memorizarlos, permite una mayor escalabilidad. No harán falta servidores tan potentes, capaces de almacenar todos los estados de sus clientes.[4]

Este factor es especialmente relevante para cualquier API bancaria, como por ejemplo las de BBVA. Si el *fintech* aspira a atraer a todos los clientes de instituciones bancarias y lograr que los desarrolladores introduzcan en sus aplicaciones herramientas financieras, la escalabilidad de sus API es primordial.[4]

### **2.7.4- JSON Web Token**

El token web JSON (JWT) es un estándar abierto (RFC 7519) que define un método compacto y autocontenido para la transmisión segura de información entre partes codificadas como un objeto JSON. Esta información puede verificarse y ser fiable porque está firmada digitalmente. Los JWT pueden firmarse utilizando un secreto (con el algoritmo HMAC) o un par de claves públicas/privadas utilizando RSA.[5]

Estos son algunas situaciones en las que los tokens web JSON son útiles:

* **Autenticación**: Esta es la situación típica para el uso de JWT, una vez que el usuario ha iniciado sesión, cada solicitud posterior incluirá el JWT, permitiendo al usuario acceder a las rutas, servicios y recursos que están permitidos con ese token. El inicio de sesión único es una característica que utiliza ampliamente JWT hoy en día, debido a su pequeña sobrecarga y su capacidad para ser utilizado fácilmente entre los sistemas de diferentes dominios.[5]
* **Intercambio de información**: Los JWT son una buena forma de transmitir información de forma segura entre las partes, porque como se pueden firmar, por ejemplo, utilizando un par de claves públicas/privadas, se puede estar seguro de que el remitente es quien dice ser. Además, como la firma se calcula utilizando el encabezado y la carga útil, también puedes verificar que el contenido no ha cambiado.[5]

Spring Security proporciona un marco de trabajo robusto para aplicaciones web, que, junto con el uso de JSON Web Tokens (JWT) y la autenticación sin estado (Stateless Authentication), permite implementar de manera segura y eficiente los procesos de autenticación y autorización en la aplicación. Estas herramientas combinadas garantizan que solo los usuarios autorizados accedan a los recursos protegidos, cumpliendo con los más altos estándares de seguridad.

### **2.7.5- Configuración del uso compartido de recursos entre orígenes (CORS)**

El intercambio de recursos de origen cruzado (CORS, por sus siglas en inglés), es un mecanismo basado en cabeceras HTTP que permite a un servidor indicar cualquier dominio, esquema o puerto con un origen distinto del suyo desde el que un navegador debería permitir la carga de recursos. CORS también se basa en un mecanismo por el cual los navegadores realizan una solicitud de "verificación previa" al servidor que aloja el recurso de origen cruzado, con el fin de comprobar que el servidor permitirá la solicitud real. En esa comprobación previa, el navegador envía cabeceras que indican el método HTTP y las cabeceras que se utilizarán en la solicitud real.[6]

Por razones de seguridad, los navegadores restringen las peticiones HTTP de origen cruzado iniciadas desde scripts. Por ejemplo, XMLHttpRequest y la API Fetch siguen la Política Same-origin. Esto significa que una aplicación web que utilice esas API solo puede solicitar recursos del mismo origen desde el que se cargó la aplicación, a menos que la respuesta de otros orígenes incluya las cabeceras CORS adecuadas.[6]

El mecanismo CORS soporta peticiones seguras de origen cruzado y trasferencias de datos entre navegadores y servidores. Los navegadores modernos utilizan CORS en API como XMLHttpRequest o Fetch para mitigar los riesgos de las peticiones HTTP de origen cruzado.[6]

En aplicaciones desarrolladas con Spring Boot y Spring Security, CORS se configura para definir qué dominios están autorizados a acceder a los recursos y qué métodos HTTP son permitidos. Esta configuración garantiza que solo las solicitudes autorizadas sean aceptadas, contribuyendo a una mayor seguridad.

1. **Configuración de Orígenes Permitidos**: Define qué dominios pueden acceder a la API, limitando el acceso a solicitudes provenientes de orígenes no confiables.
2. **Preflight Requests**: Responde a las solicitudes de pre-verificación (OPTIONS) que envía el navegador para validar que el servidor permita las solicitudes CORS.

### **2.7.6- Protección contra Inyecciones SQL**

Uno de los peligros, más usuales de los sistemas web, son las inyecciones SQL. La inyección de SQL es un tipo de ciberataque encubierto en el cual un hacker inserta código propio en un sitio web con el fin de quebrantar las medidas de seguridad y acceder a datos protegidos. Una vez dentro, puede controlar la base de datos del sitio web y secuestrar la información de los usuarios.[7]

Los ataques de inyección de SQL únicamente son viables cuando un sitio web carece de un *saneamiento de entrada* adecuado: el proceso que vela por que la información que introducen los usuarios finales no pueda colarse por ningún resquicio y funcionar como código ejecutable en el servidor. Esto requiere más trabajo por parte del desarrollador, pero, en última instancia, protege frente a la inyección de SQL, las secuencias de comandos en sitios cruzados y otras clases de ataques a sitios web.[7]

Los hackers recurren a los ataques de inyección de SQL con el fin de introducirse en la base de datos de un sitio web. A veces solo quieren eliminar datos para provocar el caos y, en otras ocasiones, lo que buscan es editar la base de datos, especialmente en el caso de sitios web financieros. En el momento en que el hacker ha logrado el control de la base de datos, ya es fácil interferir en los saldos de las cuentas de los clientes y mandarse dinero a su propia cuenta.[7]

Sin embargo, a menudo lo que elciberdelincuente quiere son los datos de usuario guardados en el sitio web, como las credenciales de inicio de sesión. Estos datos de inicio de sesión robados puede emplearlos para realizar acciones en nombre de los usuarios afectados o reunirlos en una gran lista que luego venderá a otros ciberdelincuentes en la red oscura. Las personas que compran información robada lo hacen, frecuentemente, con la finalidad de robar identidades y cometer fraudes.[7]

Para prevenir dichos ataques, se ha utilizado el control que establece Spring Data JPA y la validación de la entrada de datos a la aplicación y el sistema, que no resulten en una consultas o palabras reservadas de SQL que resulten perjudiciales para el sistema.

Solución de Spring Data JPA contra las inyecciones SQL:

* **Uso de JPQL (Java Persistence Query Language):** JPQL se utiliza para escribir consultas en Spring Data JPA y está orientado a objetos. A diferencia de SQL, que trabaja directamente con tablas y columnas, JPQL opera sobre entidades y atributos. Esto reduce el riesgo de inyección SQL, ya que las consultas no interactúan directamente con el esquema de la base de datos.
* **Consultas Basadas en Métodos Derivados:** Métodos Derivados en Spring Data JPA permiten generar consultas automáticamente basadas en el nombre del método en el repositorio. Por ejemplo, un método como findByUsername(String username) genera una consulta segura basada en el nombre del método y los parámetros proporcionados, sin exponer la base de datos a inyecciones SQL.
* **Uso Seguro de Parámetros:** Cuando se utilizan parámetros en consultas JPQL, Spring Data JPA maneja estos parámetros de manera segura. Los parámetros se envían al motor de consultas de forma que están adecuadamente escapados y no pueden modificar la consulta original, lo que previene la inyección SQL.
* **Protección en Consultas Nativas:** Si necesitas ejecutar consultas SQL nativas en Spring Data JPA, es crucial utilizar parámetros nombrados o posicionados en lugar de concatenar directamente cadenas SQL. El uso de @Query con parámetros seguros garantiza que las entradas del usuario no puedan modificar la consulta de manera maliciosa.

### 2.7.7- Monitoreo y registro de actividades de seguridad

Se tiene un sistema de registro que permite almacenar cada actividad que el usuario hay realizado y no es eliminable a nivel de código ya que está protegido, contra eliminaciones o modificaciones tanto a nivel de la API como directamente en la base de datos.

La implementación de estas medidas seguridad es un buen comienzo para la protección de un sistema, pero en un terreno donde las amenazas van evolucionando diariamente, es siempre mantener a los sistemas actualizados en el aspecto de seguridad.