Sistema de Prestación de Servicios de Certificación Electrónica

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Mes de 2023

Autor

Tomás Calderón López

Tutor/a

Ángel Luis Sánchez Lázaro

Certificado de los tutores

El Dr. Ángel Luis Sánchez Lázaro, Profesor Titular de Universidad del Departamento de Informática y Automática de la Universidad de Salamanca

Certifica:

Que el trabajo titulado “Sistema de Prestación de Servicios de Certificación Electrónica” ha sido realizado bajo su dirección por D. Tomás Calderón

López y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo

de Fin de Grado de la titulación Grado en Ingeniería informática.

Y para que así conste a todos los efectos oportunos.

En Salamanca, a 7 de septiembre de 2023

Fdo: Ángel Luis Sánchez Lázaro

Firmado por SANCHEZ LAZARO ANGEL LUIS - ***4852** el día 07/09/2023 con un certificado emitido por AC FNMT Usuarios

Resumen

El proyecto de fin de carrera trata sobre una aplicación web, cuyo objetivo central es la configuración de un sitio web seguro que se comunique por medio de HTTPS, además de mostrar y realizar ciertos procesos relacionados con la seguridad, como firma de mensajes y autenticación con certificados y claves con fines didácticos.

Los usuarios en la web se podrán crear una cuenta con credenciales mediante un proceso de verificación con un código por correo, y, una vez autenticados y verificados, podrán realizar los distintos procedimientos, a la vez que son informados del proceso, obteniendo al final de este los resultados reales, como un certificado personal.

La página web se ha construido en el entorno de Spring Boot, un framework de Java, y en concreto para todo lo relacionado con la seguridad, el módulo Spring Security.

Para la creación, gestión y almacenamiento de claves criptográficas y certificados digitales, se ha utilizado la herramienta Keytool, una herramienta de línea de comandos, también accesible desde Java.

Para la interfaz del usuario se ha utilizado el motor de plantillas Thymeleaf, que permite la creación de vistas HTML con renderizado dinámico.

Palabras clave: seguridad, certificados digitales, criptografía, claves, Java, HTML, Spring, autenticación

Abstract

The end-of-degree project is a web application, whose main objective is the configuration of a secure website that communicates via HTTPS, in addition to displaying and performing certain security-related processes, such as message signing and authentication with certificates and keys for educational purposes.

Users on this website will be able to create an account with credentials through a verification process with a code by mail, and, once authenticated and verified, they will be able to perform the different procedures, while being informed of the process, obtaining at the end of the process the actual results, such as a personal certificate.

The web page has been built in the Spring Boot environment, a Java framework, and specifically for everything related to security, the Spring Security module.

For the creation, management and storage of cryptographic keys and digital certificates, the Keytool, a command line tool, also accessible from Java, has been used.

For the user interface, the Thymeleaf template engine has been used, which allows the creation of HTML views with dynamic rendering.

Keywords: security, digital certificates, cryptography, keys, Java, HTML, Spring, authentication

Índice

Índice de ilustraciones

1. Introducción

La era digital ha traído al mundo multitud de tecnologías y servicios que agilizan y facilitan los procesos de la vida diaria, siendo el más extendido de lejos la comunicación web o internet, pero por desgracia, la seguridad no está nunca garantizada y no están exentas de ataques de todo tipo con malas intenciones.

Para acceder a internet hoy en día, lo más común es utilizar un navegador web, un cliente, que utiliza en primera instancia tecnología HTTP, que se conecta a servidores en línea. Algunas veces las navegaciones no tienen ningún tipo de compromiso para el usuario, ya que no gestionan ningún tipo de dato sensible, pero ya la mayoría sí lo hace, como por ejemplo el manejo de credenciales (usuario y contraseña) y la comunicación entre cliente y servidor que enviaba y recibía esos datos planos era vulnerable.

En este contexto nace en 1994 SSL, un protocolo seguro de transport, implementado por primera vez en el navegador Netscape y cuyo objetivo era proporcionar seguridad, como el cifrado de la información, en las comunicaciones web, aunque no fue nunca lanzada por sus numerosos inconvenientes. SSL fue evolucionando debido a los fallos y vulnerabilidades que presentaba en varias versiones, pasando por SSL 2.0, la primera versión pública, introdujo mejoras significativas, todavía con vulnerabilidades, como el "Padding Oracle". De ahí evolucionó a SSL 3.0, con un gran rediseño y mejora, pero no se tardó en encontrar otras vulnerabilidades, como la de "Poodle".

TLS 1.0 nace como actualización de SSL 3.0 y ha seguido evolucionando con los años y volviéndose más eficiente cada vez. La última versión estable actualmente es TLS 1.3. TLS se integra con HTTP, y nace la nomenclatura HTTPS, que actualmente es el protocolo por excelencia del internet, no hay organización o sitio web que se precie que

no utilice esta tecnología, que ya no es un “plus”, sino una obligación, llevando a HTTP al desuso prácticamente.

En una explicación precaria e introductoria, esta tecnología utiliza el protocolo de “handshake”, donde primeramente ocurre un intercambio de mensajes para acordar las “reglas” de la comunicación y para la comunicación como tal, la tecnología principal para conseguir la confidencialidad es el cifrado simétrico. Cliente y servidor intercambian sendos certificados digitales, que son archivos que sirven para identificar de forma fiable a los interlocutores y que contiene una clave pública. Cliente y servidor, cada uno, tiene un par de claves, una pública y una privada. Cualquier emisor utiliza la clave pública para cifrar lo que quiera enviar al receptor, el mensaje cifrado recorre la red, una amalgama de números y letras sinsentido e indescifrables, que cuando llegan al receptor, es capaz de descifrar con su clave privada.

A lo largo de esta memoria se describirán en detalle los procesos y tecnologías involucrados en el proyecto, además de otros puntos clave que son:

- **Objetivos:** Definir lo que se intenta conseguir en este proyecto.

- **Conceptos teóricos:** Repaso y explicación de los fundamentos teóricos que ayudarán a comprender las bases del proyecto.

- **Técnicas y herramientas usadas:** Los medios y programas utilizados en el proyecto para llevarlo a cabo y explicación de cómo se han usado.

- **Aspectos relevantes del desarrollo:** En este apartado se hablará de la experiencia y puntos claves del desarrollo del proyecto.

- **Conclusión y líneas de trabajo futuras:** Al final del trabajo se resumirá lo logrado con la realización del proyecto, así como la utilidad en aplicaciones reales y ciertos puntos a tener en cuenta para llevar a cabo de forma satisfactoria la adaptabilidad del contexto en el futuro.

2. Objetivos del proyecto

En este apartado se resumirán los objetivos principales que se desean alcanzar con la realización del proyecto, los cuales se pueden dividir en funcionales, técnicos y didácticos:

5.1 Objetivos funcionales

El objetivo funcional del proyecto es tener una página web configurada de forma segura, que ofrezca una comunicación segura con el cliente, así como un servicio de registro y autenticación con credenciales. Cuando el usuario esté autenticado, podrá crear certificados personales emitidos por la web con los cuales se podrá autenticar y podrá crear y verificar firmas digitales.

Lo primero es gestionar el portal de manera segura, activando HTTPS con un certificado que identifique a la página web como segura. Esta página también tendrá un formulario de registro de usuarios con verificación por email para asegurarse de evitar la suplantación de mail.

Una vez registrado, el usuario se podrá autenticar con sus credenciales, y una vez verificado podrá acceder a las funcionalidades de la página web.

La web será capaz de generar y emitir certificados personales con los que autenticarse y de generar y verificar firmas digitales basadas en certificados, en ambos casos con opción de descarga de los archivos resultantes para uso personal del usuario.

Cuando un usuario haya generado, descargado e instalado el certificado personal, el navegador detectará esto y cuando no haya sesión abierta, le ofrecerá autenticarse con certificado personal.

5.2 Objetivos técnicos

El objetivo técnico principal es la seguridad de la web, habilitar la gestión segura y desarrollarla de forma responsable para que su adaptación a cualquier circunstancia sea lo más proactiva posible.

Al haber tantos procesos distintos dentro del proyecto, desde leer archivos o gestionar almacenes de claves, pasando por la interfaz del usuario y hasta firmar y crear certificados, se busca que haya granularidad y modularidad en el código del proyecto, con clases cuyas funcionalidades estén bien definidas y separadas para la mayor comprensión del proyecto.

Asimismo, al manejar constantemente claves y contraseñas, se evita la aparición explícita de los mismos a lo largo de todo el código, para los que se utilizará encriptadores de contraseñas, que se almacenarán de formada cifrada en la base de datos, y archivos locales de credenciales, para evitar que los valores aparezcan donde no deberían estar.

5.3 Objetivos didácticos

Una parte especial del trabajo no solo significa que pueda realizar procesos relacionados con la seguridad, sino que también los comprenda y aprenda cómo funcionan de forma subyacente.

En cada apartado, en la web habrá las explicaciones detalladas, tanto de las tecnologías y protocolos como de la realización de los procesos paso por paso.

El objetivo principal desde el punto de vista didáctico, es que se extienda el conocimiento sobre las configuraciones de webs seguras, el importante uso del cifrado asimétrico y los certificados para firmar, cifrar y autenticar la información, así como los pasos que hay por debajo de todas estas técnicas relacionadas con la seguridad, para que se haga un uso eficiente y responsable de las mismas.

3. Conceptos teóricos

A continuación, se explicarán los conceptos básicos teóricos para comprender el marco de la seguridad que rodea a la aplicación.

3.1 Objetivos didácticos

El cifrado es un proceso de transformación de datos legibles, que pueden estar en texto claro o contener información y pueden ser leídos, comprendidos o interpretados para obtener información, en datos en un formato completamente ilegible y carente de sentido, utilizando un algoritmo y una clave, utilizada para cifrar y descifrar, cuyo objetivo es proteger la confidencialidad y seguridad de la información, siendo que el poseedor de la clave es el único capaz de descifrar la información.

En general, hay dos tipos:

- Cifrado simétrico, donde la clave utilizada para cifrar y descifrar es la misma.
- Cifrado asimétrico, o de clave pública, es un método de cifrado que utiliza un par de claves pública y privada.

El cifrado asimétrico es más eficiente en términos computacionales, pero tiene un gran inconveniente: el intercambio de la clave. El hecho de que haya que cifrar y descifrar con la misma clave, hace que ambas partes deban tener la clave, que uno de ellos generará y tendrá que enviar al otro. Es en ese envío donde está el problema, pues la propia clave se puede interceptar y de nada valdría cifrar la información si cualquiera que pueda interceptar y utilizar la clave puede descifrarla también.

En este contexto surge el cifrado asimétrico, creado precisamente para lidiar con este problema. En el cifrado asimétrico se generan dos claves por usuario, una privada y una

pública, diferentes pero relacionadas matemáticamente entre sí, de forma que información cifrada con la clave pública, se pueda descifrar con la clave privada. El usuario, como el nombre indica, pone a disposición su clave pública para que cualquiera que quiera comunicarse con él de forma segura, la utilice, cifre la información con ella y la envíe al susodicho, el cual es el único que posee la clave privada, que debe ser suya y solo suya, e implica que sólo él podrá descifrar esa información.

El método es seguro, ya que se garantiza la imposibilidad de obtener la clave privada a partir de la clave pública, por lo que, gracias a este método, el problema de intercambiar claves para obtener la capacidad de cifrar/descifrar se resuelve. Si dos usuarios se quisieran comunicar de forma continua, cada uno tiene su par de claves, ambos utilizan la clave pública del otro para cifrar y cada uno utiliza su clave privada para descifrar la información cifrada por el emisor con su clave pública.

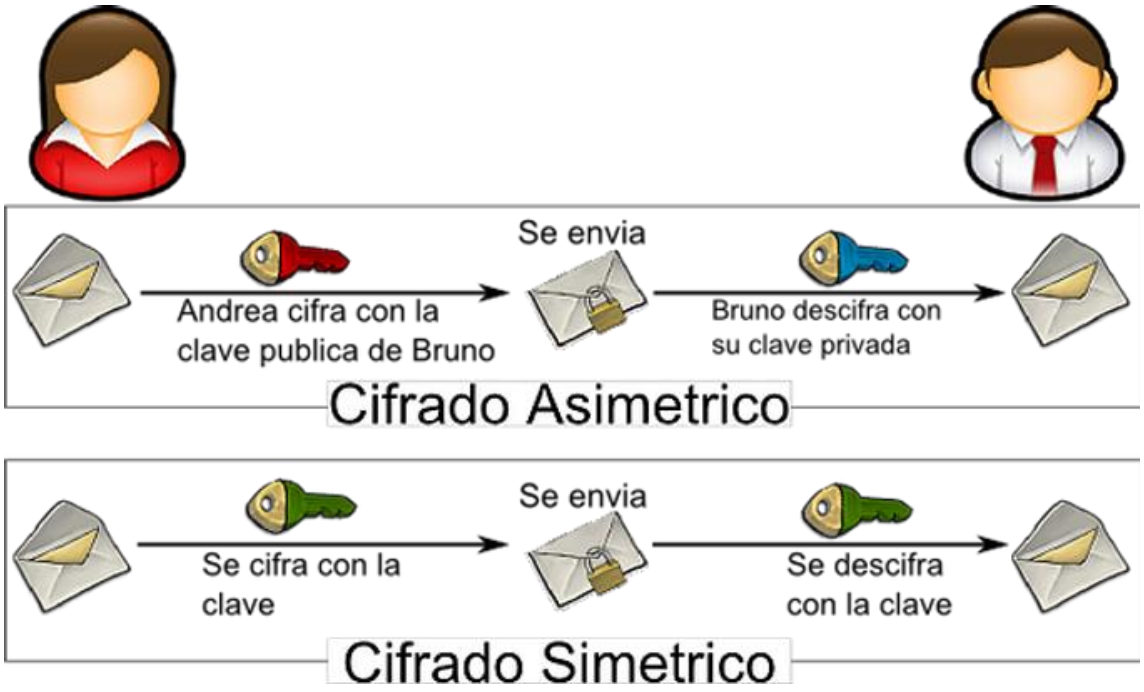
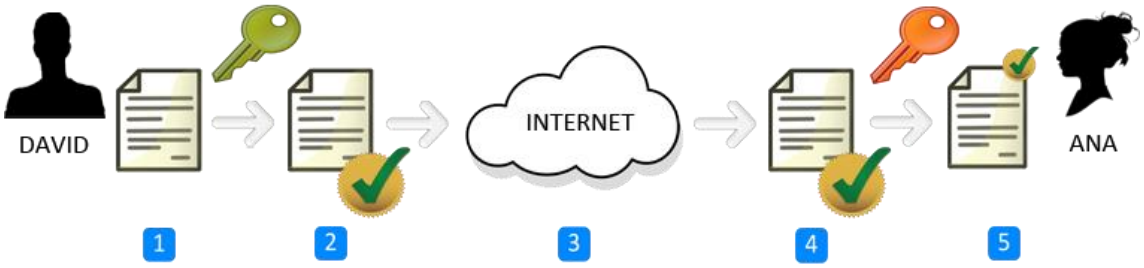


Figura 1: Tipos de cifrado

Además, el cifrado asimétrico no solo sirve para proteger la información y utilizar la clave pública para cifrar y enviar datos cifrados, tiene otra gran funcionalidad igual de importante: las firmas digitales. En este caso, el dueño del par de claves puede utilizar su clave privada para cifrar unos datos que pueden ser descifrados utilizando la clave pública. Esto no se utiliza para proteger la información, lógicamente, pues la clave pública es conocida por todo el mundo, sino que, sirve para asegurarse de que el remitente del mensaje es quien dice ser. Si al descifrar con la clave pública un mensaje firmado con clave privada no hay ningún problema (modificación del mensaje o falsificación), el receptor puede estar seguro de que el mensaje es de quien dice ser. (Wikipedia, s.f.)



1. David redacta un mensaje.
2. David firma digitalmente el mensaje con su **clave privada**.
3. David envía el mensaje firmado digitalmente a Ana a través de internet, ya sea por correo electrónico, mensajería instantánea o cualquier otro medio.
4. Ana recibe el mensaje firmado digitalmente y comprueba su autenticidad usando la **clave pública** de David.
5. Ana ya puede leer el mensaje con total seguridad de que ha sido David el remitente.

Figura 2: Proceso de la firma digital

El cifrado asimétrico tiene una gran aplicabilidad y se utiliza en una gran variedad de entornos, aquí algunos ejemplos:

- **Autenticación:** Un ejemplo que se utiliza en este proyecto. Las claves públicas se utilizan para autenticar la identidad de los usuarios en los servidores.
- **Protección de claves:** Ya que el cifrado simétrico es más rápido y eficiente computacionalmente hablando, una práctica común es utilizar el cifrado simétrico para cifrar la información y utilizar el cifrado asimétrico para asegurar el envío de la clave simétrica al destino.
- **Seguridad en correo electrónico:** Tanto para cifrar como para firmar mensajes.

Hay muchos algoritmos distintos de clave asimétrica, siendo algunos de los más populares RSA, DSA, ElGamal, Curvas Elípticas…

3.2 Certificados digitales

Un certificado digital es un documento firmado por una Autoridad de Certificación (CA), una entidad que ambas partes consideran de confianza, que puede representar a una persona, empresa, etc., y que sirve para asociar su clave pública con más información y datos. Cuando se use la clave privada equivalente a esta clave pública, se podrá saber que ha sido usada por el sujeto que aparece representado en dicho certificado. Estos certificados contienen varios y diversos campos para el usuario, pero los más comunes son:

- **Nombre Común (CN):** El sujeto del certificado, normalmente suele contener el nombre del dominio.

- **Organización (O):** Nombre de la empresa u organización a la que pertenece el sujeto.
- **Unidad de Organización (OU):** Unidad dentro de la organización.
- **Localidad (L):** Localidad geográfica del sujeto
- **Provincia (ST):** Provincia del sujeto.
- **Código de País (C):** Código del país del sujeto, codificado con dos letras, estandarizado por la norma ISO 3166-1 (Ministerio de Justicia del Gobierno de España)

Aparte de estos campos relacionados con la información del usuario, tiene otros campos únicos, como el número de serie, el firmante, las fechas de creación y expiración, la clave pública, el algoritmo de firma…

Un certificado garantiza la autenticidad de las personas y entidades a las que representa en un intercambio de información. Generalmente, sirve para autenticar la identidad del usuario de forma electrónica ante terceros, para firmar datos o documentos que garanticen su integridad y autoridad, y para cifrar datos. (Instituto Nacional de Estadística, s.f.)

Hay varios estándares e infraestructuras para crear y gestionar certificados, en este proyecto se trabaja con el extendido estándar X.509 (Wikipedia, s.f.), cuya característica principal, además de las mencionadas anteriormente, comunes para cualquier certificado, es la existencia de una cadena de certificados jerárquica de confianza, que demuestra la confianza en la identidad de las entidades certificadas. En lo alto de la cadena están las Autoridades de Certificación (CA), que pueden emitir certificados intermedios que a su vez tengan la capacidad de expedir otros certificades y ampliando la cadena. Cuando se

verifica un certificado de una cadena, se sigue un proceso de verificación ascendente desde el certificado personal hasta el certificado raíz (Los certificados de las CA). Esto es esencial, ya que no solo se verifica el propio certificado, sino la cadena al completo detrás de él, lo que permite una fácil ampliación de una red segura a través de internet.

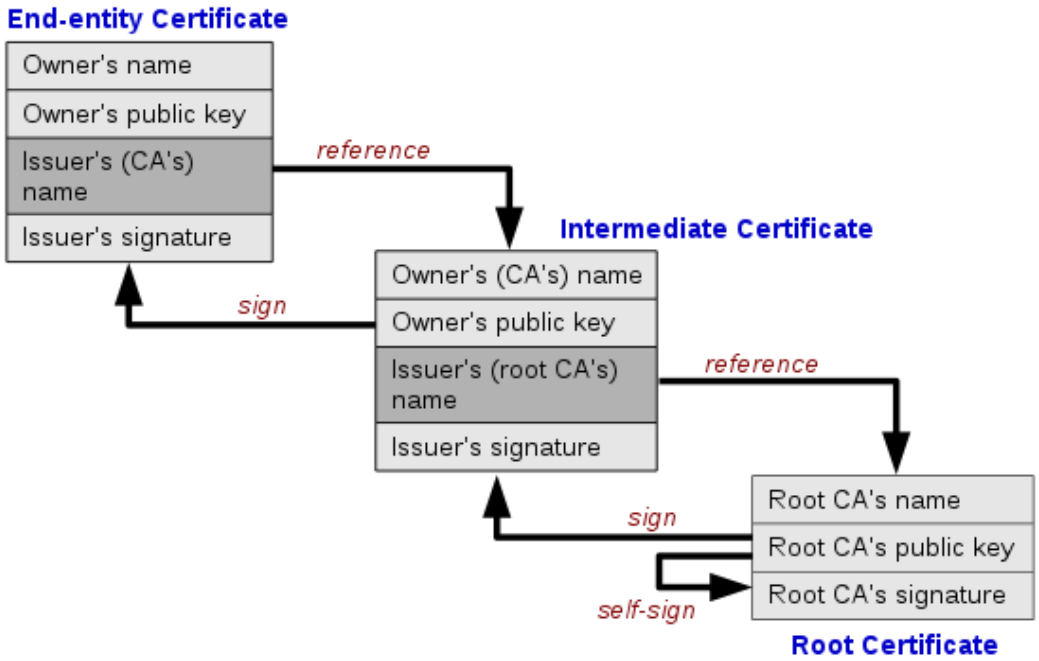


Figura 3: Esquema representativo de una cadena de certificados

Se pueden almacenar en varios formatos:

- **.cer o .crt:** Estas extensiones formateadas en binario o Base64 contienen la información sobre la clave pública y más detalles sobre el emisor o extensiones.
- **.der:** El formato DER es un formato binario y se utiliza para codificar datos de una manera que es independiente de la plataforma y del lenguaje de programación.
- **.pem:** Los pem pueden contener claves privadas aparte de certificados. Se codifican en Base64 entre los delimitadores "-----BEGIN CERTIFICATE-----" y "-----END CERTIFICATE-----"

- **.p7c:** Este formato se utiliza para certificados, que a veces tienen cadenas, y además puede incluir firmas digitales y datos cifrados.
- **.pfx o p12:** Estas extensiones se utilizan para archivos que aparte de contener el certificado completo también acoplan la clave privada, que es común para aplicaciones de seguridad como esta pues protege de forma conjunta clave privada y certificado. Suelen estar protegidos por una contraseña. En este proyecto se usan para la descarga del usuario para poder usar tanto el certificado como su clave privada generada al crear el certificado.

3.3 Encabezado HTTPS

Para comprender HTTPS, lo primero es saber qué es HTTP. Es el protocolo de transferencia de hipertexto, de la capa de aplicación, utilizado mundialmente en la red para cargar páginas web. Permite la transmisión y petición de información a través de la World Wide Web (WWW).

Se gestiona mediante solicitudes, en las cuales hay información como la versión, la URL, el método (Determina la acción, hay 4: GET, POST, PUT, DELETE), encabezados y cuerpo. Cada solicitud recibe una respuesta; después de la solicitud del cliente, el servidor responde. Estas solicitudes y respuestas están en un formato de texto, que es legible por las personas y esto es bueno por razones como la facilidad para el desarrollo, la depuración… Pero tiene un gran inconveniente: no es seguro. Si en cualquier momento se necesitara transmitir información sensible a través de este protocolo, se estaría cometiendo una imprudencia y cometiendo un gran error, ya que con el más sencillo analizador de red se podrían interceptar los paquetes y consultar la información de los mismos de forma muy sencilla.

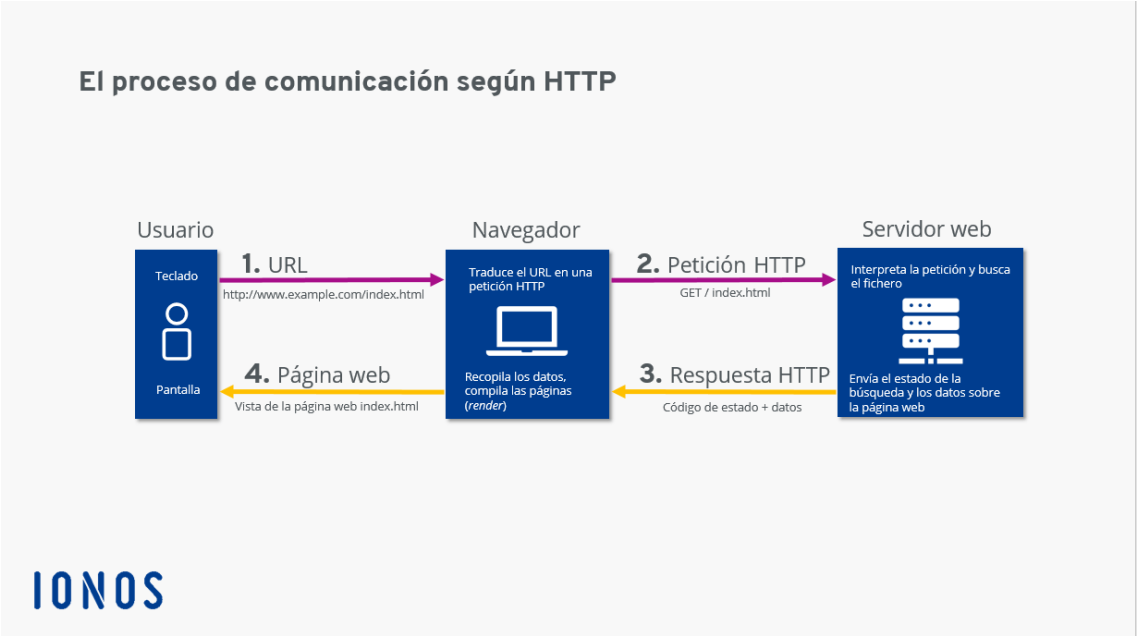


Figura 4: Proceso de comunicación HTTP

Hoy en día esto ya es totalmente impensable y desfasado, es obligatorio que se garantice la información de la seguridad de alguna forma y no dejarla tan expuesta, como pasa en HTTP, por lo que pronto se desarrolló una tecnología nueva para subsanar este gran problema: SSL/TLS.

Un breve resumen de su historia: Secure Socket Layer es un protocolo que se basa en el cifrado, desarrollado en 1995, cuya primera versión (SSL 1.0) no se publicó por sus numerosos fallos y con versiones sucesivas que iban enmendando los errores de sus predecesoras. Después de la versión SSL 3.0, que pertenecía a Netscape, cuando esta se desdibujó del proyecto, la nueva versión pasó a llamarse TLS (Transport Layer Secutiy), el sucesor de SSL. Hoy en día TLS se encuentra en su versión 1.3 y está mucho más blindado que todas las versiones anteriores de SSL/TLS, aunque no exenta de fallos, pero igualmente muy segura.

SSL/TLS en las comunicaciones entre cliente y servidor: encripta los datos, garantiza que los interlocutores son quienes dicen ser y garantiza la integridad del mensaje, verificando que no se haya modificado. Funciona de la siguiente manera:

- Al arrancar la comunicación entre cliente y servidor, se especifica qué versión de TLS (TLS 1.0, 1.2, 1.3, etc.) van a utilizar
- Se decide qué algoritmos de cifrado se van a usar
- Se autentica la identidad del servidor utilizando el certificado TLS de este
- Se generan claves de sesión (asimétricas) para encriptar los mensajes entre cliente y servidor después de completar el protocolo de enlace

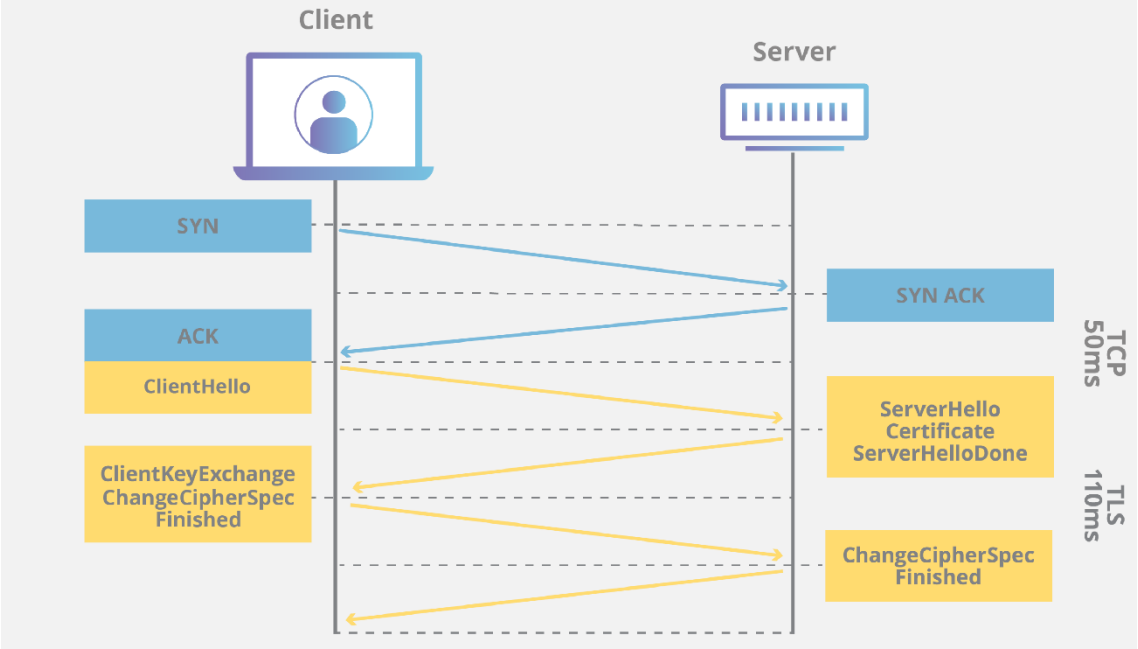


Figura 5: Esquema del “handshake” de TLS

Una vez se ha establecido la sesión, la comunicación entre cliente y servidor ya es más segura y estará completamente cifrada para actores externos. HTTPS no es más que HTTP con TLS. HTTPS se basa en la transmisión de los certificados TLS/SSL, que verifican que un determinado proveedor es quien dice ser. Un servidor comienza a usarlo cuando lo configura con un certificado de sitio emitido por una CA. (Cloudflare, s.f.)

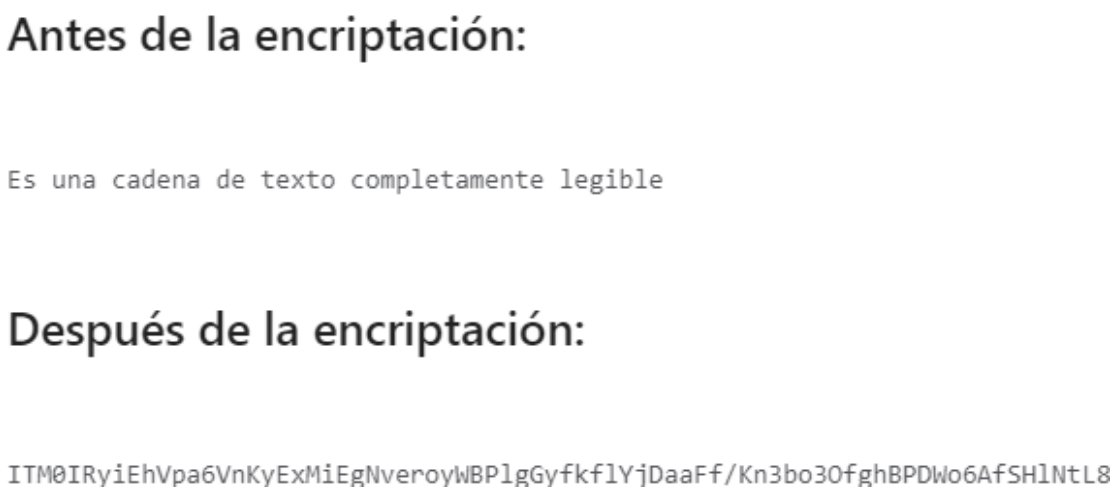


Figura 6: Ejemplo en la diferencia en la muestra de información en HTTP (arriba) y HTTPS (abajo)

3.4 Autenticación

La autenticación, en el contexto de la informática, es un proceso por el cual se verifica la identidad de los usuarios en un sistema, cuyo objetivo principal es confirmar que el solicitante es quien dice ser y que además forma parte y tiene permisos para acceder a ese sistema. Su objetivo es prevenir el acceso no autorizado y garantizar que solo las entidades adecuadas hacen uso de los recursos de un servidor.

Antes de poder autenticarse, es lógico registrarse en el sistema previamente. Hay muchas formas de llevar a cabo una autenticación, ya que es un proceso delicado donde se manejan datos sensibles, como las credenciales del usuario:

- **Credenciales:** Es el método de autenticación más común en aplicaciones web y sistemas en línea. El usuario proporciona un nombre de usuario y una contraseña previamente registrados.

- **Certificados digitales:** Se utilizan en entornos más seguros y se basan en la criptografía de clave pública. Un certificado digital, emitido por una Autoridad de Certificación (CA) de confianza, verifica la identidad de una entidad.

- **Biometrías:** Se basa en características físicas o de comportamiento únicas del usuario, como huellas dactilares, reconocimiento facial, voz, etc.

4. Técnicas y herramientas

4.1 Lenguajes, bibliotecas y frameworks

Java

Java es un lenguaje de programación orientado a objetos de propósito general ampliamente utilizado, entre otros, para codificar aplicaciones web y aplicaciones para Java Virtual Machine. Tiene una gran portabilidad y gama de bibliotecas disponibles para su uso. En el proyecto, es la base que utilizan los frameworks que se comentarán a continuación.

Figura 7: Logo de Java

La biblioteca Java.Security ha sido ampliamente utilizada en este proyecto.

Spring Boot

Es un framework de aplicaciones Java que simplifica la configuración y desarrollo de webs empresariales gracias a su notación con etiquetas (@Spring). Ofrece multitud de módulos, en este proyecto principalmente se han usado:

- **JPA:** JPA significa "Java Persistence API". Es una especificación que proporciona una forma estándar de interactuar con bases de datos relacionales desde aplicaciones Java. JPA permite a los desarrolladores trabajar con objetos Java en lugar de consultas SQL directas, simplificando así la persistencia y recuperación de datos en bases de datos. Se basa en el concepto CRUD, en referencia a sus operadiones:
    - Create: Crear nuevos registros.
    - Read: Leer/recuperar datos.
    - Update: Actualizar registros existentes.
    - Delete: Eliminar registros.
- **Spring Security:** es un framework de seguridad de aplicaciones para el ecosistema de Spring en Java. Proporciona un conjunto de herramientas y componentes que permiten a los desarrolladores asegurar sus aplicaciones de manera efectiva. Spring Security se utiliza para autenticar usuarios, autorizar sus acciones y proteger los recursos de una aplicación web.
- **MVC:** Spring MVC es un marco de desarrollo web que implementa el patrón MVC en aplicaciones Java. Permite a los desarrolladores crear aplicaciones web de manera estructurada y escalable. El Controlador en Spring MVC gestiona las

solicitudes HTTP y coordina la lógica de la aplicación, el Modelo representa los datos y la Vista controla la presentación.

Figura 8: Logo de Spring

Spring facilita la resolución de problemas comunes para los desarrolladores a la hora de diseñar sus modelos, estando entre sus características más apreciadas la inversión e inyección de dependencias, el contenedor IoC que gestiona los objetos y la fácil inicialización de las instancias con el autowiring. (Anónimo, s.f.)

H2

H2 es un sistema de gestión de bases de datos relacional (DBMS) de código abierto desarrollado en Java. Se caracteriza por ser una base de datos embebida y ligera que se utiliza comúnmente en aplicaciones Java para desarrollo y pruebas. Será utilizada en este proyecto para almacenar las credenciales de los Usuarios por su versatilidad, fácil administración y rapidez.

BouncyCastle

Bouncy Castle es una biblioteca de criptografía de código abierto desarrollada en Java. Su nombre completo es "The Legion of the Bouncy Castle" y es ampliamente utilizada en aplicaciones de seguridad y criptografía.

Esta biblioteca es conocida por ser altamente versátil, lo que la hace popular entre desarrolladores y empresas que requieren soluciones de seguridad sólidas en sus aplicaciones. También ofrece una amplia gama de algoritmos criptográficos, incluyendo cifrado simétrico y asimétrico, funciones hash, generación de números aleatorios, certificados…

Figura 9: Viñeta de The Legion of the Bouncy Castle

Una de las características destacadas de Bouncy Castle es su soporte para algoritmos menos comunes y protocolos criptográficos, lo que la convierte en una elección popular para proyectos que requieren una gama más amplia de opciones de seguridad. También es ampliamente utilizada en aplicaciones de firma digital, seguridad de redes y en la implementación de estándares de seguridad como TLS/SSL.

En el proyecto ha sido ampliamente utilizada en la parte de generación de certificados X.509.

Thymeleaf

Thymeleaf es un motor de plantillas HTML que trabaja en sintonía con Spring Boot. Genera contenido dinámico a partir de la vista que obtiene los valores del modelo. Su objetivo es simplificar y "naturalizar" las plantillas HTML, manteniendo su esencia y legibilidad y también haciendo más sencilla la comunicación entre la interfaz del usuario y el servidor, puesto que son plantillas directamente generadas por el backend que además no solo pueden funcionar por sí mismas, sino que también pueden ser integradas en otros lugares, como por ejemplos frameworks de JavaScript. (Thymeleaf, s.f.)

```
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Raffle Thymeleaf Demo</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
    <h1>RESULTADOS DEL SORTEO</h1>
    <h3>ENHORABUENA A LOS PREMIADOS</h3>
    <table border="1">
      <tr th:each="winner : ${raffle.winners}">
        <td th:text="${winner}">Winner</td>
      </tr>
    </table><br/><br/>
    <a href="/index">Submit another message</a>
</body>
</html>
```

Figura 10: Trozo de código HTML con Thymeleaf

Bootstrap

Bootstrap es un marco de diseño de código abierto que es utilizado para diseñar las interfaces de aplicaciones web a partir de plantillas y elementos básicos ya creados. Lo desarrolló Twitter y hoy en día es mantenido por una gran comunidad.

Figura 11: Logo de Bootstrap

Sobre todo, facilita la aplicación de estilos y animaciones predefinidos que aportan unos grandes cimientos, útiles para personas a las que no se les dé bien el diseño web.

4.2 Entornos de trabajo

En este apartado se describirán las herramientas y programas utilizados para desarrollar el proyecto:

Keytool

Keytool es una herramienta de línea de comandos, que forma parte del conjunto de herramientas de JDK (Java Development Kit). Su uso se resume en la creación y gestión de claves y certificados por medio de ficheros, que en realidad son almacenes y posteriormente podrán ser tratados individualmente con la herramienta o desde el propio Java. Keytool permite también importación y exportación de claves entre almacenes, lo que facilita la migración y la gestión. Utiliza un sistema de alias para identificar cada entrada de sus almacenes.

Figura 12: Java Keytool

Tiene dos formatos posibles de fichero:

- **JKS (Java Keystore):** Este es el formato de almacén de claves predeterminado en Java. Los almacenes de claves JKS utilizan una estructura de archivos específica y pueden contener claves privadas y certificados. Es ampliamente compatible y se utiliza en muchos contextos de seguridad de Java. Los archivos de almacén de claves JKS suelen tener la extensión ".jks". En Java Spring se utilizarán para crear un truststore, el almacén de certificados en los que confía el servidor para autenticarse,
- **PKCS12**: También conocido como PFX, es un formato de almacén de claves estándar de la industria basado en el estándar PKCS12. Los almacenes de claves PKCS12 son compatibles con una variedad de herramientas y plataformas de seguridad, lo que los hace útiles para la interoperabilidad. Los archivos de almacén de claves PKCS12 suelen tener extensiones como ".p12" o ".pfx". En Java Spring se utilizarán para almacenar tanto los certificados y claves privadas del servidor como los de los usuarios de forma individual.

Visual Studio Code

VS Code es un editor de código desarrollado por Microsoft. Al ser un editor sencillo y versátil gracias a la gran cantidad de extensiones, que sirven desde el desarrollo hasta despliegues y depuración, su uso está muy extendido. En este proyecto se han utilizado extensiones de Spring y Java para crear y desarrollar el proyecto de Spring Boot.

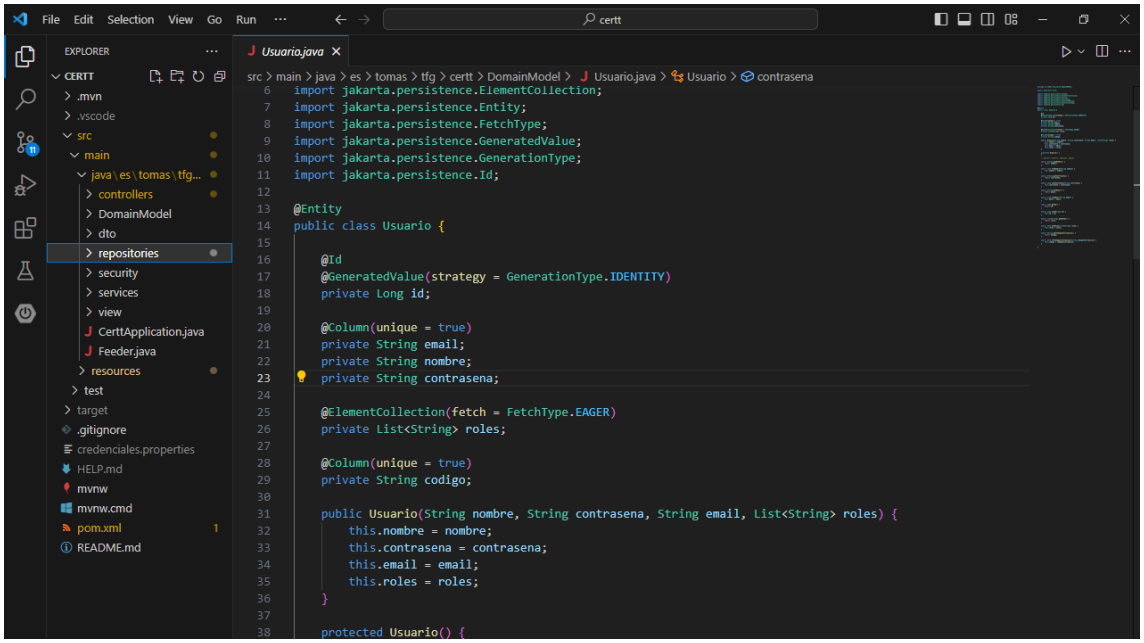


Figura 12: Captura de VSCode

Google Chrome

Google Chrome es un popular navegador web desarrollado por Google conocido por su velocidad, seguridad y sincronización de datos en múltiples dispositivos. Ofrece una interfaz limpia y sencilla, extensiones personalizables y un enfoque en la seguridad del usuario.

Durante todo el desarrollo se ha utilizado Google Chrome como navegador cliente para acceder a la aplicación. Cabe destacar que Chrome utiliza el almacén de certificados del

sistema, por lo que todas las instalaciones de certificados que sean necesarias hacer, serán en este.

5. Aspectos relevantes del desarrollo del proyecto

5.1 Metodología

La naturaleza de este proyecto de basa en la interacción del usuario con la plataforma web, pues como ya se ha visto en los objetivos, las metas del proyecto son diseñar un portal seguro en el que usuarios se puedan registrar y autenticar, tanto con credenciales como con certificados digitales, para aprender cómo funcionan los procesos de creación de certificados, autenticación y demás características relacionadas con la seguridad web.

El hecho de realizar un proyecto con temática de seguridad implica que la propia funcionalidad segura sea el centro del proyecto, lo primero en realizarse y debe hacerse de forma eficiente para construir los cimientos de la aplicación web. Por eso las funcionalidades clave como la autenticación y registro, así como las barreras de acceso a los distintos apartados de la web giran en torno al usuario, ya que todo el tema de la seguridad web está diseñado para la comunicación segura entre cliente y servidor, que sirve para la protección de datos sensibles y de los procesos en la comunicación.

Debido a esta filosofía en la que la implementación de un sistema seguro para usuarios sea la base del proyecto, se ha decidido escoger una metodología acorde esta necesidad expuesta, y esta es la metodología de desarrollo RAD (Rapid Application Development), basada en el desarrollo rápido de aplicaciones.

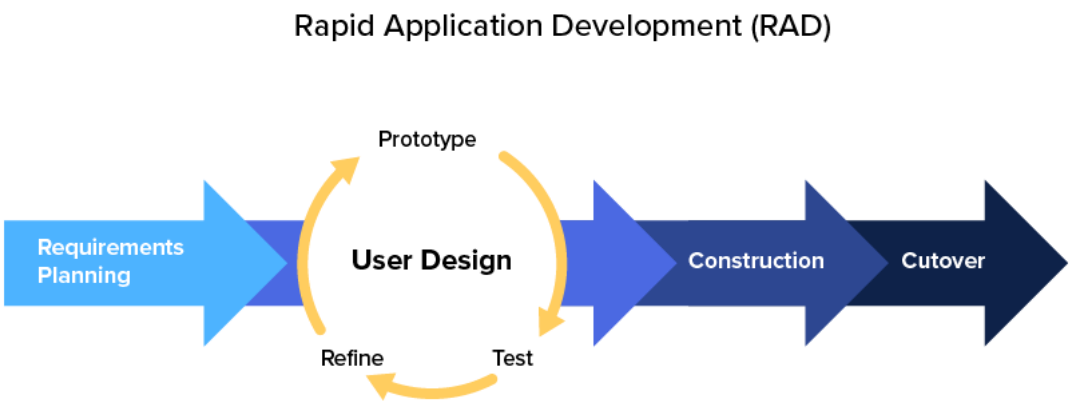


Figura 13: Diagrama de flujo básico de la metodología RAD.

RAD es una metodología que se centra en la entrega rápida de un sistema funcional y de prototipos iniciales con los que se pueda interactuar y de los que se puedan sacar conclusiones en claro desde el principio, construyendo rápidamente funcionalidades básicas, comprobar que funcionen y mejorarlas o cambiarlas a medida que las pruebas reflejen las distintas necesidades. Se busca la colaboración estrecha de la retroalimentación de la experiencia de usuario con el desarrollador, realizando iteraciones cortas en las que se implementa algo en los prototipos, se prueba y se refina, volviendo a probar el prototipo, en un ciclo realizado hasta que se consiga el efecto deseado.

Este enfoque basado en el usuario es esencial para la aplicación, pues la mayoría del ciclo de interacción se basa en métodos relacionados con la seguridad, autenticación… Por lo que es adecuado generar un entorno dinámico que se adapte a los cambios en las decisiones de desarrollo que puedan desviarse de la idea original debido a las características concretas de las tecnologías aplicadas y lo que permiten hacer. Al ser Spring Security una tecnología que desde el principio tiene unas bases predeterminadas

al crear el proyecto, por ejemplo, el inicio de sesión predeterminado se entra en un entorno de prueba y error, desde la configuración de interfaces o modificaciones en las peticiones al servidor hasta la propia configuración de Spring Security, en el que se van realizando pequeños cambios que se prueban como usuario para ver su efecto y se van adaptando a la necesidad del proyecto.

La emisión de certificados personales agrega un nivel adicional de seguridad y confianza a la plataforma. La metodología RAD permite implementar este proceso de manera incremental, lo que significa que se pueden realizar pruebas exhaustivas de cada paso antes de avanzar al siguiente. Además, dado que los certificados personales pueden ser un requisito crítico para los usuarios, su implementación debe ser precisa y fiable desde el principio.

La elección la metodología RAD para un proyecto de portal web seguro con registro, autenticación y emisión de certificados personales entre otros se basa en la necesidad de rapidez, adaptabilidad, seguridad y la capacidad de brindar una experiencia confiable a los usuarios finales. Este enfoque permite enfrentar los desafíos de seguridad de manera efectiva mientras se cumple con los requisitos cambiantes del proyecto y se prioriza la satisfacción y la confianza del usuario.

5.2 Fase de análisis

Tras un pensamiento exhaustivo sobre la posible implementación cuantitativa del portal web, se materializan los objetivos y requisitos del sistema, representando todas las operaciones posibles con los tres estados en los que pueden estar un usuario en el sistema y sus capacidades.

En los objetivos se ven reflejadas las intenciones del sistema, siendo los generales el enfoque y la intención que se tiene al diseñar esta página web y los específicos el cómo hacerlo, y derivados de estos, los requisitos son la formalización y serialización de los pasos necesarios para llevar a cabo los objetivos.

Sin embargo, es común que en primera instancia no se engloben todas las posibles necesidades que pueda llegar a necesitar el desarrollo, por lo que podrá haber desvíos del camino o incongruencias respecto a las siguientes fases en pos del desarrollo natural y las necesidades que vayan surgiendo de la aplicación, el entorno y las tecnologías. A continuación, los extractos más significativos del anexo II, donde se encuentra esta fase completa:

OBJ-01	Desarrollar un sistema de prestación de servicios de certificación electrónica
Versión	1
Autor	Tomás Calderón López
Descripción	El sistema deberá prestar un servicio web que permita al usuario, registrarse, autenticarse con credenciales y certificado, crear certificados personales y crear firmas digitales
Importancia	Muy alta
Sub-objetivos	OBJ-02

Figura 14: Objetivo principal de la aplicación

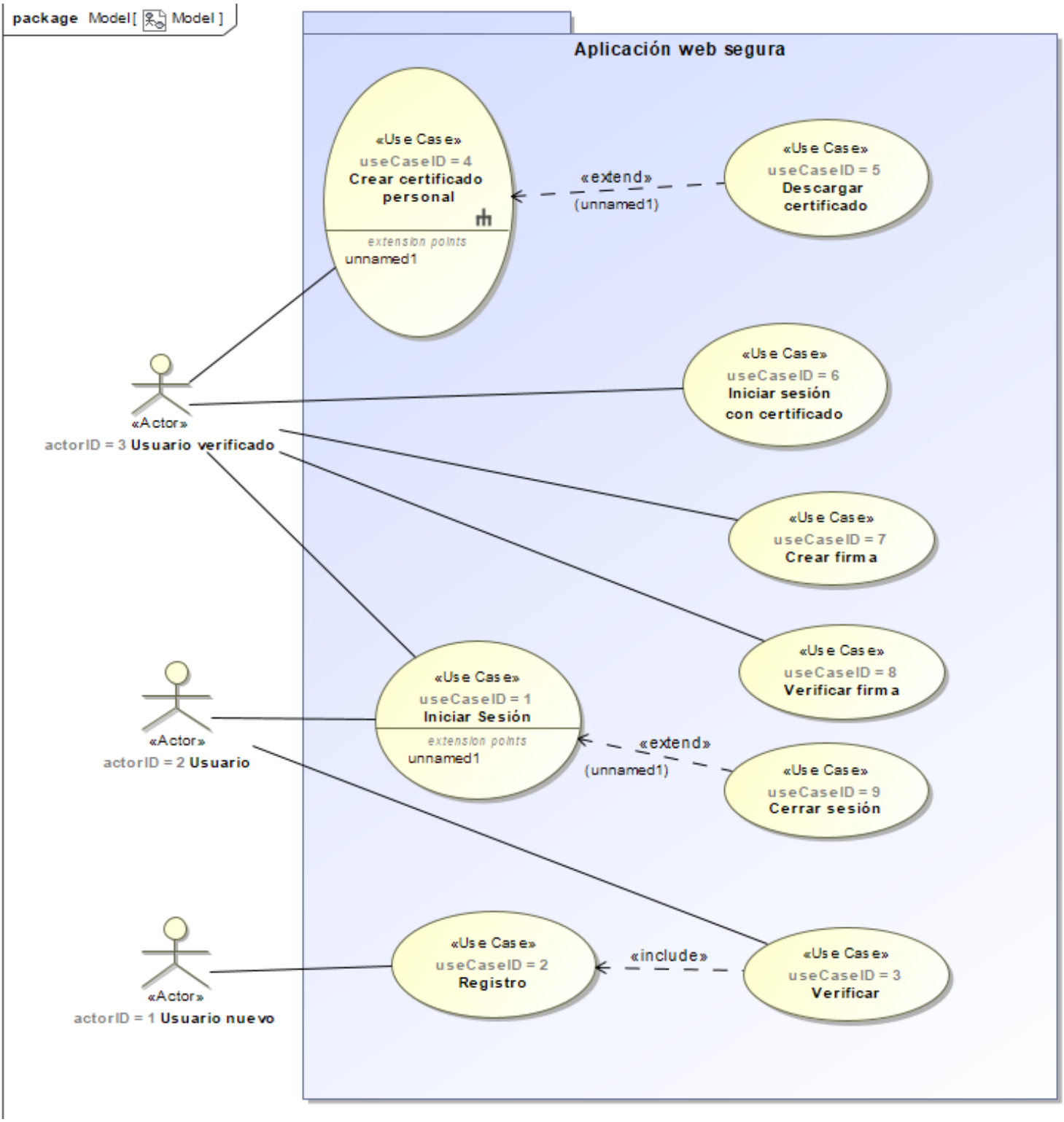


Figura 15: Diagrama de casos de uso

5.3 Fase de diseño

Tras el análisis realizado en la fase previa, se puede llevar a cabo el siguiente paso siguiendo las pautas establecidas en la medida de lo posible.

Diseño arquitectónico

Dado las características específicas de las aplicaciones web y las utilidades del framework que se va a utilizar para desarrollar la web, se ha decido utilizar el patrón arquitectónico MVC web: (Gaetanino Paolone)

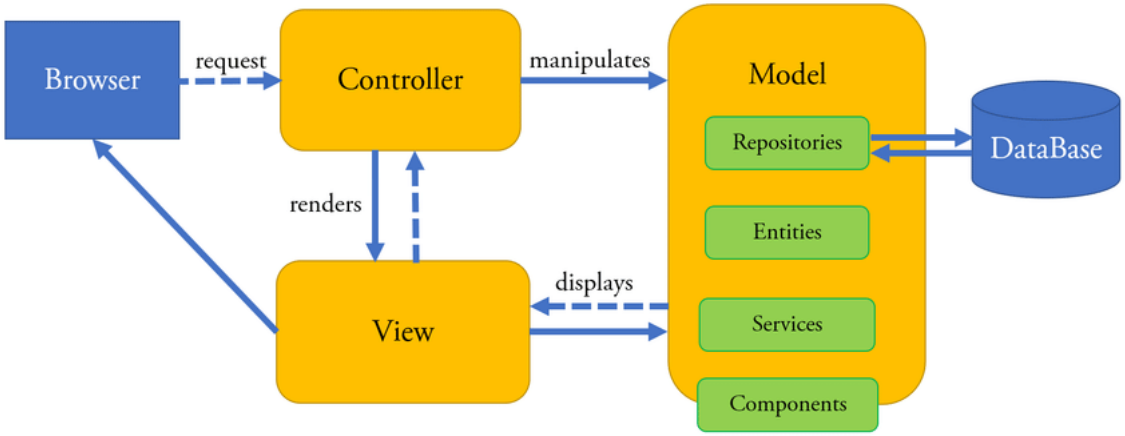


Figura 16: Modelo arquitectónico MVC web, relacionado con el framework Spring

Obviando el cliente, que es el navegador, quien realiza y recibe respuestas, y la base de datos que se comunica con el modelo por medio de los repositorios, en el esquema se pueden ver los tres elementos dominantes:

- **Modelo:** El modelo representa los datos y la lógica de negocio de la aplicación. Es responsable de acceder a la base de datos, realizar cálculos, procesar datos y proporcionarlos a la vista para su presentación. Esta arquitectura no solo engloba el modelo de dominio, sino los repositorios, servicios y demás componentes.

- **Vista:** La vista se encarga de generar el contenido que recibe del modelo después de realizar los cálculos en el modelo de negocio y de enviar el resultado al usuario.
- **Controlador:** El controlador es el mediador entre el sistema y el cliente, el cual realiza las peticiones a través de él, y éste se las hace llegar al modelo. Además, es el encargado de renderizar la vista.

Como se puede observar, esta arquitectura dista de la original MVC, donde las interacciones son lineales de la forma Vista-Controlador-Modelo. En el caso de las aplicaciones web, el cliente realiza peticiones directamente al controlador, el cual redirige al modelo los cálculos pertinentes. La vista es quien se encarga de mostrar los resultados tras recibir los datos del modelo y ser redirija por el controlador hacia el cliente.

5.4 Configuración SSL/TLS

Definidos los requisitos y objetivos, que han dejado claras las funcionalidades, y marcando las pautas que se seguirán en el diseño, comienza la fase de implementación, donde el objetivo es crear un portal web sencillo, con registro y autenticación, donde posteriormente de realizar ambas, se abrirán las funcionalidades de la web, que no se desbloquearán hasta que el usuario verifique su correo electrónico, que será cuando entonces tenga habilitadas tanto la creación de certificados como la creación y verificación de firmas, además si lo desea, podrá descargar e instalar un certificado personal que lo autentique en la web.

Se ha decidido enfocar el análisis del sistema desde el prisma de Spring Security, dado las características singulares del proyecto, cuyas iteraciones principales no fluyen en un

modelo de dominio con muchas clases, sino que es un usuario interactuando de formas distintas con la configuración segura de un lugar web creado en Spring.

En primer lugar, la configuración básica y necesaria es habilitar la conexión segura, HTTPS. En Spring, esto se activa en el archivo *application.properties* ,donde se especifican los siguientes valores:

```
#Puerto donde se va a ejecutar el servidor
server.port=8088

#El formato del Keystore
server.ssl.key-store-type=PKCS12

#La localización del Keystore
server.ssl.key-store=src/main/resources/keystore.p12

#La contraseña del keystore
server.ssl.key-store-password=holahola

#Alias del certificado del servidor dentro del keystore
server.ssl.key-alias=tcert

#Activación de HTTPS
server.ssl.enabled=true
```

Figura 17: Configuración SSL/TLS en Spring

Un servidor Spring seguro necesita tener un Keystore. Un Keystore es un archivo que almacena claves criptográficas, certificados digitales y, a menudo, pares de claves públicas y privadas. En Spring se usa para almacenar el certificado SSL del servidor y su clave privada.

Dicho Keystore se ha creado mediante el siguiente comando de Keytool:

```
keytool -genkeypair -alias tcert -keyalg RSA -keysize 2048 -keystore
keystore.p12 -storetype PKCS12 -ext san=dns:localhost -ext
BasicConstraints:critical=ca:true
```

Este comando crea el fichero si no existe, genera un par de claves y un certificado autofirmado con un alias, *tcert*, con extensiones de *dns:localhost* para que el navegador lo reconozca como seguro y como Autoridad de Certificación para poder expedir otros certificados con este. El programa pide una contraseña y después de introducirla pide los campos del certificado (CN=TCertCA), y cuando son rellenados se genera el archivo.

Una vez que el keystore se ha creado y configurado en la aplicación, el servidor detectará automáticamente el archivo y utilizará el certificado SSL para cifrar las comunicaciones y autenticarse ante los clientes, pero al intentar conectarse por primera vez pasa esto:

Figura 18: Error de conexión no privada

Es lógico, pues, aunque se haya configurado el servidor como seguro, hace falta instalar el certificado, en el caso del proyecto al utilizar Chrome, en el Almacén de entidades raíz de confianza del almacén de certificados del sistema, por lo que extraemos el certificado de clave pública del Keystore:

```
keytool -exportcert -alias ec_tcert -keystore keystore.p12 -rfc -file ec_tcert.crt
```

Esto generará un archivo *.cer*, al que simplemente hay que hacer doble click para mostrar e instalarlo en el sistema. Es importante instalarlo en el almacén específico nombrado anteriormente, ya que es el almacén que utiliza el sistema para almacenar los certificados de los sitios web en los que confía. Esto hará que el cliente se pueda conectar a la web de forma segura.

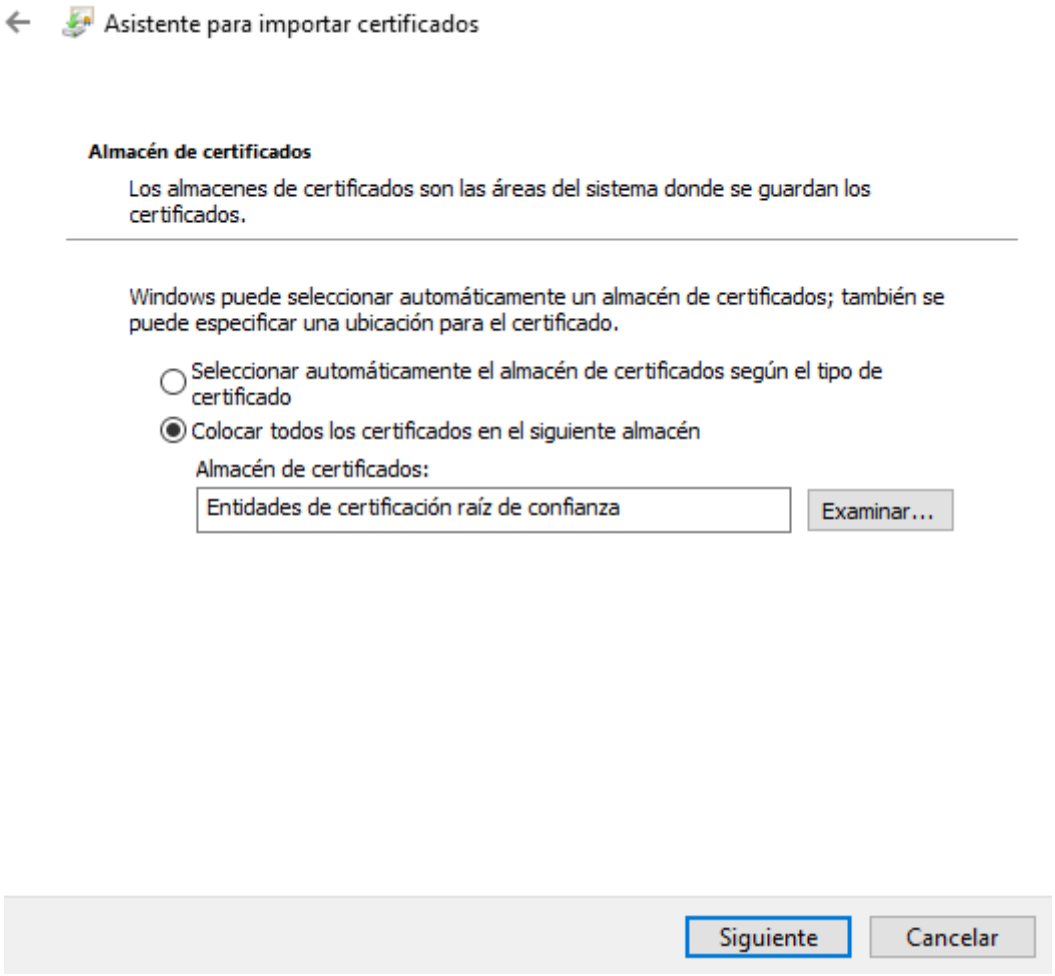


Figura 19: Instalación del certificado de clave pública

Y cuando se vuelve a reintentar la conexión:

Figura 20: Confirmación de lugar seguro

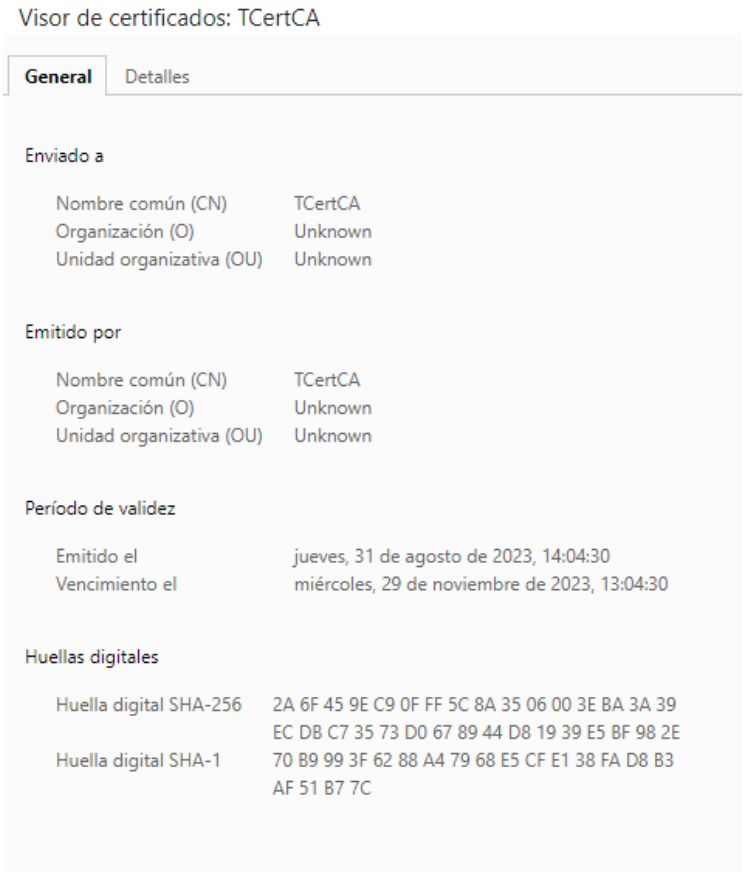


Figura 21: Certificado abierto por el navegador

Se puede comprobar que el certificado de la web es el creado previamente y que, al haber sido exportado e instalado en el almacén de certificados, el navegador ya confía en el sitio web y permite navegar con todas las funcionalidades de un sitio web seguro.

5.5 Configuración de Usuarios, Persistencia y Autenticación

Antes del diseño del modelo y las funcionalidades de la web, lo más importante es establecer la configuración de seguridad de Spring, que se encarga de implementar la autenticación, el encriptador de contraseñas para guardarlas de forma segura en las bases

```
@Entity
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true)
    private String email;
    private String nombre;
    private String contrasena;

    @ElementCollection(fetch = FetchType.EAGER)
    private List<String> roles;

    @Column(unique = true)
    private String codigo;

...
```

de datos, y el filtro de seguridad, el centro de la configuración en Spring. Cabe destacar que Spring gestiona todas las clases dependiendo de las etiquetas que se usen.
El primer paso es definir lo que es un Usuario en el sistema, esta clase será la única del modelo de dominio:

Los usuarios tendrán unas credenciales básicas, cuyo diferenciador será el email, y aparte una lista de roles asociados y el código que servirá para verificarlos. Con la definición de las propiedades, se encuentran las etiquetas de JPA (@), la tecnología que se va a usar para gestionar las bases de datos como ya se comentó anteriormente. Estas etiquetas definen los comportamientos que se desea que tengan las clases y sus propiedades respecto a su persistencia y características, a continuación, se van a explicar el porqué de su uso:

- **@Entity** sirve para marcar a la clase como un objeto que será persistente y monitorizable en el curso de vida del sistema.
- Se le ha asignado un **@Id** autogenerado a cada usuario para facilitar a Spring diferenciar los usuarios con enteros sencillos.
- **@Column(unique = true):** Esta anotación se utiliza para especificar que la columna correspondiente en la base de datos debe tener valores únicos. En este se aplica al mail, para que no se produzca más de un registro con correos diferentes, que podría causar problemas, de la misma forma que con el código de verificación
- **@ElementCollection(fetch = FetchType.EAGER):** Este atributo representa los roles asociados a un usuario. Puede contener múltiples roles, y la anotación anterior define cómo se deben cargar estos roles. En este caso, se cargan de manera inmediata (EAGER) cuando se recupera la entidad.

Una vez que se ha definido al Usuario, se crea el repositorio, que es la interfaz con la que se va a comunicar Spring con la base de datos:

```
@Repository
public interface UsuarioRepository extends JpaRepository<Usuario, Long> {

    Optional<Usuario> findByEmail(String email);

    Optional<Usuario> findByCodigo(String codigo);
}
```

A la interfaz se le etiqueta con **@Repository,** que indica a Spring que es un repositorio, en el cual hay Usuarios cuyo Id es un Long. Dentro del repositorio se definen las funciones, cuya funcionalidad implementa automáticamente Spring por el nombre, es decir, todas las funciones se nombran como *findBy…* seguido del nombre de una propiedad de la entidad del repositorio, en este caso Usuario, y JPA se encarga de forma automática de implementar estas operaciones para la base de datos. Con solo definirlas ya pueden usarse directamente. Devuelven objetos *Optional<>*, que guardan los valores obtenidos de las búsquedas y permiten realizar operaciones para comprobar si hay valores, si están vacíos, etc.

En esta implementación sólo se necesitarán dos operaciones: encontrar a un Usuario por email o por nombre. En este punto ya están definidos los Usuarios y su persistencia junto con sus operaciones. En la creación del proyecto se eligió **H2** como tecnología de base de datos en memoria, ideal para entornos de desarrollo por su versatilidad, control y facilidad de acceso. Es una base de datos volátil, pero en Spring con un simple cambio de

módulo de dependencias, se podría cambiar la base de datos para tener una persistente y no habría que realizar modificación alguna.

El último paso para implementar la autenticación es crear la configuración de seguridad, una clase que la defina con las etiquetas **@Configuration** y **@EnableWebSecurity:**

```
@Configuration
@EnableWebSecurity
public class ConfiguracionSeguridad {

  @Autowired
  private UserDetailsService userDetailsService;

  @Autowired
  public void configure(AuthenticationManagerBuilder auth) throws
Exception {

auth.userDetailsService(userDetailsService).passwordEncoder(passwordE
ncoder());
  }

  @Bean
  public static PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
  }

  @Bean
  public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception
...
}
```

Antes de seguir, el **@Autowired** es una etiqueta que hace que Spring inyecte las dependencias necesarias e instancie la clase etiquetada, y los **@Bean** sirven para etiquetar miembros de una configuración.

Tiene 3 funciones principales:

- **configure(AuthenticationManagerBuilder auth):** Configura cómo Spring Security maneja la autenticación de usuarios, como autenticación en memoria o con una base de datos, en este caso con base de datos.

  UserDetailsService es una implementación de una interfaz **UserDetails** que maneja la autenticación cuando se reciben los parámetros a la llamada */login* (preestablecida por defecto en Spring Security), busca al usuario si existe en la

```
@Service
public class UsersDetailsServiceImpl implements UserDetailsService {

    @Autowired
    private UsuarioRepository ur;

    @Override
    public UserDetails loadUserByUsername(String email) throws
UsernameNotFoundException {

        Optional<Usuario> u = ur.findByEmail(email);

        if (!u.isPresent()) {
            throw new UsernameNotFoundException(email);
        }
        return new UsuarioDetails(u.get());
    }
}
```

base de datos, autentica y crea una sesión para el que haya introducido sus credenciales a esta llamada o que se haya autenticado con certificado.

Cabe destacar la elección de que para iniciar sesión se requiera utilizar como credencial el email en vez de el nombre de usuario, ya que es único. En este sistema los usuarios inician sesión con el email.

- **passwordEncoder():** Proporciona un codificador de contraseñas para cifrar y descifrar contraseñas de usuarios.

- **filterChain(HttpSecurity http):** Define las reglas de seguridad para proteger rutas de la aplicación, especificando quién puede acceder a ellas y quién necesita autenticación. Aquí es donde van las reglas de acceso de toda la aplicación, que se dividen en tres:

    - **permitAll():** permitir a cualquiera el acceso a esa URL
    - **authenticated():** necesita estar autenticado
    - **hasRole("X"):** si tiene el rol X puede ir a la URL

Se utiliza *requestMatchers()* para especificar la URL y el permiso, y se configura

```
        http.authorizeHttpRequests(authorizeHttpRequests->
        authorizeHttpRequests
             .requestMatchers("/verificar/**").permitAll()
             .requestMatchers("/certificados/**").hasRole("VERIFIEDUSER")
             .requestMatchers("/crear-
    certificado/**").hasRole("VERIFIEDUSER")
             .requestMatchers("/firmas").hasRole("VERIFIEDUSER")
             .requestMatchers("/crear-firma").hasRole("VERIFIEDUSER")
             .requestMatchers("/comprobar-firma").hasRole("VERIFIEDUSER")
             .requestMatchers("/acceso-denegado/**").hasRole("USER")
             .requestMatchers("/registro/**").permitAll()
             .requestMatchers("/login").permitAll()
             .requestMatchers("/").permitAll()
             .anyRequest().authenticated())
        ...
```

como en fases previas se ha indicado el acceso:

anyRequest() es la regla maestra que afecta a todas menos a las de *permitAll()*, por lo que en las de *hasRole()*, también hay que estar autenticado.

Las reglas restantes configuran el formulario de inicio de sesión con credenciales, el cual define la URL que devuelve el formulario de inicio de sesión para completarlo y la URL a la que ir cuando se completa, la autenticación con certificado, indicando que se toma el CN como nombre de inicio de sesión, donde va a estar el email, ya que como hemos visto antes en UserDetails, en la base de datos se busca por email, y por último una gestión de eventos, en la cual cada vez que un usuario intente acceder a un lugar al que no tiene permisos, se le

redireccione a una página que le avise de su error. Está implementado en este sistema para los usuarios que todavía no hayan verificado su cuenta desde el correo enviado a su email personal, los cuales no tienen permiso para acceder a ciertas funcionalidades del sistema.

```
        .formLogin(formLogin -> formLogin
                .loginPage("/login")
                .defaultSuccessUrl("/inicio")
                .permitAll())
              .x509(x509 -> x509
                .userDetailsService(userDetailsService)
                .subjectPrincipalRegex("CN=(.*?)(?:,|$)"))
                .exceptionHandling(exceptionHandling                    ->
       exceptionHandling.accessDeniedPage("/acceso-denegado"));
```

Con esta configuración básica se activa la autenticación de los usuarios, su persistencia en la base de datos y los permisos que tienen para acceder a cada parte de la web según su estado.

Para finalizar esta configuración y asegurarse de que la autenticación con certificado funcione con los certificados expedidos por el certificado de la web, hay que configurar un **Truststore** en el *application.properties*.

```
#Se indica localización y contraseña del truststore y se activa
la autenticación por medio de SSL/TLS, es decir, con certificado
e intercambio de claves
server.ssl.trust-store=src/main/resources/truststore.jks
server.ssl.trust-store-password=holahola
```

```
server.ssl.client-auth=want
```

Un Truststore es un repositorio de confianza utilizado que contiene los certificados en los que se quiere confiar en el sistema. Estos certificados se utilizan para verificar la autenticidad de otras partes en una comunicación segura. Se crea igual que un Keystore, pero el formato es JKS, específico de Java, en vez de en PKCS12.

Cuando un sistema se comunica con otro sistema de forma segura (por ejemplo, a través de HTTPS), verifica el certificado del otro sistema con la ayuda del Truststore. Si el certificado se encuentra en el Truststore y es válido, se considera que la comunicación es segura y se establece la confianza. Si el certificado no se encuentra en el Truststore o no es válido, la comunicación se considera no confiable y puede ser rechazada. En este Truststore sólo estará el certificado del sistema, el que identifica al sitio y emite los certificados personales, para que estos sean aceptados por el sistema cuando se intenten autenticar contra el servidor.

5.6 Modelo

En la memoria se nombrarán por encima las clases que intervienen en cada parte del patrón arquitectónico, la especificación y explicación del código estará en el *Anexo III*.

El modelo es la parte del proyecto en la que se realizan las operaciones específicas sobre los datos y sobre el modelo de dominio, eso engloba a la clase Usuario, ya comentada en la parte previa. Aparte de esta, se encuentran todos los servicios creados, marcados con la etiqueta **@Service**, cada uno para un fin específico, en los que se llevan a cabo las operaciones esenciales:

- **CertificadoService.java:** En esta clase se gestiona la creación y gestión de los certificados personales.

- **ClaverService.java:** En esta clase se ha implementado una interfaz sencilla para crear pares de claves

- **EmailService.java:** Esta clase implementa la operación de envío del correo de verificación.

- **FirmaService.java:** Esta clase implementa las operaciones relacionadas con crear y verificar firmas digitales.

- **GestorAlmacenesService.java:** Una interfaz que simplifica la gestión de los KeyStore de Java de los Usuarios.

- **LeerPKCSService.java:** Lee los certificados y claves del sistema necesarios para firmar.

- **UsuarioService.java:** La clase UsuarioService se encargaría de la gestión de usuarios en el sistema. Esto incluye operaciones de registro, inicio de sesión, y otras tareas relacionadas con la autenticación y la autorización de usuarios en la aplicación.

5.7 Controlador

Marcados con la etiqueta **@Controller**, son las clases que median entre el cliente, modelo y vista, enviando datos al modelo y renderizando las vistas cuando acaban las operaciones:

- **AlmacenController:** Gestiona las peticiones relacionadas con las peticiones sobre los certificados.
- **FirmaController:** Gestiona las peticiones relacionadas con la creación y verificación de firmas.

- **UsuarioController:** Gestiona las peticiones del usuario, como el registro, login, verificaciones…

5.8 Vista

La vista es el monitor de la aplicación, esta está gestionada por Thymeleaf, recibe los datos del modelo y la renderiza el controlador y responde las peticiones de formulario que realiza el cliente, se utiliza la etiqueta *@GetMapping* para mapear las peticiones desde el servidor, cada una devuelve su respectiva vista, cabe destacar que durante todas las vistas hay una barra de navegación, que cambia según si el usuario está autenticado o no:

No autenticado:

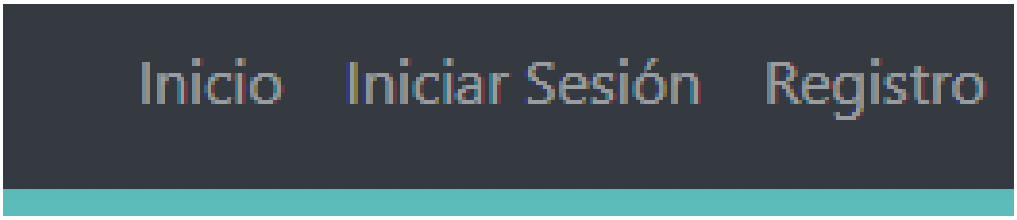


Figura 22: Navegador de cliente sin autenticar

Autenticado:

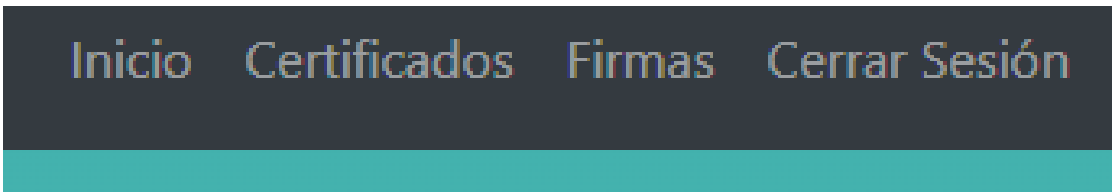


Figura 23: Navegador de cliente autenticado

- **@GetMapping("/")**

Figura 24: Página de bienvenida de la web

Esta es la pantalla de inicio, donde se muestra información sobre la configuración web y da la bienvenida a cualquier usuario.

- **@GetMapping("/login")**

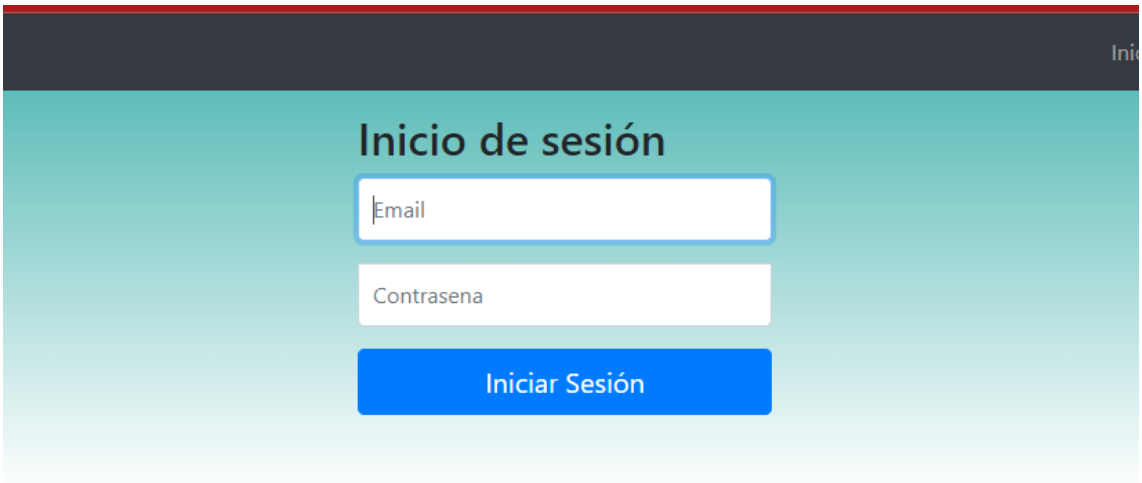


Figura 25: Página de inicio de sesión

En la pestaña iniciar sesión se encuentra el formulario para acceder al sistema a través de las credenciales.

- **@GetMapping("/registro")**

Figura 26: Página de registro

El registro es similar a iniciar sesión, tras completarlo, el usuario se registra en el sistema y se le envía un correo de verificación para verificarse en la web.

- **@GetMapping ("/inicio")**

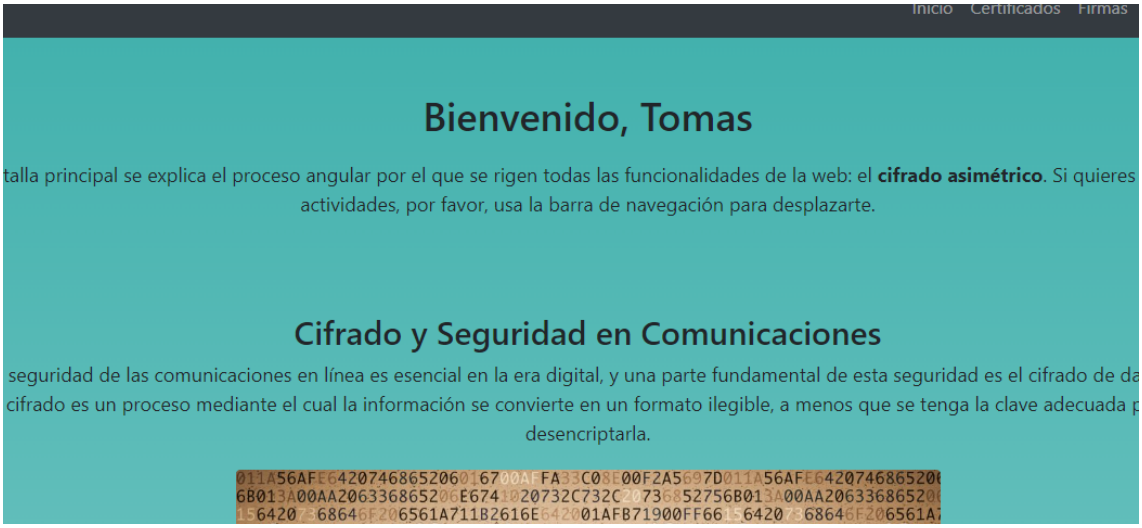


Figura 27: Página de inicio del usuario

Página de inicio del usuario cuando se autentica, donde se le ofrece la opción de navegar y una explicación básica del cifrado simétrico y asimétrico.

- **@GetMapping ("/crear-certificado ")**

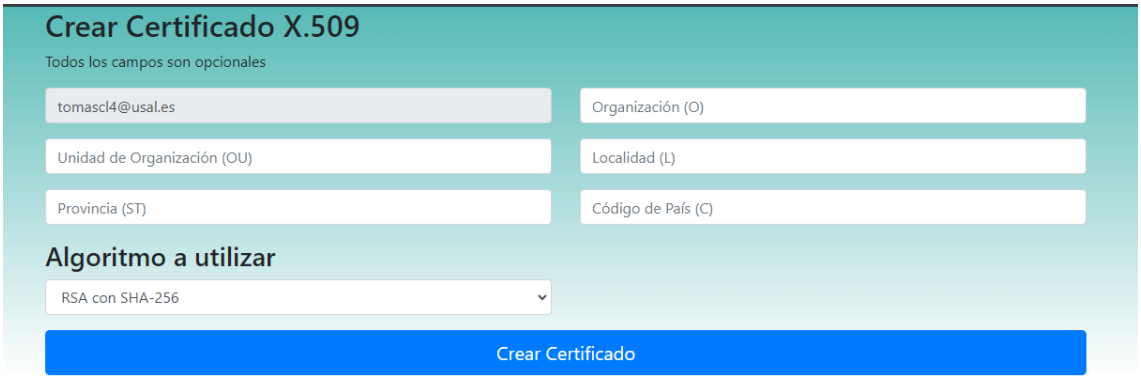


Figura 28: Página de creación de certificado personal

Al seleccionar “Crear Certificado” se redirige a la página con un formulario para personalizar el certificado, tanto los datos como el algoritmo (El CN debe ser el email, pues se utiliza para autenticarse).

- **@GetMapping ("/certificados-usuario")**

Figura 29: Página de certificados del usuario

En la vista de certificados hay una pequeña explicación sobre los mismos, un botón para ir a la página donde se crean y abajo del todo un listado donde aparecerán los certificados creados por el usuario.

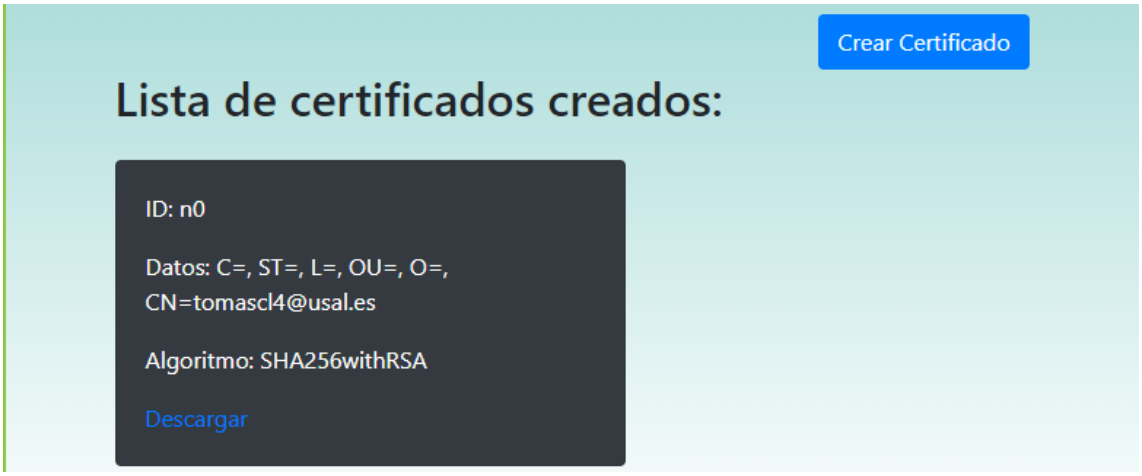


Figura 30: Certificado en la lista del usuario

- **@GetMapping ("/acceso-denegado")**

Figura 31: Redirección a acceso denegado

Si no se está verificado y se intenta navegar, aparecerá este mensaje.

- **@GetMapping ("/firmas")**

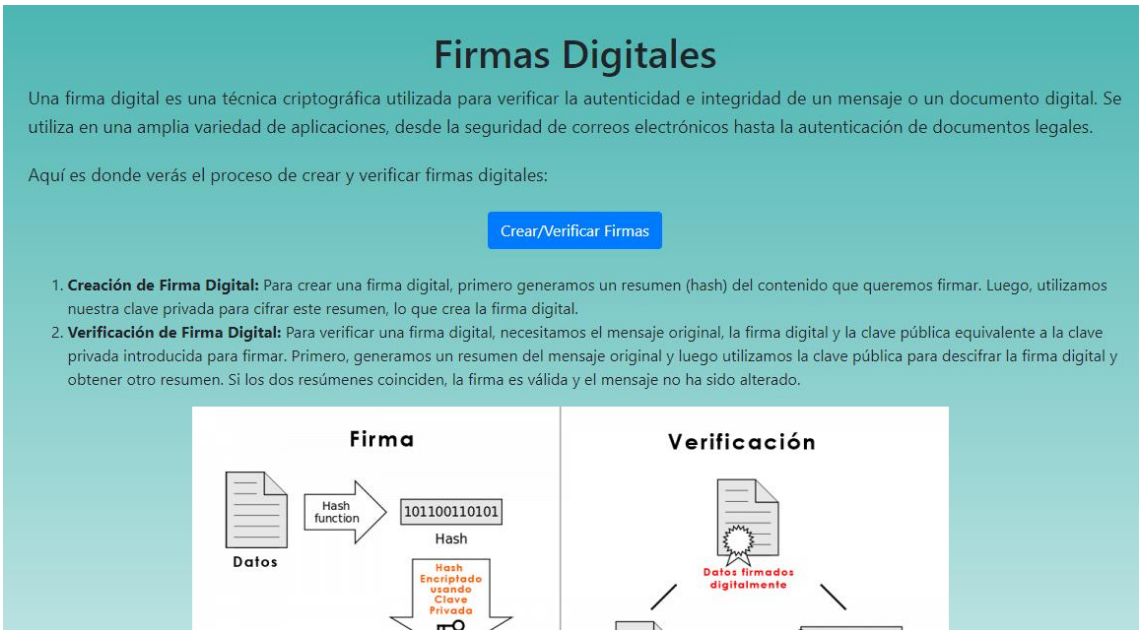


Figura 32: Página principal de firmas

Al ir hacia firmas, se muestra una vista donde también hay una pequeña explicación sobre la creación y verificación de firmas. El botón lleva a la vista donde se realizan estas funciones nombradas.

- **@GetMapping ("/crear-verificar-firmas")**

Figura 33: Formulario de creación de firmas

Figura 34: Formulario de verificación de firmas

En esta vista están ambas funciones implementadas. En la primera al introducir los datos se generará el mensaje firmado de esta manera tras introducir los datos y pulsar firmar:

Figura 35: Resultado de la firma

En la segunda función, tras introducir los datos y comprobar, aparecerá un popup que nos confirmará si la verificación es correcta o ha fallado;

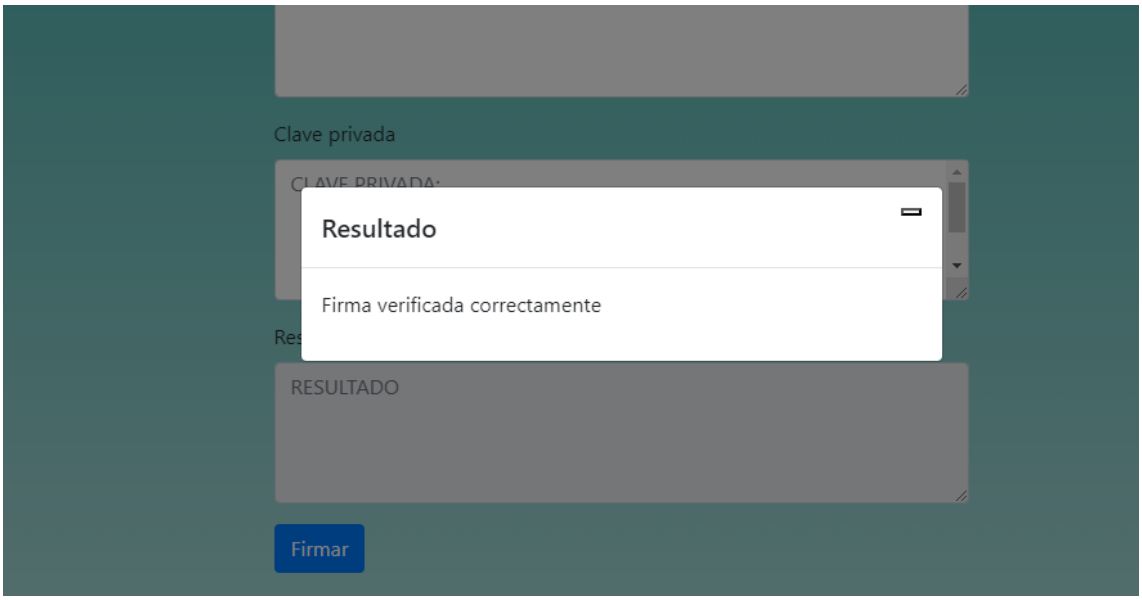


Figura 36: Aviso de firma verificada correctamente

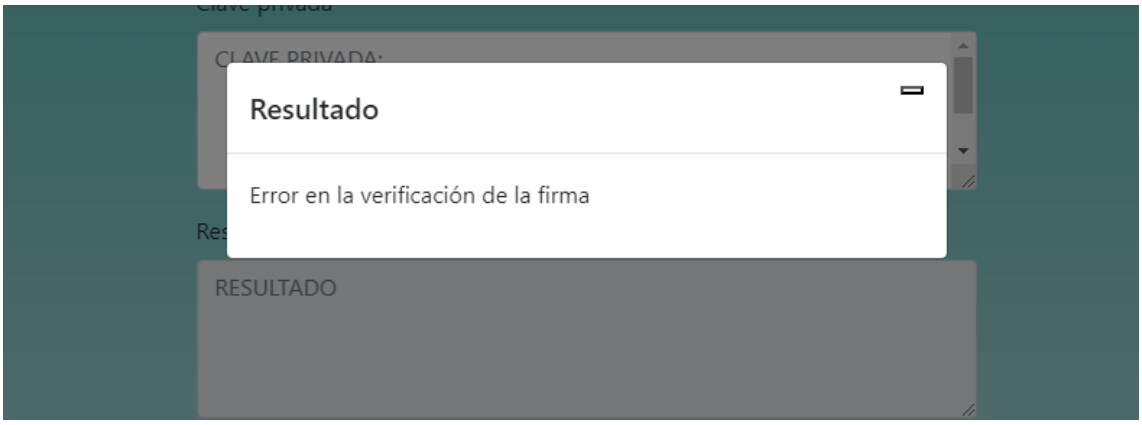


Figura 37: Aviso de fallo en la verificación de la firma

6. Conclusiones y líneas de trabajo futuras

6.1 Conclusiones

Tras la realización del trabajo se han llegado a una serie de conclusiones, fruto de la experiencia ganada en el desarrollo de este.

La importancia de la seguridad digital es primordial hoy en día, está presente en todos los aspectos tecnológicos y es muy importante que se respeten sus procedimientos y se tomen todas las precauciones posibles para proteger la información., además, el uso básico de herramientas que son totalmente eficientes hoy en día, como las firmas y los certificados, no es una tarea difícil de aprender, pero es algo que no mucha gente conoce y que va a ser cada día más vital en un mundo más digitalizado, como se ha visto por ejemplo con las nuevas implementaciones de DNIs electrónicos, por lo que su divulgación es una tarea muy adecuada y fácil de realizar.

Desde el punto de vista didáctico del trabajo, se ha valorado también el hecho de que el aprendizaje es más ameno y eficiente si se acompañan de ejemplos interactivos, en los que el usuario realiza el proceso mientras lo aprende, asimilando mucho más rápido los conceptos al aprender con la práctica.

Desde un punto de vista técnico, el uso de Spring se ha considerado un acierto por todas las características que ofrece: la facilidad para su configuración, por ejemplo con los módulos de JPA y Security, que facilitan enormemente el desarrollo y ahorran tiempo al programador en tareas repetitivas y ya resueltas, su fácil adaptación y cambios de módulos, que hace que tenga un enorme potencial para la escalabilidad y la actualización constante, y todo el soporte que ofrece gracias a la enorme comunidad que lo mantiene hacen de Spring una de las opciones más potentes para desarrollar entornos web seguros y fáciles de mantener.

Asimismo, se ha valorado la importancia que tiene la separación de responsabilidades de las clases, haciendo fácil y comprensible tanto la creación como la depuración del proyecto, desde la separación en términos de código, clases y funciones, como la separación en el proceso de desarrollo. La adopción de arquitecturas de diseño

establecidas simplifica el desarrollo en etapas posteriores, mejorando la legibilidad del código y facilitando la implementación de mejoras o modificaciones. Aunque en las fases iniciales pueda parecer un gasto de tiempo aparentemente innecesario, en proyectos más grandes resulta altamente beneficioso.

Por último, cabe destacar la importancia en la planificación del tiempo, que sirve como medida paliativa frente a los imprevistos que siempre suelen surgir.

6.2 Líneas de trabajo futuras

En cuanto a las líneas de trabajo futuras:

- **Interfaces de Usuario:** Explorar la posibilidad de mejorar la interfaz de usuario para hacer que la aplicación sea más intuitiva y fácil de usar, lo que podría aumentar su adopción por parte de los usuarios y la mayor compresión de los temas a enseñar.

- **Ampliación de Algoritmos Criptográficos:** Investigar y adoptar algoritmos criptográficos más avanzados y seguros a medida que evolucionen las tecnologías.

- **Auditorías de Seguridad:** Realizar auditorías de seguridad periódicas para identificar posibles vulnerabilidades y debilidades en la aplicación y tomar medidas proactivas para corregirlas.

- **Educación y Concienciación en Seguridad:** Impartir capacitación y concienciación en seguridad digital a los usuarios finales y al personal involucrado en la emisión y verificación de certificados y firmas digitales.

- **Criptografía Post-Cuántica:** La evolución de la computación cuántica pone en jaque a todos los algoritmos criptográficos creídos indescifrables hoy en día. La actualización y búsqueda de nuevos paradigmas para prevenir la nueva escala de fuerza bruta cuántica es una de las puntas de lanza en el estudio de la seguridad en el futuro.

- **Mejoras en la Escalabilidad:** Evaluar la capacidad de la aplicación para manejar un aumento en la carga de trabajo y considerar mejoras en la escalabilidad, especialmente si se espera un mayor uso.

- **Colaboración con Organismos de Certificación**: Explorar asociaciones con organismos de certificación reconocidos para mejorar la validez y la confianza de los certificados emitidos.

Referencias

Anónimo. (s.f.). Obtenido de https://www.techtarget.com/searchapparchitecture/definition/Spring-Framework

Cloudflare. (s.f.). Obtenido de https://www.cloudflare.com/es-es/learning/ssl/what-is-https/#:~:text=El%20protocolo%20de%20transferencia%20de,de%20las%20transferencias%20de%20datos.

Gaetanino Paolone, R. P. (s.f.). Empirical Assessment of the Quality of MVC Web ApplicationsReturned by xGenerator. *MDPI computers*.

Instituto Nacional de Estadística. (s.f.). Obtenido de https://www.ine.es/ss/Satellite?c=Page&cid=1254734731861&lang=es_ES&pagename=SedeElectronica%2FSELayout

Ministerio de Justicia del Gobierno de España. (s.f.). Obtenido de https://www.mjusticia.gob.es/es/Ciudadano/Registros/Documents/1292428778575-CODIGOS_ISO_3166_1.PDF

Thymeleaf. (s.f.). Obtenido de https://www.thymeleaf.org/

Wikipedia. (s.f.). Obtenido de https://es.wikipedia.org/wiki/X.509

Wikipedia. (s.f.). *Wikipedia*. Obtenido de https://es.wikipedia.org/wiki/Criptograf%C3%ADa_asim%C3%A9trica