Desarrollo de un sistema inteligente para la gestión y explotación de información procedente de documentos históricos

TRABAJO FIN DE GRADO

CURSO 2024-2025

Autor

Juan José Encina Fernández

Director

Antonio Sarasa Cabezuelo

Francisco Cebreiro Ares

GRADO EN INGENIERÍA DEL SOFTWARE

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

Calificación final: 10

Development of an intelligent system for the management and exploitation of information from historical documents

FINAL DEGREE PROJECT

CURSO 2024-2025

Autor

Juan José Encina Fernández

Director

Antonio Sarasa Cabezuelo

Francisco Cebreiro Ares

DEGREE IN SOFTWARE ENGINEERING

FACULTY OF COMPUTER SCIENCE

COMPLUTENSE UNIVERSITY OF MADRID

DEDICATORIA

A mi familia y a mis amigos,

por acompañarme en todo el camino.

AGRADECIMIENTOS

Me gustaría agradecer a mis directores, Antonio Sarasa Cabezuelo y Francisco Cebreiro Ares, por su guía y apoyo durante todo el proceso. A mi familia y amigos por su apoyo constante y, a Ye Fan por tantas horas de trabajo compartidas.

Resumen

Este Trabajo de Fin de Grado presenta el desarrollo de una aplicación web para la digitalización, gestión y análisis de protestos de letras de cambio, documentos notariales que certifican el impago de una letra de cambio y recopilan toda la información asociada a esta. Los protestos son fundamentales para la investigación de la crisis económica acontecida en toda Europa, por motivo de la Guerra de la Convención, declarada en 1793 tras la Revolución francesa. El proyecto surge a partir de la necesidad del profesor de Historia Francisco Cebreiro Ares, quien dispone de miles de protestos en formato papel y requería una herramienta que permitiera estructurar la información en una base de datos, acelerar el proceso de digitalización y facilitar su explotación con fines de investigar está importante crisis económica. Como solución se implementó una aplicación web con arquitectura en capas, una lógica en Node.js con Express, una interfaz de usuario construida en EJS y Bootstrap, y una base de datos relacional en MySQL. Todo ello desplegado mediante contenedores Docker sobre Google Cloud y complementado con librerías de visualización que permiten analizar los datos en gráficos, mapas y tablas interactivas.

Palabras clave

Protestos, letras de cambio, Endoso, digitalización, aplicación web, base de datos relacional, análisis de datos, Node.js, MySQL, Docker

Abstract

This Final Degree Project describes the development of a web application for the digitization, management, and analysis of historical documents known as protestos, which are crucial for studying the economic crisis that struck Europe at the end of the eighteenth century as a consequence of the War of the Convention, just after the French Revolution. The system supports the rapid and organized entry of large volumes of information, stores them in a relational database, and enables their exploration through search tools, charts, tables, interactive maps, and exports to external analysis platforms.

The application was designed in response to the needs of Professor Francisco Cebreiro Ares (Economic History), who possesses thousands of paper-based protestos and required a tool to accelerate both digitization and data exploitation. To that end, we implemented a layered web architecture: Node.js and Express on the backend; EJS and Bootstrap for the user interface; and MySQL as the persistence layer—everything deployed as Docker containers on Google Cloud.

Keywords

Protestos, bills of exchange, digitization, Economic History, web application, relational database, data analysis, Node.js, MySQL, Docker

ÍNDICE DE CONTENIDOS

Índice de figuras

Capítulo 1 - Introducción

Este Trabajo de Fin de Grado aborda el diseño y desarrollo de una aplicación web para la digitalización, gestión y análisis de protestos de letras de cambio, una fuente imprescindible para estudiar las redes de crédito y la crisis económica en la Europa de finales del siglo XVIII asociada a la Primera Guerra de Coalición (1792–1797), un conflicto que enfrentó a la recién fundada República Francesa contra gran parte del resto de países europeos cortando repentinamente los lazos comerciales y económicos del país europeo más poblado e importante del momento.

Para analizar el grado de destrucción económica generada resultan muy valiosos unos documentos denominados protestos. El protesto es un documento notarial que certifica el impago de una o varias letras de cambio e incluye, además, los endosos de cada una de esas letras, y la identificación de los agentes implicados. En la época las letras de cambio eran el principal mecanismo financiero en Europa, y eran usadas como fuente de financiamiento para los comerciantes y productores o forma de inversión para banqueros e inversores mediante la compra-venta de las letras en los documentos denominados endosos.

La importancia de los protestos, almacenados por bancos y notarios, se encuentra en la gran cantidad de información que contienen los protestos en contraste con la escasa información financiera extraída en la época. Gracias a ellos es posible analizar de forma detallada las causas de la crisis y sus efectos diferenciados por regiones, personas y periodos temporales.

El proyecto responde a una necesidad planteada por el profesor Francisco Cebreiro Ares, quien dispone de miles de protestos originales en formato papel, obtenidos de diversos archivos de toda España, con los que está realizando una investigación sobre esta influyente esta crisis financiera y económica europea. Dada la magnitud y heterogeneidad del material, se propone un sistema que organice la información estructuradamente, que facilite la introducción rápida y fiable de los

datos, permita realizar consultas complejas y habilite un análisis visual mediante gráficos, tablas, mapas y grafos, además de incorporar mecanismos de exportación hacia herramientas externas.

Desde un punto de vista técnico, la aplicación se ha desarrollado siguiendo una arquitectura en capas. El backend se implementó en Node.js con Express, mientras que la interfaz de usuario se construyó con EJS y Bootstrap. Para la estructuración de la información se empleó una base de datos relacional MySQL, y el despliegue se realizó mediante contenedores Docker sobre la plataforma Google Cloud. Para el análisis de datos se han utilizado Chart.js, D3.js y Leaflet.

El proyecto busca, por un lado, ofrecer al investigador un método para almacenar y gestionar la información de forma estructurada y fiable, y que además facilite y acelere el proceso de digitalización de los protestos manuscritos. Y por otro, proporcionar una herramienta que permita realizar análisis avanzados de la información extraída.

1.1 Motivación

La motivación principal surge de la investigación del profesor Francisco Cebreiro Ares sobre la crisis económica y fiscal que afectó a Europa tras el estallido de la Primera Guerra de Coalición en 1792. Este conflicto, que enfrentó a la recién fundada Francia revolucionaria contra gran parte de las potencias europeas, provocó un colapso repentino del comercio y una ruptura del sistema financiero europeo. Para analizar las causas y los efectos de esta crisis en el caso español, el profesor Cebreiro ha reunido varios miles de protestos conservados en archivos de todo el país. Estos documentos, que certifican letras de cambio impagadas junto con sus endosos, resultan de gran valor porque en el siglo XVIII las letras de cambio constituían el principal mecanismo financiero, comparable hoy en día al sistema de préstamos bancarios o incluso las acciones de empresas en el mercado bursátil.

Francisco se enfrentó a la tarea de estructurar los protestos, que son documentos en ocasiones heterogéneos y manuscritos, lo que dificulta su lectura y homogeneización. Por otro lado, la digitalización manual resultaba poco viable, Francisco estimó que transcribir cada protesto en una hoja de cálculo requeriría unos diez minutos, lo que multiplicado por los miles de protestos a los que tiene acceso supondría varios meses de trabajo dedicados exclusivamente a la digitalización. Por último, aunque existen herramientas potentes para el análisis de bases de datos, no ofrecían un entorno adaptado a las particularidades de los protestos ni a las preguntas de investigación planteadas.

En este contexto, y con la dirección del profesor Antonio Sarasa Cabezuelo, asumimos el reto de comprender las necesidades de Francisco Cebreiro y proponer una solución que agilizara la digitalización y explotación de los datos.

1.2 Explicación de datos estudiados en la aplicación

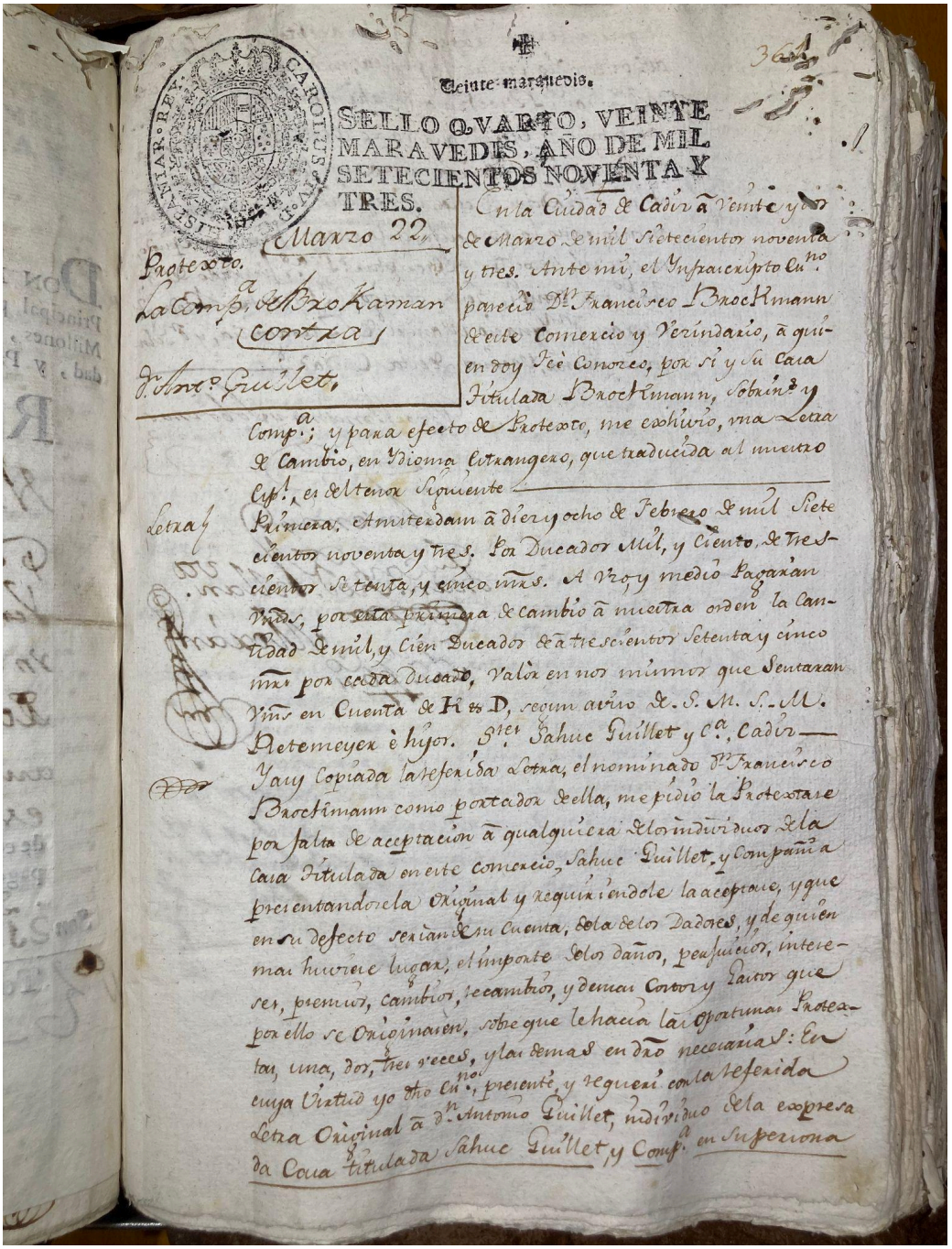
Entender los documentos es la base del proyecto, ya que de estructurar bien los datos dependen tanto la aplicación como el análisis de los datos. Por ello es necesario explicar brevemente los tres elementos principales del sistema, los protestos, las letras de cambio y los endosos[1]:

Protestos

El protesto es un documento notarial que se levanta cuando una o varias letras de cambios no son pagadas en la fecha de vencimiento. Se trata de una prueba oficial del impago, donde el notario transcribe íntegramente la o las letras (con sus endosos, marcas y anotaciones), recoge la reclamación formal y anota los motivos alegados por el deudor.

En el contexto histórico, el protesto es una fuente de gran valor, ya que contiene el contenido completo de las letras originales y estos endosos han sido mejor conservados que las letras de cambio, la identidad de los participantes, los lugares y las cantidades implicadas… Además, los protestos generan información adicional, como los costes derivados del impago (intereses, comisiones, gastos

notariales) y en ocasiones los motivos del impago. Estos son especialmente importantes porque aportan información acerca de las causas de la crisis y el gran número de protestos finales del siglo XVIII.



Veinte marauedis.

SELLO QVARTO, VEINTE MARAVEDIS, AÑO DE MIL SETECIENTOS NOVENTA Y TRES.

Marzo 22,

Protesto.

La Comp.a de Brockman contra

Dn. Ant.o Quillet,

En la Ciudad de Cadiz à Veinte y dos de Marzo de mil Setecientos noventa y tres. Ante mi, el Yn.frascripto Esc.no parecio Dn. Francisco Brockmann de este Comercio y Vecindario, à quien doy Fee Conozco, por si y su Casa titulada Brockmann, Sobrin.e y Comp.a; y para efecto de Protesto, me exhivio, vna Letra de Cambio, en Idioma Estrangero, que traducida al nuestro Cast.no, es del tenor siguiente

Letra primera. Amsterdam à dies y ocho de Febrero de mil Siete cientos noventa y tres. Por Ducador Mil, y Ciento, de tres cientos Setenta, y cinco mrs. A Vso, y medio Pagaran Vras, por esta primera de Cambio à nuestra orden la Cantidad de mil, y Cien Ducador de à trescientos Setenta y cinco mrs por cada Ducado, Valor en nro mismo que Sentaran Vras en Cuenta de H.os D, segun aviso de S. M. S. M. Detemeyer è hijos. S.res Sahuc Quillet y C.a, Cadiz—

Y aui Copiada la referida Letra, el nominado Dn. Francisco Brockmann como portador della, me pidio la Protestare por falta de aceptacion à qualquiera de los individuos de la Casa titulada en este comercio, Sahuc Quillet, y Compañia presentandola Original y requiriendole la aceptase, y que en su defecto seria de su Cuenta, de la de los Dadores, y de quien mas hubiese lugar, el importe de los daños, perjuicios, intereses, premios, cambios, recambio, y demas Costos y Daños que por ello se Originasen, sobre que le hacia las Oportunas Protestas, una, dos, tres veces, y las demas en dro necesaria: En cuya Virtud yo dho Esc.no, presente, y requeri con la referida Letra Original à Dn. Antonio Quillet, individuo de la expresada Casa titulada Sahuc Quillet, y Comp.a en Superiona

Figura 1 – Imagen protesto original del 1793. Archivo Histórico Provincial de Cádiz, protocolos de la ciudad de Cádiz.

Letras de cambio

La letra de cambio es uno de los instrumentos financieros más antiguos e importantes en la historia financiera y comercial europea. Se trata de un documento mediante el cual una persona (llamada librador) ordena a otra (el librado) que pague una cantidad determinada de dinero a un tercero (el tomador o beneficiario), en una fecha y lugar concretos. Su origen se encuentra en los siglos medievales, cuando el transporte de metálico a largas distancias resultaba costoso y arriesgado. La letra ofrecía una solución: facilitaba los pagos internacionales y, además, permitía sortear restricciones legales sobre la usura al encubrir intereses en los cambios de moneda o en los plazos de pago.

La letra de cambio no sólo resolvía un problema logístico, sino que también se convirtió en un mecanismo de crédito y en un activo financiero que circulaba entre distintos agentes, llegando a ser central en los mercados monetarios europeos hasta el siglo XIX.

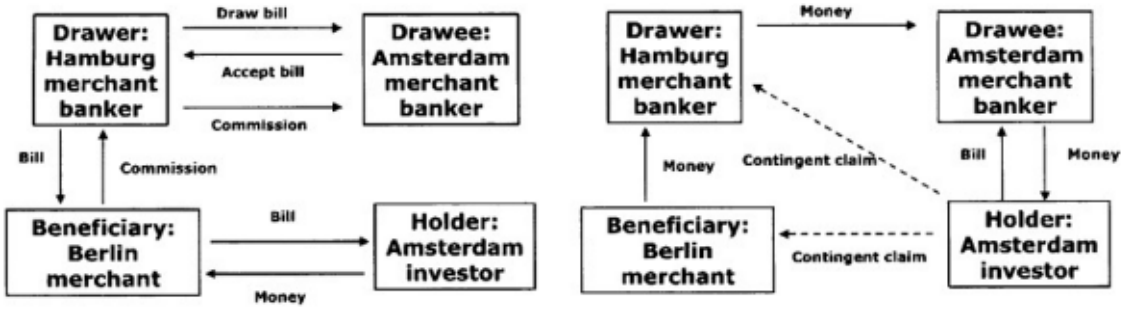


Figura 2 – Diagrama explicativo letra de cambio

Endosos

El endoso es la cesión (venta del endoso) que hace el tenedor de una letra de cambio a favor de un nuevo acreedor (comprador del endoso). Se anota en el dorso del documento y convierte al endosado en el nuevo beneficiario, que adquiere todos los derechos de cobro y las obligaciones que hubiera. Gracias al endoso, una misma letra podía circular repetidamente en distintas transacciones, convirtiéndose en un medio de pago transferible y otorgándole liquidez. En algunos casos, se añadían tantos endosos que era necesario adjuntar nuevas hojas para seguir transmitiendo el documento.

Este mecanismo amplió notablemente la funcionalidad de la letra, pues la transformó en un instrumento negociable que podía servir para saldar deudas, financiar operaciones comerciales o incluso especular.

En este sistema, cada endosante respondía solidariamente del pago en caso de impago final. Esto implicaba que, si el último deudor no podía satisfacer la letra, el tenedor podía reclamar a cualquiera de los endosantes anteriores. Este mecanismo reforzaba la confianza en la letra como medio de pago, pero también tenía un efecto potencialmente negativo en situaciones de crisis: un solo impago podía trasladar la carga a toda la cadena de endosantes, provocando pagos o incluso quiebras en cascada. De esta forma, el incumplimiento de algunos agentes podía generar crisis de liquidez, al verse obligados otros comerciantes o banqueros a disponer de efectivo de forma inmediata para pagar su deudas, y, en casos extremos, derivar en espirales deflacionarias. En esos casos, los comerciantes ante la necesidad de dinero de inmediato para pagar sus deudas bajaban los precios de sus mercancías, lo que bajaba sus beneficios y reducía el valor de sus mercancías y con ello el volumen del comercio. Como el pago de las letras dependía en última instancia de la actividad mercantil, esta contracción reducía la capacidad de pedir nuevo crédito y de satisfacer el existente, intensificando el círculo vicioso [1][5]

1.3 Objetivos

Durante el desarrollo de este Trabajo de Fin de Grado se plantearon inicialmente una serie de objetivos generales en las primeras reuniones. A medida que avanzó la implementación de la aplicación, dichos objetivos fueron reevaluados y ajustados en función de la viabilidad y de las prioridades. Estos objetivos se definieron a partir de las necesidades planteadas por el profesor Francisco Cebreiro Ares y de las propuestas de implementación del profesor Antonio Sarasa Cabezuelo y el alumno Juan José Encina Fernández.

1.3.1 Objetivo general

El objetivo principal de este Trabajo de Fin de Grado ha sido diseñar e implementar una aplicación web que permita la digitalización manual, la elaboración de una base de datos estructurada que refleje los documentos a estudiar y el análisis de protestos de letras de cambio de finales del siglo XVIII, con el fin de facilitar su explotación como fuente documental en investigaciones de Historia Económica.

1.3.2 Objetivos específicos

1. **Modelado de datos:**

   Construir una base de datos relacional que represente de manera precisa las entidades involucradas (protestos, letras de cambio, endosos, personas, roles, ciudades, monedas y tipos), garantizando la integridad y consistencia de los registros mediante claves primarias, foráneas y restricciones adecuadas.

2. **Diseño del formulario de entrada:**

   Implementar un formulario optimizado que permita introducir grandes volúmenes de protestos de forma ágil y segura, reduciendo al mínimo el tiempo de registro y evitando redundancias o errores.

3. **Gestión y explotación de datos:**

   Desarrollar operaciones CRUD (crear, leer, actualizar y eliminar) sobre todas las entidades principales, de modo que el investigador disponga de un sistema para administrar la información.

4. **Módulo de análisis y visualización:**

   Incorporar herramientas de consulta y análisis exploratorio mediante gráficos, tablas, mapas interactivos y grafos, que permitan descubrir patrones, relaciones y dinámicas económicas entre agentes, lugares y documentos.

5. **Exportación de información:**

   Habilitar mecanismos para exportar datos en formatos compatibles con aplicaciones externas de análisis (como Gephi para redes), garantizando la interoperabilidad del sistema.

6. **Accesibilidad y disponibilidad:**

   La aplicación tiene que estar pensada para poder usarse desde cualquier lugar y potencialmente por varias personas a la vez, por eso se decidió desplegar la aplicación en la nube (Google Cloud) para que sea accesible desde distintos dispositivos y ubicaciones.

1.3.3 Objetivos futuros

- El desarrollo o uso de una aplicación OCR con la que digitalizar automáticamente los protestos manuscritos
- Implementación de cuentas y usuarios para la aplicación web. Actualmente todo el mundo puede acceder a todo
- Ampliación de la capacidad de análisis
- Exportación de datos en más formatos(json, csv, informes pdf)
- Importación de datos guardados en otros formatos (csv, json)

En conjunto, estos objetivos buscan ofrecer una herramienta accesible y escalable, que no solo responda a los requisitos inmediatos del investigador, sino que también siente las bases para futuras ampliaciones.

1.4 Plan de trabajo

Una vez declarados los objetivos generales del proyecto en la primera reunión, el desarrollo de la aplicación se estructuró en una serie de fases que se definen los objetivos para la siguiente reunión. La planificación se realizó de forma iterativa, con reuniones periódicas entre los directores y el alumno, lfacilitó el intercambio de comentarios (feedback) y la adaptación a las necesidades a medida que estas iban surgiendo.

1.4.1. Comprensión del problema y análisis de requisitos

En esta etapa inicial se llevaron a cabo varias reuniones con el profesor Francisco Cebreiro Ares, usuario principal del sistema, y con el profesor Antonio Sarasa Cabezuelo, tutor del área informática. El objetivo fue comprender el contexto histórico de los protestos y las necesidades concretas de la investigación. A partir de este análisis se definieron los casos de uso principales:

- Digitalización manual de protestos.
- Gestión estructurada de entidades relacionadas (personas, roles, ciudades, monedas, letras, endosos, tipos de protesto…).
- Herramientas de análisis (búsquedas, gráficos, tablas, mapas y grafos).
- Exportación de datos

También se evaluó la posibilidad de integrar reconocimiento automático de manuscritos (OCR/HTR), aunque más adelante se pospuso por su elevada complejidad técnica.

Fase 1 - Diseño del modelo de datos y base de datos

En esta fase se analizaron las opciones entre bases de datos relacionales y no relacionales, optando finalmente por un modelo relacional en MySQL por adaptarse mejor a los documentos comerciales y jurídicos con reglas y relaciones claras. Se diseñó el modelo entidad–relación, se definieron las tablas y relaciones necesarias, y se implementó la base de datos.

Fase 2 - Implementación de la arquitectura y desarrollo del formulario de introducción de datos

En esta etapa se construyó la arquitectura en capas de la aplicación:

- Rutas para definir los endpoints de la aplicación.
- Controladores para gestionar peticiones y respuestas.
- Servicios de negocio (SA) encargados de validar reglas y coordinar operaciones.
- DAOs (Data Access Objects) para acceder a la base de datos MySQL.

La aplicación se desarrolló en el lado del servidor con Node.js y Express, en el cliente mediante vistas dinámicas en EJS y Bootstrap.

Paralelamente, se creó un formulario específico para la digitalización rápida de los protestos, que incorporó validaciones, autocompletado de entidades frecuentes y controles de consistencia para reducir errores en la inserción de datos.

Fase 3 - Módulo de análisis y visualización

Una vez consolidada la base de datos y el formulario, se incorporó un módulo para el análisis visual y exploración de datos, que incluye:

- Gráficos y tablas para análisis con Chart.js.
- Filtros de entidades

Fase 4 - Despliegue en contenedores y en la nube, mejoras en el formulario y análisis de datos con mapa Leaflet

Con el sistema ya operativo, se diseñaron los contenedores Docker para la aplicación y la base de datos, y se configuró su despliegue en Google Cloud. Esto garantiza que la herramienta pueda ejecutarse en diferentes entornos de forma uniforme y accesible desde cualquier ubicación.

Mapas interactivos para representar geográficamente ciudades y rutas Leaflet(mapas) y tabulaciones en el formulario para agilizar el volcado de datos.

Fase 5 - Exportación datos y Grafo D3.js

Una vez implementados los filtros y consultas principales, se añadió un método para exportar los resultados de una consulta en SQL. Además, se dejó planteada la posibilidad de ampliar la exportación a otros formatos como JSON o CSV, ya que gran parte de la infraestructura necesaria estaba prácticamente implementada.

Grafo de relaciones para explorar conexiones entre ciudades, documentos y roles implementado usando D3.js

En resumen, el plan de trabajo consistió en avanzar de manera progresiva desde la definición de los requisitos hasta el despliegue de una aplicación web funcional. Cada fase se apoyó en la anterior: primero se comprendió el problema y se diseñó un modelo de datos sólido; después se construyeron la arquitectura y el formulario de entrada, se continuó con el desarrollo de los módulos de análisis y visualización, subió a Google Cloud contenedorizado la aplicación con Docker y por último se implementó la exportación de consultas en sql.

CHAPTER 1 - INTRODUCTION

INTRODUCTION

This Bachelor's Thesis develops a web application to digitize, manage, and analyze protestos of bills of exchange—primary sources that help trace credit networks and bankruptcies in late-eighteenth-century Europe during the First Coalition War (1792–1797). A protest is a notarial deed that certifies the non-payment of a bill and typically reproduces the bill's content in full, including endorsements and the parties involved. By organizing this information in a relational database, the project moves beyond plain transcription and enables the systematic analysis of relationships among people, places, amounts, and dates.

The work responds to a specific research need raised by Professor Francisco Cebreiro Ares, who holds several amounts of historical and finantial documents to study the financial turmoil of the period of 1800. To cope with the size and heterogeneity of the documents in question, the system facilitates dependable data input, offers structured search and retrieval of specific documents, and provides visual analytics through charts, tables, maps, and network graphs. It additionally facilitates export to external tools (such as SQL) for further analysis.

In the technical design, the application follows a layered structure: Node.js and Express power the backend; EJS and Bootstrap implement the presentation layer; and MySQL serves as the database engine. The whole stack runs in Docker containers and is deployed on Google Cloud, a third user application which simplifies reproducibility and scaling. This decision remains over other options because it´s pragmatic approach, fast to develop, consistent with data-integrity requirements, and open to future additions such as handwriting recognition (OCR/HTR).

In short, the project sits at the intersection of Software Engineering and Economic History. It delivers a usable research tool while showing how careful relational modeling and modern web development can turn scattered archival sources into structured, actionable evidence.

1.1 MOTIVATION

The primary motivation is practical: transform thousands of heterogeneous historical documents into structured, analyzable data that can support hypotheses about monetary flows, key actors, and the timing of the crisis. Manual processing—fragmented and non-standard—limits traceability and comparability. A dedicated system was therefore needed to ensure quality, consistency, and speed from capture to analysis.

Academically, the project aligns with Digital Humanities by integrating document management with quantitative exploration. Rather than relying on general-purpose platforms, it offers a domain-specific solution for protestos: a data model tailored to bills, endorsements, roles, cities, currencies, and typologies, plus features aimed at historical questions (who interacts with whom, where and when non-payments arise, and which routes or relationships become central).

From software engineering perspective, the thesis serves as an experimental setup for good practice, layered architecture, separation of concerns, automated tests with the third party application Jest, and containerized deployment under realistic constraints: large record volumes, referential integrity, and complex queries. The chosen technologies favor easy to mantain and incremental growth, so the system can evolve without sacrificing stability.

Looking ahead, the platform lays the groundwork for future additions in the field of OCR/HTR on manuscripts, richer analytical modules (advanced descriptive statistics, pattern detection, reproducible reports), and multi-user collaboration. The overarching aim is to accelerate research, improve data quality, and open new paths for studying historical financial networks.

In short, the project sits at the intersection of Software Engineering and Economic History. It delivers a usable research tool while showing how careful relational modeling and modern web development can turn scattered archival sources into structured, actionable evidence.

1.3 Overview of the sources modeled in the application

Bills of exchange

A bill of exchange is an established financial instrument with widespread use between european merchants.

In its simplest form, one party (*the drawer*) instructs another party (*the drawee*) to pay a fixed amount to another person (*the payee*) at a specified time and place. Beginning in the medieval era, when moving coin over long distances was costly and unsafe, bills improved cross-border payments and, at times, involved remuneration through exchange rates or deferred terms rather than explicit interest. Over time, this bills were not limited to being payment orders but also as circulating credit instruments that could change hands, becoming essential to European financial markets well into the nineteenth century.

Endorsements

An endorsement transfers a bill's rights to recover funds from the current holder to a new creditor. Written on the reverse of the document, it designates the endorsee as the new beneficiary with full rights to payment. Because a bill can be endorsed repeatedly, the same instrument may circulate through multiple transactions, providing liquidity and serving as a adaptable way of settlement or short-term financing. In some historical cases, the chain of endorsements grew so long that extra sheets had to be added to continue the sequence.

Protests

A protest is a notarial act recorded when a bill is not accepted or not paid at moment of payment. It provides official evidence of non-payment, typically reproducing the original bill (including endorsements, marks, and annotations), documenting the formal demand, and any reasons advanced by the debtor. For historians, protests are very valuable sources: they preserve precise details about participants, places, amounts, dates, and even the auxiliary costs triggered by default (interest, commissions, notarial fees), enabling detailed economic analysis.

Capítulo 2 - Estado de la cuestión

En los últimos años ha aumentado la capacidad y el interés para digitalizar documentos históricos por parte de instituciones, archivos y comunidades académicas, y así realizar estudios históricos y económicos. El desarrollo de IA aplicada a la lectura automática de manuscritos y su trascripción a formato digital, ha aumentado la cantidad de información disponible para consultar y difundir. Esta nueva disponibilidad de fuentes ha impulsado nuevas aplicaciones especializadas y proyectos de investigación que emplean datos digitalizados para estudiar distintos aspectos de la Historia.

Sin embargo, al no tratarse de un ámbito económicamente rentable, la mayor parte de estas iniciativas provienen de universidades, organismos públicos, ONGs e investigadores particulares. Al no disponer de una inversión tan alta como otros campos, no existe una herramienta única y consolidada para abarcar todo el proceso, desde la digitalización hasta el análisis, sino que suelen utilizarse aplicaciones parciales para cada etapa: digitalización(ej. Transkribus), gestión y explotación de datos(ej. hojas de cálculo, bases de datos), o análisis y visualización(ej. Gephi). A menudo, los propios investigadores desarrollan soluciones propias adaptadas a sus necesidades.

La aplicación desarrollada en este TFG se enmarca en este contexto. Su propósito es integrar en una sola herramienta la digitalización, gestión y análisis de protestos de letras de cambio, con el que realizar estudios con un enfoque metodológico cercano al de Monetary Geography before the Industrial Revolution (Flandreau et al., 2009)[5], pero apoyándose en tecnologías de análisis más actuales y adaptadas a otra épocas y otros datos.

Los protestos de letras de cambio, al tratarse de contratos relativamente estandarizados y con responsabilidades claramente definidas, constituyen un tipo de documentación idónea para su digitalización y análisis, que puede aportar mucha información. Por este motivo es importante examinar cómo otros proyectos han abordado problemas similares, aun cuando no exista una herramienta única que integre todas las funciones que aquí se plantean.

2.1 Digitalización de documentos históricos

En los últimos años, uno de los ámbitos más dinámicos ha sido la conversión de manuscritos y documentos impresos en versiones digitales accesibles en línea. Este proceso no solo garantiza la preservación del patrimonio, sino que también abre nuevas posibilidades de consulta y análisis.

- Europeana: una plataforma de la Unión Europea que agrupa millones de documentos digitalizados de archivos, bibliotecas y museos. Aunque es muy útil para la preservación y difusión cultural, sus funcionalidades de análisis son limitadas. [6]
- PARES (Portal de Archivos Españoles): repositorio digital gestionado por el Ministerio de Cultura de España. Permite consultar documentos escaneados y metadatos básicos, pero no está diseñado para análisis relacionales ni visualización avanzada. [7]
- Transkribus: herramienta de referencia en el campo del HTR, que ofrece modelos de reconocimiento entrenados para la transcripción automática de documentos manuscritos. Aunque muy potente, requiere grandes volúmenes de entrenamiento y un esfuerzo inicial de preparación que en el marco de este TFG resultaba inviable. [8]

Si bien estas iniciativas garantizan la conservación del patrimonio documental, no resuelven el reto específico de convertir información manuscrita en estructuras relacionales aptas para el análisis económico e histórico detallado.

2.2 Gestión y explotación de datos históricos

Más allá de la digitalización, han surgido plataformas destinadas a la gestión estructurada de datos históricos. Algunas de las más utilizadas en el ámbito de las Humanidades Digitales son:

- Omeka: gestor de colecciones digitales orientado a archivos y museos, que permite crear catálogos y exposiciones virtuales. Sin embargo, su foco está en la difusión cultural, no en el análisis intensivo de datos. [9]

- Heurist: sistema flexible para modelar y gestionar bases de datos complejas en proyectos de investigación histórica. Destaca por su accesibilidad, pero su personalización para casos muy específicos puede resultar limitada. [10]
- Nodegoat: plataforma que combina modelado de datos con visualización en mapas y grafos, muy utilizada en proyectos académicos. Aunque potente, no está adaptada a la especificidad de los protestos, ni permite un flujo de introducción rápida de registros. [11]

Estos sistemas evidencian la tendencia hacia bases de datos relacionales y visualizaciones interactivas, validando el enfoque seguido en este TFG. No obstante, muestran también la necesidad de soluciones más ligeras, adaptadas y focalizadas en un dominio concreto.

2.3 Herramientas de análisis de redes y visualización

En el ámbito del análisis de relaciones económicas e históricas, destacan varias herramientas que permiten representar interacciones entre agentes y lugares, aqui pongo varios ejemplos:

- Gephi: software ampliamente usado en la investigación académica, que permite analizar estructuras complejas y métricas de grafos, aunque exige que los datos estén previamente organizados en un formato compatible y permite la visualización de redes**.** [12]
- Cytoscape: plataforma similar a Gephi, utilizada inicialmente en biología de sistemas, pero con aplicaciones en otros dominios. Su complejidad y curva de aprendizaje la hacen menos accesible para un investigador de Historia Económica. [13]
- QGIS: sistema de información geográfica de código abierto que posibilita la representación de datos en mapas. Aunque es muy útil para análisis espaciales, no incorpora un sistema propio de gestión documental. [14]

Estas herramientas son excelentes para la fase de análisis, pero no incluyen un módulo de gestión documental ni facilitan la captura inicial de datos. Por ello, la

aplicación desarrollada en este TFG actúa como puente entre los documentos históricos y las plataformas de análisis avanzadas.

2.4 Proyectos en Historia Económica

Entre los proyectos que estudian datos históricos y ya digitalizados podemos encontrar Venice Time Machine en el que han explorado archivos para estudiar redes mercantiles y financieras. En el contexto español, hay repositorios como PARES y estudios sobre comercio atlántico y letras de cambio han generado bases de datos ad hoc, si bien muchas permanecen cerradas o ligadas a un equipo concreto. [7]

En términos metodológicos, el enfoque de “monetary geography” propuesto por Flandreau, relaciona espacios, monedas y flujos, y constituye un referente para pensar los patrones económicos previos a la industrialización, compatible con el objetivo de este TFG de modelar personas, roles, ciudades y monedas a partir de protestos. [5]

Lo que diferencia este TFG es su aplicación práctica y adaptada: un sistema que no solo digitaliza y almacena protestos, sino que también facilita su análisis inmediato a través de gráficos, mapas y exportaciones, ofreciendo al investigador una herramienta ligera, funcional y especializada.

En conjunto, el estado de la cuestión muestra que, si bien existen múltiples plataformas para digitalizar, gestionar o analizar documentos históricos, la mayoría responden a objetivos generales (preservación, catalogación o visualización) y no a flujos de trabajo centrados en la captura rápida y el análisis relacional de fuentes concretas como los protestos. En este sentido, este TFG se propone cubrir ese hueco para el caso particular de los protestos de letras de cambio.

Capítulo 3 - Tecnologías empleadas

1. **HTML** (HyperText Markup Language) es el lenguaje que estructura las vistas de la aplicación web.

2. **CSS** (Cascading Style Sheets) es el lenguaje que da estilo al código en HTML. Se empleó para personalizar la apariencia y dar un aspecto limpio y profesional.

3. **EJS** (Embedded JavaScript Templates) es un motor de plantillas que permite crear páginas dinámicas con datos enviados desde el backend. Gracias a eso las vistas pueden mostrar y recibir información directamente desde la aplicación y actualizar los datos dinámicamente.

4. **Bootstrap** es un framework de CSS y HTML que facilita la creación de interfaces modernas y adaptables. Se utilizó para crear vistas profesionales rápidamente gracias a sus componentes estandarizados.

5. **Express** es un framework para Node.js que simplifica la creación de aplicaciones web. En este proyecto se utilizó como el núcleo de la aplicación. Permitiendo definir rutas, gestionar peticiones y respuestas, y conectar la aplicación: la lógica de negocio con las vistas y con la base de datos Es ligero y fiable lo que le lleva a ser un estándar en la construcción de aplicaciones web.

6. **Javascript** es el lenguaje de programación que se ejecuta tanto en el cliente (navegador) como en el servidor (Node.js y Express.js). En este proyecto se utilizó en ambos lados: en el cliente para añadir interactividad a las páginas EJS y validaciones en los formularios, y en el servidor para implementar la lógica de negocio y conectar con la base de datos.

7. **MySQL** es un sistema de gestión de bases de datos relacional. Se escogió por su robustez, fiabilidad y familiaridad adquirida durante la carrera. En este proyecto se utilizó para modelar los protestos y las entidades relacionadas

(personas, letras de cambio, endosos, monedas, etc.), garantizando la integridad de los datos.

8. **Docker** es una aplicación que permite empaquetar o contenedorizar aplicaciones junto con todas sus dependencias. Esto simula un sistema que, independientemente del equipo en el que se encuentre la aplicación, esta se ejecuta de la misma forma siempre, evitando problemas de errores de compatibilidad. En este proyecto se crearon dos contenedores uno para la aplicación y otro para la base de datos, simplificando así el despliegue a los servidores de Google Cloud.

9. **Google Cloud** (GCP) es la plataforma de servicios en la nube de Google. En este trabajo se utilizó para desplegar la aplicación y la base de datos en un entorno accesible desde cualquier navegador. Gracias a ello, la herramienta no depende de un único ordenador, puede ser usada de forma remota por el investigador y puede emplear el dispositivo que desee.

10. **Charts.js** es una librería de JavaScript de código abierto usada para crear gráficos de líneas (evolución del número de entidades buscadas a lo largo del tiempo), gráficos de barras(cantidades de dinero más usadas) y gráfico de sectores (monedas más usadas en las entidades seleccionadas).

11. **D3.js** (Data-Driven Documents) es una librería de Javascript que sirve para crear gráficos, visualizaciones o en el caso de esta aplicación el Grafo de conexiones, en el que se ve que ciudades están más relacionadas con otras y con cuales.

12. **Leaflet** es una librería especializada en mapas interactivos. En la aplicación se usa en el mapa interactivo para mostrar información de las ciudades como la ciudad con más de las entidades seleccionadas(protesto, letras….) ciudad con más dinero gastado o número de relaciones(encuanto documento o relaciones es referenciado en común con otra ciudad,

Capítulo 4 - Arquitectura de la aplicación y modelo de datos

4.1. Arquitectura general de la aplicación

La aplicación se ha desarrollado con una arquitectura en capas que separa presentación, control, lógica de negocio y acceso a datos. El backend utiliza Node.js y Express, las vistas están servidas con EJS y la persistencia se realiza en MySQL mediante el driver mysql2 y un pool de conexiones. El frontend incorpora Bootstrap 5 y jQuery, y los módulos de análisis integran Chart.js para gráficas, Leaflet para los mapas, D3.js para los grafos. El despliegue se realiza en Google cloud facilitado con Docker.

Se usan los patrones:

MVC (Modelo vista controlador) o por capas

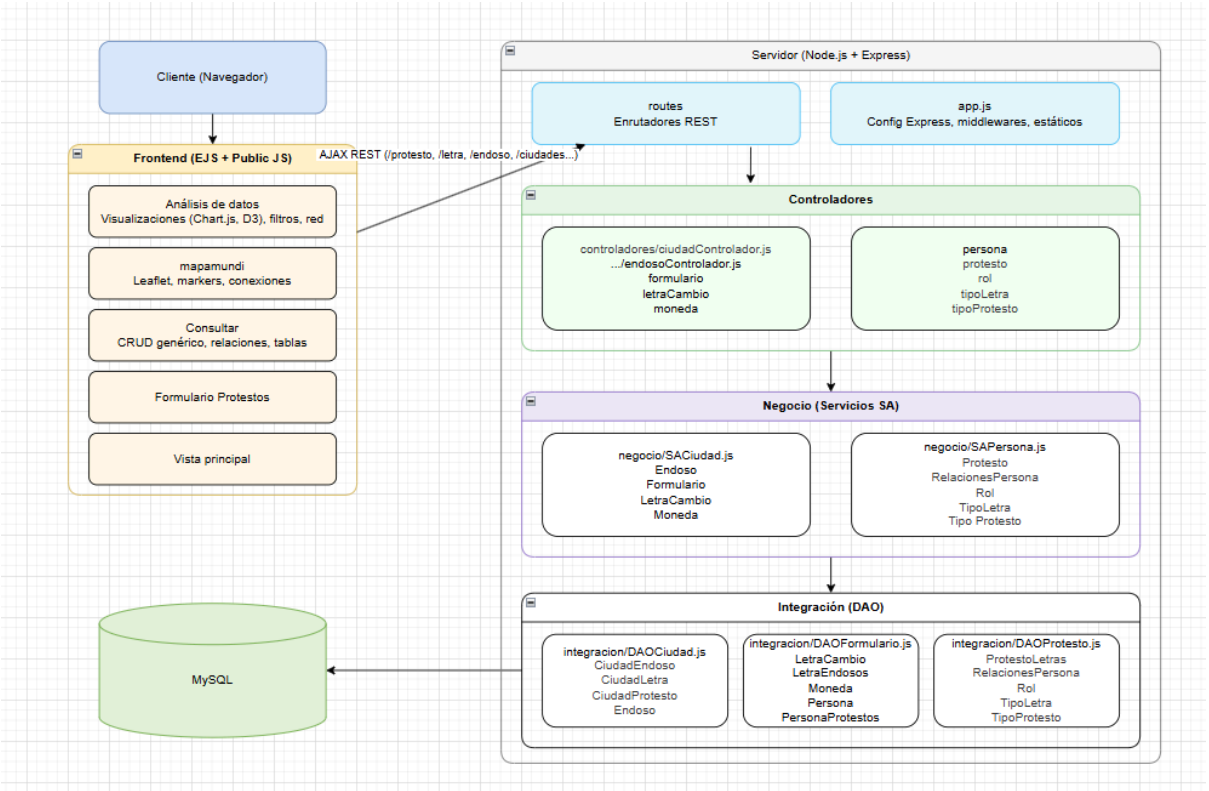


Figura 3 – Diagrama arquitectura de la aplicación

4.2. Estructura del proyecto y flujo de petición

El punto de entrada está en app.js, donde se configuran middlewares, el motor de vistas EJS, los estáticos public/ y el enrutado (routes/).
El flujo estándar es:

1. **Ruta** (por ejemplo, routes/protestoRoutes.js) recibe la petición;
2. **Controlador** (p. ej., controladores/protestoControlador.js) válida y traduce a operaciones de dominio;
3. **Servicio de Aplicación (SA)** (p. ej., negocio/SAProtesto.js) aplica reglas de negocio y orquesta transacciones;
4. **DAO** (p. ej., integracion/DAOProtesto.js) ejecuta SQL sobre MySQL;
5. El **resultado** vuelve al controlador y se renderiza o retorna como JSON.

4.3. Capa de presentación (vistas)

Las vistas se organizan por casos de uso:

- Formulario (views/formulario/formulario.ejs): interfaz de digitalización manual con secciones dinámicas para protesto, letras y endosos. Usa utilidades de autocompletado y búsqueda y modales de selección/alta rápida.
- Consulta (views/consultar/consultar.ejs): listados con filtros y acciones CRUD.
- Análisis (views/analisis/analisis.ejs): integra Chart.js, Leaflet y D3; los scripts public/javascripts/analisis/* implementan filtros, exportaciones y un mapamundi interactivo (mapamundi.js) con Leaflet y markers por ciudad.

La capa cliente se apoya en Bootstrap para maquetación responsive y en utilidades propias (public/javascripts/formulario/*.js) para validaciones, mostrar/ocultar secciones, autorrellenar entidades frecuentes y paginar resultados en análisis.

4.4. Enrutado y controladores

El directorio routes/ agrupa rutas por recurso: personas, ciudades, monedas, letras, endosos, protestos, catálogos (tipos), y módulos funcionales (formulario, analisis, consultar, exportar, relaciones). Algunos ejemplos:

- routes/formulario.js expone /save (persistencia transaccional del formulario) y /last-protesto (obtener datos del último protesto para el autorellenado).
- routes/protestoRoutes.js ofrece operaciones CRUD, relaciones letra–protesto y monta subrutas para tipos de protesto (routes/tipoProtestoRoutes.js).
- routes/analisisRoutes.js sirve la vista y un endpoint /estadisticas-mapa para KPIs de ciudades.

Los controladores (controladores/*) traducen HTTP → dominio y delegan en los SA. Por ejemplo, tipoProtestoControlador.js implementa search/get/create/update/delete delegando en SATipoProtesto. En protestoControlador.js la creación usa SAProtesto.createProtesto tras una validación mínima de datos obligatorios.

4.5. Servicios de Aplicación

La carpeta negocio/ concentra la lógica de negocio y la orquestación de procesos:

- SAFormulario.js: pieza central de la digitalización. Normaliza y resuelve IDs frente a nombres (personas, ciudades, monedas y tipos). Si el usuario escribe un nombre en vez de un ID, el servicio lo crea o localiza antes de persistir. Procesa roles y ciudades por documento, y coordina la transacción en DAOFormulario.
- SAProtesto.js: valida campos esenciales (fechas, importes, relaciones) y gestiona la creación de un protesto con sus vínculos (roles y letras).
- Servicios de catálogos (SAPersona, SACiudad, SAMoneda, SATipoLetra, SATipoProtesto): exponen utilidades como process*Fields para alta on-the-fly cuando llegan literales.

Este diseño reduce complejidad en controladores y centraliza reglas (integridad semántica, normalizaciones, decisiones por defecto).

4.6. Acceso a datos y transacciones

Los DAOs (integracion/*) encapsulan SQL explícito. Destaca DAOFormulario.js, que ejecuta una transacción completa para el guardado del formulario:

1. Inserta/recupera personas (búsqueda LIKE por nombre normalizado) para evitar duplicados;
2. Inserta el protesto y sus roles (protesto_roles);
3. Inserta cada letra y la relación protesto–letra (protesto_letra);
4. Inserta endosos y la relación letra–endoso (letra_endoso);
5. Inserta roles por documento (letra_roles, endoso_roles);
6. Commit; ante error, rollback y liberación de conexión.

Otros DAOs implementan CRUD y búsquedas tipadas: p. ej., DAOTipoProtesto.js y DAOPersona.js. El pool de MySQL (en config/db.js) configura límites de conexión, timeouts y maneja eventos de error comunes. En producción se soporta conexión a Cloud SQL mediante socketPath.

4.7. Modelo de Datos de la aplicación

Refleja la relación entre las entidades como los protestos, endosos y letras. Las entidades se han implementado con una estructura abierta que permite ampliar los tipos y la información de cada entidad o relación, dada la gran cantidad de variabilidad en los documentos antiguos. Aunque la relación entre las entidades se mantiene fija al ser contratos diseñados para ser fiables y claros en los derechos y obligaciones de cada uno de los participantes.

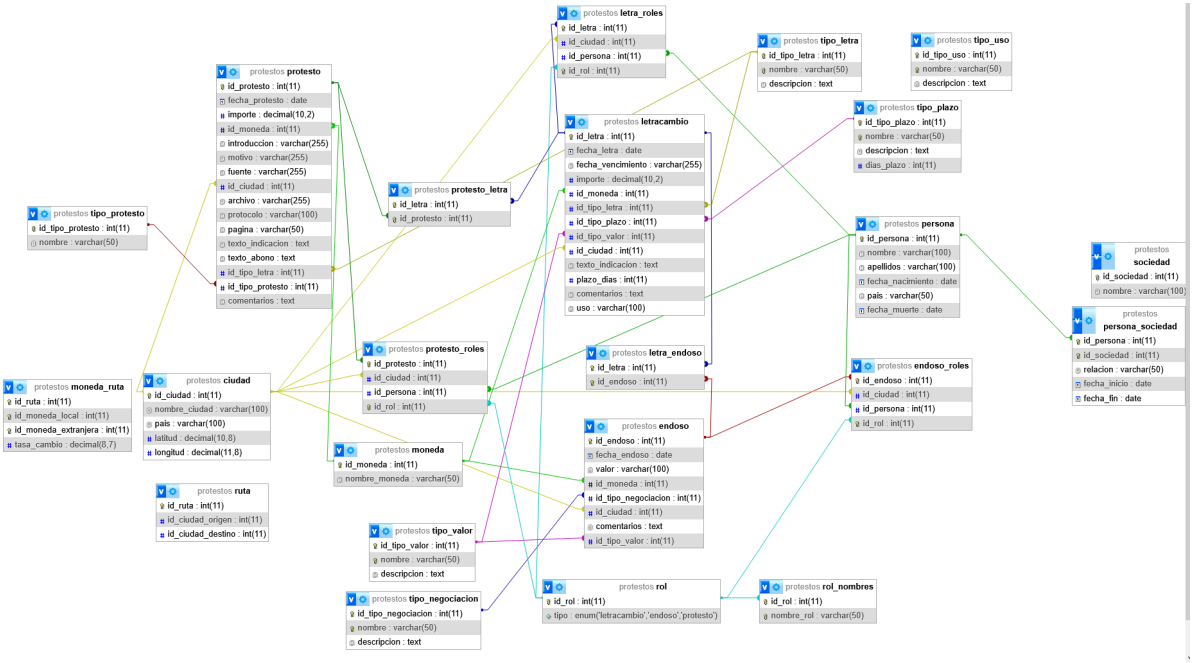


Figura 4 – Diagrama Entidad Relación de la base de datos

El modelo de datos permite transformar documentos notariales en cierta medida heterogéneos, en una base consultable y trazable. Se adoptó MySQL con diseño relacional porque el dominio, protestos, letras y endosos, responde a contratos estables y relaciones claras entre personas, lugares, importes y fechas. Este enfoque facilita conseguir integridad referencial, formular consultas analíticas con joins y, sobre todo, mantener la trazabilidad hacia el documento original

El esquema se articula en torno al protesto, documento que contiene -entre otras cosas- la letra, que a su vez contiene el endoso. Sin embargo, desde el punto de vista informático, estas tres partes constituyen las tres entidades principales del modelo: protesto, letra de cambio y endoso. Cada entidad posee su tabla principal y tablas de relación que registran la participación de las personas o “roles” (librador, librado, tomador, notario, etc.). Así, el protesto funciona como contenedor del incidente de impago; la letra concentra datos económicos (importe, moneda, vencimiento); y el endoso documenta la cadena de cesiones que hace transferible la letra. Esta descomposición respeta la lógica entre los documentos.

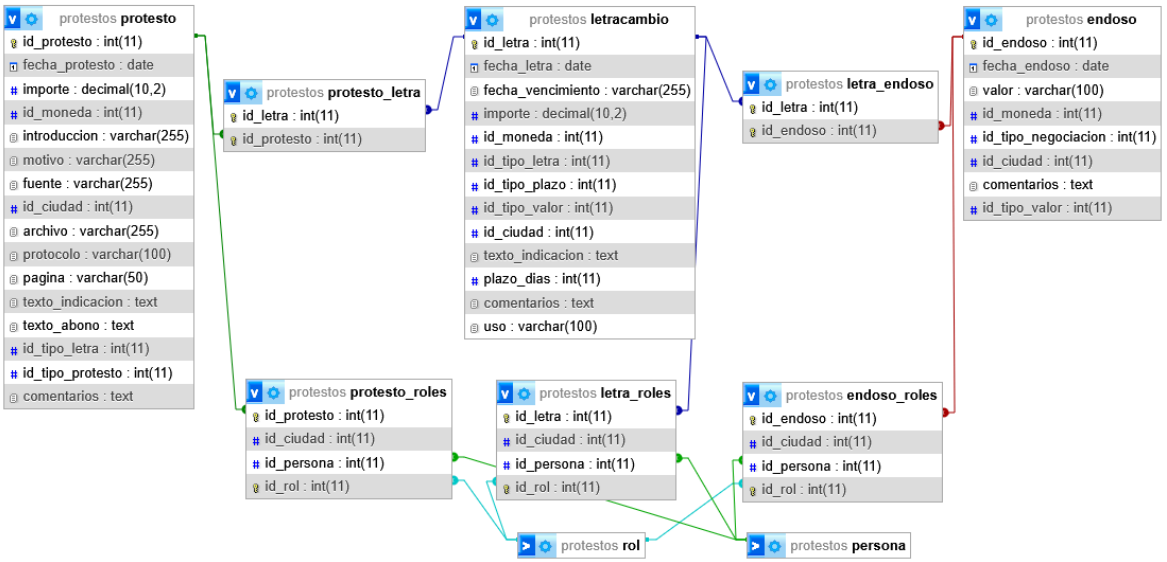


Figura 5 – Diagrama Entidad Relación entidades principales

Una decisión central fue separar personas de roles. La persona es única en todo el sistema, mientras que su papel se define en el contexto del documento mediante tablas intermedias (p. ej., protesto_roles, letra_roles, endoso_roles). Con ello se evitan duplicidades, se responde con precisión a “quién hizo qué y dónde” y se habilita el análisis de redes como D3(usado en el programa) o Gephi (popular entre los historiadores) sin sacrificar integridad

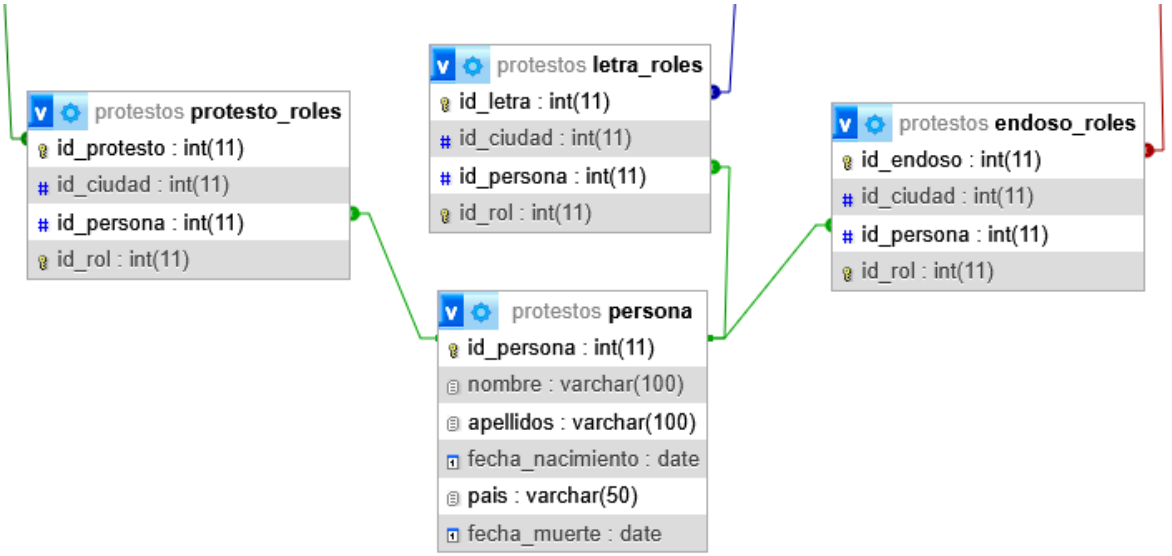


Figura 6 – Diagrama Entidad Relación Persona - Roles

Las ciudades se almacenan normalizadas e incluyen coordenadas geográficas, lo que garantiza consistencia en consultas espaciales y alimenta directamente el mapamundi de análisis. Las monedas se gestionan en una entidad independiente para controlar variantes históricas, dejando abierta la puerta a conversiones, ampliación de datos, o estudio de rutas futuras

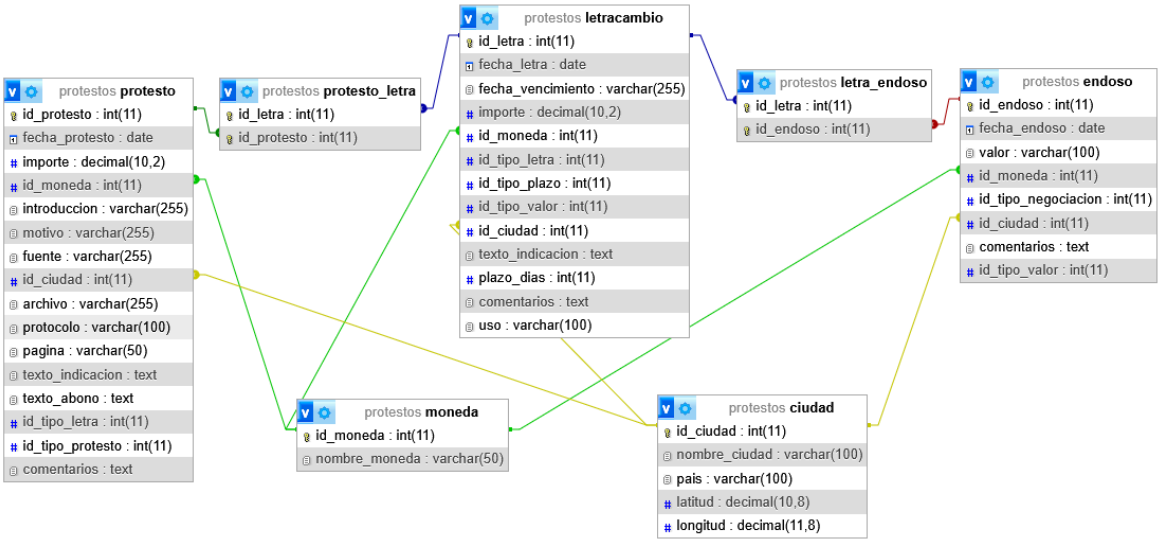


Figura 7 – Diagrama Entidad Relación Entidades principales - Moneda -Ciudad

Para preservar la trazabilidad documental, los protestos incluyen archivo, protocolo y, cuando procede, referencia al escaneo. Además, todas las inserciones del formulario se ejecutan en una transacción: primero se resuelven/crean catálogos (personas, ciudades, monedas, tipos) y después se insertan protesto, letras, endosos y sus roles. Cualquier fallo provoca rollback, de modo que el usuario no gestiona manualmente las dependencias. En el capítulo de implementación veremos un ejemplo de transacción.

Se valoraron alternativas (bases de datos no relacionales o documentales), pero la prioridad de calidad de datos, las validaciones fuertes y la claridad contractual de los documentos jurídicos justifican la elección del modelo relacional. Aun así, sería conveniente implementar un sistema que permita trasladar estos datos a un modelo no relacional como json para poder ser exportado a aplicaciones como Gephi o Excel.

4.8. Módulo de análisis y visualización

El módulo de análisis (views/analisis/analisis.ejs) consume endpoints específicos (routes/analisisRoutes.js) y construye gráficas (Chart.js), mapas (Leaflet) y grafos (D3) sobre los datos normalizados. Los scripts public/javascripts/analisis/* implementan:

- Filtros dinámicos por entidad, rango temporal y campos categóricos;
- Tablas;
- Mapamundi con markers y conexiones (ciudad-ciudad), con popups enriquecidos;
- Exportación de resultados en sql
- Análisis de conexiones mediante grafo hecho con D3.js

4.9. Pruebas automatizadas

El proyecto usa Jest para pruebas de servicio, persistencia y vistas:

- tests/SAFormulario.test.js: orquesta y mocks de dependencias para verificar reglas;
- tests/formulario.persistence.test.js: mock del pool MySQL y transacciones (begin/commit/rollback);
- tests/formulario_vista.test.js: render y captura de inputs en una simulación DOM.

4.10. Despliegue y contenedores

El Dockerfile empaqueta la aplicación, instala cliente MySQL y cloud-sql-proxy para GCP(Google Cloud Plataform). El docker-compose.yml levanta app y MySQL con volumen de inicialización.

Capítulo 5 - Implementación

5.1. Visión general del código

La aplicación se ha implementado con una arquitectura en capas que separa responsabilidades: rutas HTTP, controladores, servicios de aplicación y DAOs. Sobre esta base, las vistas EJS y en Bootstrap y los script en JavaScript conforman la capa de presentación. Esta separación garantiza que cada componente pueda evolucionar sin afectar al resto, simplificando el mantenimiento y la incorporación de nuevas funcionalidades.

En el repositorio, la estructura es clara y homogénea: app.js como punto de arranque, directorios routes/, controladores/, negocio/, integracion/, views/, public/ y tests/. Las convenciones de nombres siguen un patrón (*Controlador.js, SA*.js, DAO*.js), lo que facilita la lectura y localización de componentes.

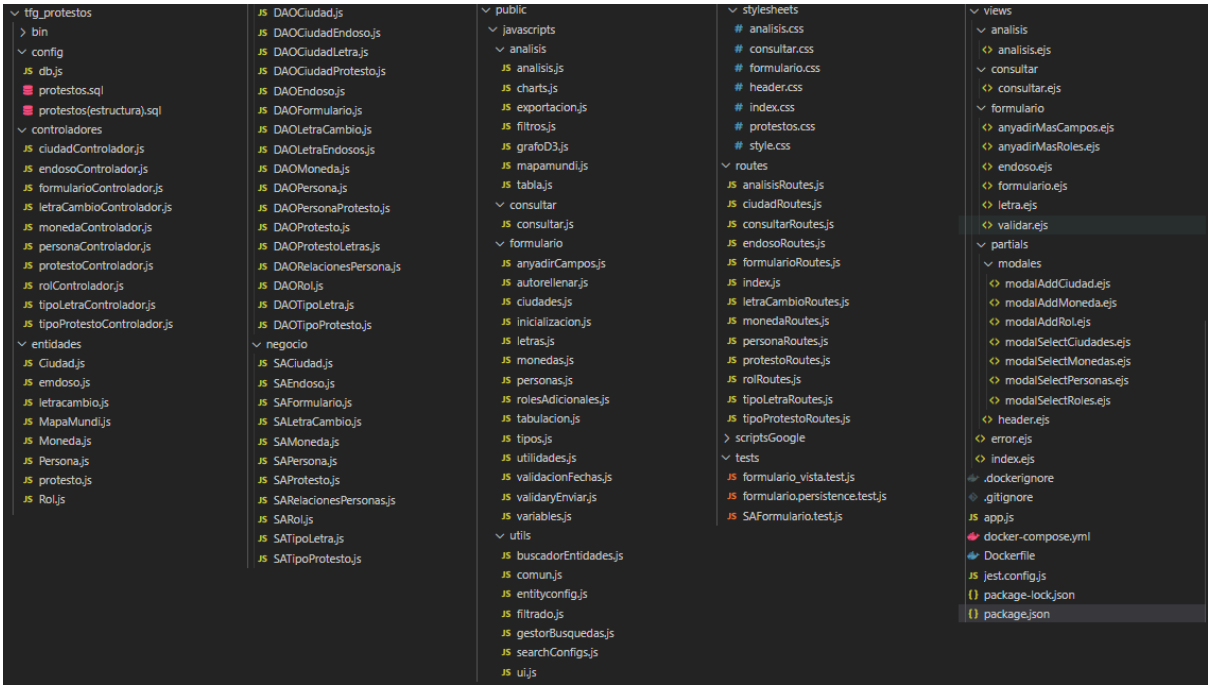


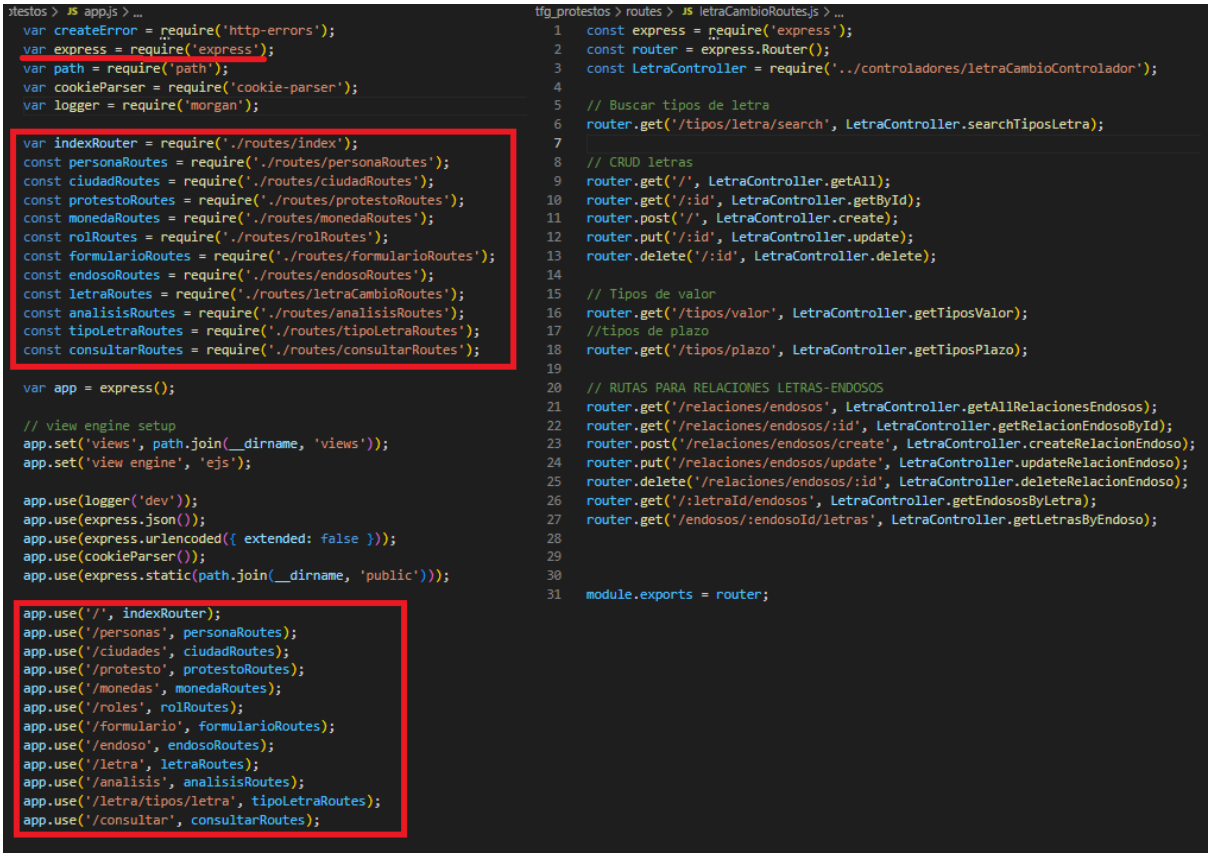
Figura 8 – Estructura código completa

5.2. Enrutado y controladores (Express)

El enrutado se organiza por recurso o módulo funcional, con rutas específicas para protestos, letras, endosos, catálogos y análisis. Cada ruta valida lo imprescindible, delega en el servicio correspondiente y devuelve la respuesta en JSON o renderizada en EJS. Los controladores se mantienen deliberadamente “delgados”, limitados a transformar datos de entrada/salida.

En el archivo app.js se configuran los middlewares generales (parsers, sesiones, estáticos) y se montan las rutas de cada módulo sobre prefijos específicos (/letras, /protestos, /analisis, etc.).La aplicación realiza la inicialización y la asignación de rutas.

Un ejemplo concreto es letraCambioRoutes.js donde se definen las operaciones CRUD de las letras de cambio, junto a rutas adicionales para catálogos y relaciones con endosos. Estas rutas solo declaran los endpoints y delegan en el controlador, manteniendo el código limpio.



```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
const personaRoutes = require('./routes/personaRoutes');
const ciudadRoutes = require('./routes/ciudadRoutes');
const protestoRoutes = require('./routes/protestoRoutes');
const monedaRoutes = require('./routes/monedaRoutes');
const rolRoutes = require('./routes/rolRoutes');
const formularioRoutes = require('./routes/formularioRoutes');
const endosoRoutes = require('./routes/endosoRoutes');
const letraRoutes = require('./routes/letraCambioRoutes');
const analisisRoutes = require('./routes/analisisRoutes');
const tipoLetraRoutes = require('./routes/tipoLetraRoutes');
const consultarRoutes = require('./routes/consultarRoutes');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/personas', personaRoutes);
app.use('/ciudades', ciudadRoutes);
app.use('/protesto', protestoRoutes);
app.use('/monedas', monedaRoutes);
app.use('/roles', rolRoutes);
app.use('/formulario', formularioRoutes);
app.use('/endoso', endosoRoutes);
app.use('/letra', letraRoutes);
app.use('/analisis', analisisRoutes);
app.use('/letra/tipos/letra', tipoLetraRoutes);
app.use('/consultar', consultarRoutes);
```

```
const express = require('express');
const router = express.Router();
const LetraController = require('../controladores/letraCambioControlador');

// Buscar tipos de letra
router.get('/tipos/letra/search', LetraController.searchTiposLetra);

// CRUD letras
router.get('/', LetraController.getAll);
router.get('/:id', LetraController.getById);
router.post('/', LetraController.create);
router.put('/:id', LetraController.update);
router.delete('/:id', LetraController.delete);

// Tipos de valor
router.get('/tipos/valor', LetraController.getTiposValor);
//tipos de plazo
router.get('/tipos/plazo', LetraController.getTiposPlazo);

// RUTAS PARA RELACIONES LETRAS-ENDOSOS
router.get('/relaciones/endosos', LetraController.getAllRelacionesEndosos);
router.get('/relaciones/endosos/:id', LetraController.getRelacionEndosoById);
router.post('/relaciones/endosos/create', LetraController.createRelacionEndoso);
router.put('/relaciones/endosos/update', LetraController.updateRelacionEndoso);
router.delete('/relaciones/endosos/:id', LetraController.deleteRelacionEndoso);
router.get('/:letraId/endosos', LetraController.getEndososByLetra);
router.get('/endosos/:endosoId/letras', LetraController.getLetrasByEndoso);



module.exports = router;
```

Figura 9 - Imagen app.js configuración de las rutas y letraCambioRouter.js

Los controladores actúan como traductores entre HTTP y la lógica de negocio. Se limitan a validar la entrada, llamar al servicio correspondiente y devolver el resultado en JSON o EJS. La Figura 10 muestra la función create del LetraCambioController, que recibe los datos de una nueva letra, y los pasa a SALetra.createLetra, gestionando errores de validación 400 o de servidor 500, el éxito 201.

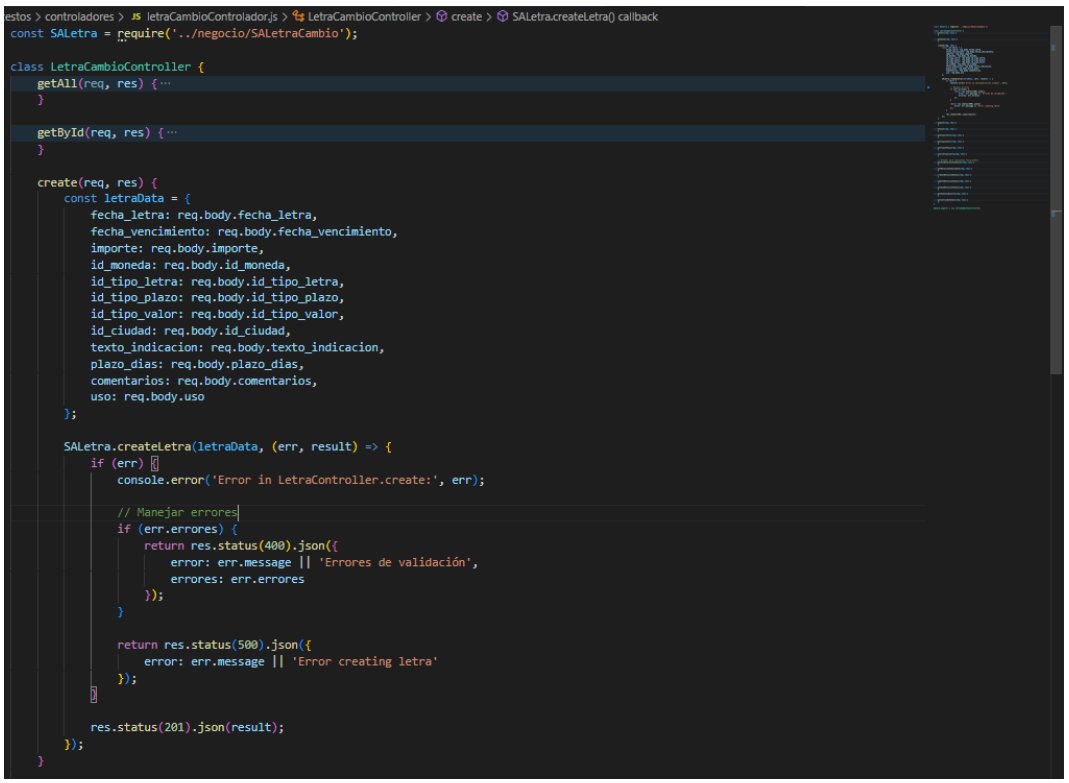
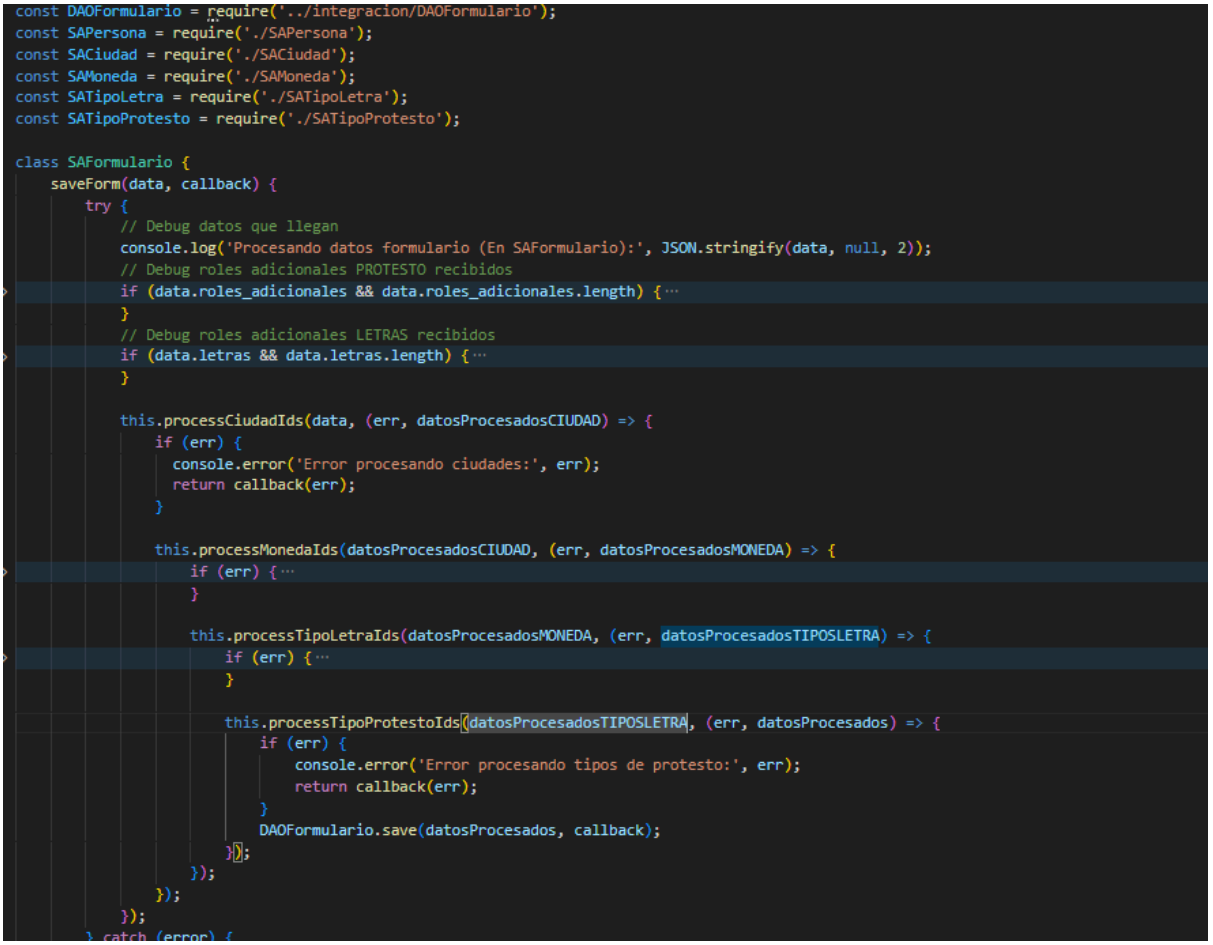


Figura 10 - Imagen función createLetra letraCambioControlador.js

5.3. Servicios de aplicación (lógica de negocio)

Los servicios encapsulan las reglas de negocio y resuelven casos complejos. Por ejemplo, SAFormulario gestiona todo el flujo de digitalización: normaliza nombres, resuelve referencias a catálogos y valida fechas, importes y roles. Otros servicios como SAProtesto o SAPersona aseguran consistencia en operaciones CRUD y evitan inconsistencias como eliminar entidades con dependencias.



```
const DAOFormulario = require('../integracion/DAOFormulario');
const SAPersona = require('./SAPersona');
const SACiudad = require('./SACiudad');
const SAMoneda = require('./SAMoneda');
const SATipoLetra = require('./SATipoLetra');
const SATipoProtesto = require('./SATipoProtesto');

class SAFormulario {
    saveForm(data, callback) {
        try {
            // Debug datos que llegan
            console.log('Procesando datos formulario (En SAFormulario):', JSON.stringify(data, null, 2));
            // Debug roles adicionales PROTESTO recibidos
            if (data.roles_adicionales && data.roles_adicionales.length) {···
            }
            // Debug roles adicionales LETRAS recibidos
            if (data.letras && data.letras.length) {···
            }

            this.processCiudadIds(data, (err, datosProcesadosCIUDAD) => {
                if (err) {
                  console.error('Error procesando ciudades:', err);
                  return callback(err);
                }

                this.processMonedaIds(datosProcesadosCIUDAD, (err, datosProcesadosMONEDA) => {
                    if (err) {···
                    }

                    this.processTipoLetraIds(datosProcesadosMONEDA, (err, datosProcesadosTIPOSLETRA) => {
                        if (err) {···
                        }

                        this.processTipoProtestoIds(datosProcesadosTIPOSLETRA, (err, datosProcesados) => {
                            if (err) {
                                console.error('Error procesando tipos de protesto:', err);
                                return callback(err);
                            }
                            DAOFormulario.save(datosProcesados, callback);
                        });
                    });
                });
            });
        } catch (error) {
```

Figura 11 - Imagen función saveFormulario de SAFormulario.js

5.4. Formulario de introducción de protestos

El formulario constituye la pieza central de la aplicación, ya que permite registrar de manera completa un protesto, incluyendo tanto las letras como los endosos, de la forma más ágil posible. Para ello se ha diseñado una vista en el frontend con lógica que valida y simplifica la introducción de datos, y una capa en el backend encargada de verificar y almacenar la información en la base de datos.

Uno de los principales retos en este proceso es garantizar que ciertos datos sean únicos manteniendo la introducción de datos lo más rápido posible. Para resolverlo, el formulario incorpora buscadores en aquellas entidades como persona, ciudad, moneda, tipo de protesto y tipo de letra. Como ayuda visual, cuando un valor ya existe en la base de datos, aparece resaltado en verde pero, si no existe, se muestra en azul y se creará automáticamente al guardar el formulario, siempre que no haya ningún error en la transacción de guardado.

En el lado del servidor se realiza una validación adicional, si se detectan nombres repetidos en entidades que deben ser únicas, estos se unifican bajo una misma referencia evitando duplicidades.

La única excepción son los roles. Dado que no es común la creación de nuevos roles, solo se pueden crear desde la sección de gestión de entidades o crud de la aplicación. También se pueden añadir nuevos campos a cada protesto, letras o endosos como roles adicionales o un comentario.

Por último, se ha implementado un sistema de tabulación que facilita el uso exclusivo del teclado. Los campos del formulario están ordenados siguiendo la misma estructura de los documentos originales, de modo que se pueden introducir los datos de manera fluida sin necesidad de usar el ratón.

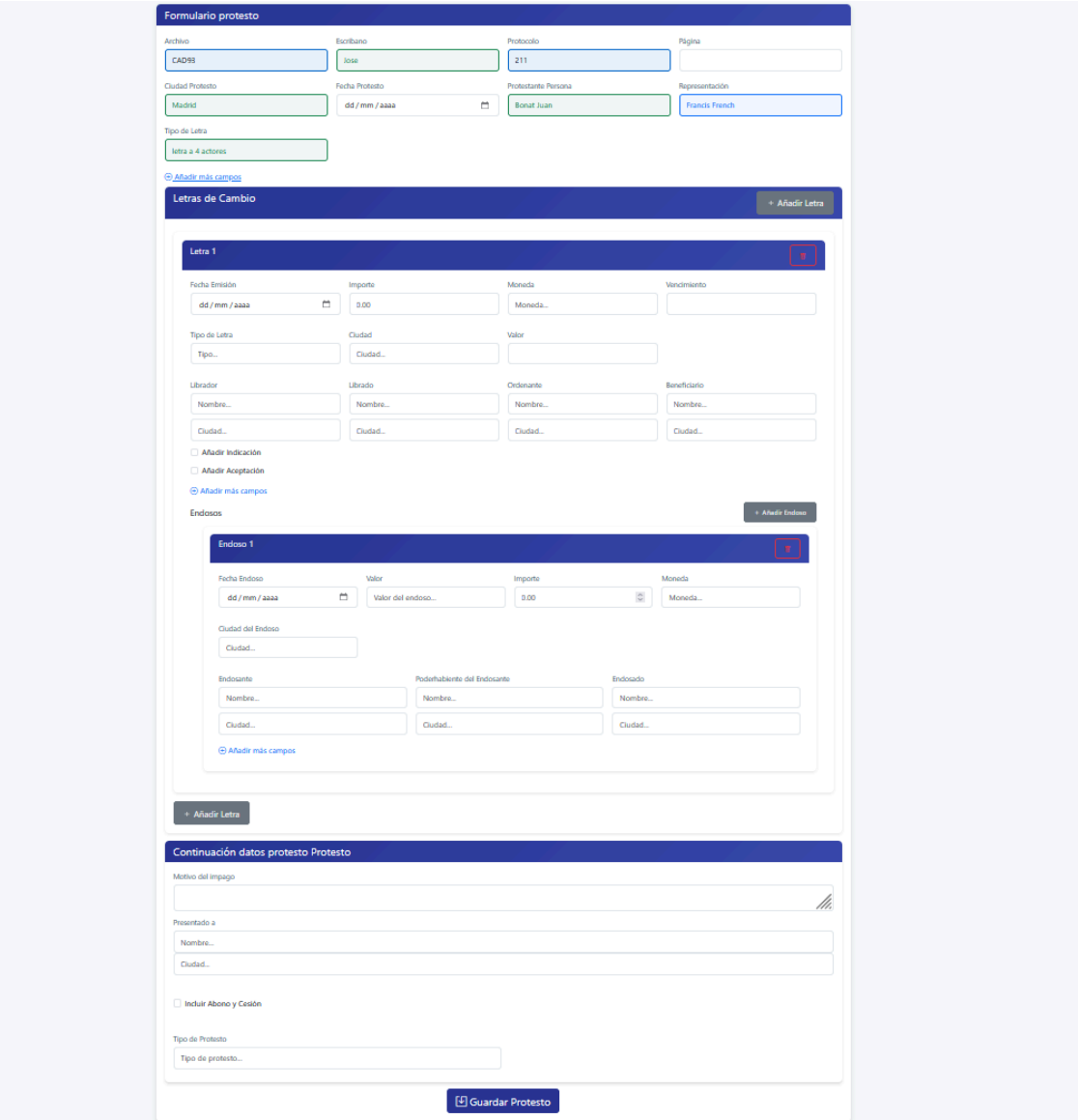


Figura 12 - Imagen formulario completo

5.4.1. Vista o frontend formulario:

Estructura: La vista principal se encuentra en views/formulario/formulario.ejs, organizada en secciones dinámicas. En la cabecera se recogen los datos básicos del protesto (archivo, escribano, protocolo/página, ciudad y fecha). A continuación se construye dinámicamente mediante javascript el formulario para cada letra permitiendo que cada protesto tenga varias: cada letra puede incluir campos propios, roles principales (librador, librado, ordenante, beneficiario) y secciones opcionales como Indicación o Aceptación. Dentro de cada letra, se anidan los endosos, que se agregan de manera incremental.

El formulario también carga Bootstrap, jQuery y una serie de módulos JavaScript para búsquedas/autocompletado, gestión dinámica de letras y endosos, validaciones, roles y campos adicionales, tabulación y carga inicial.

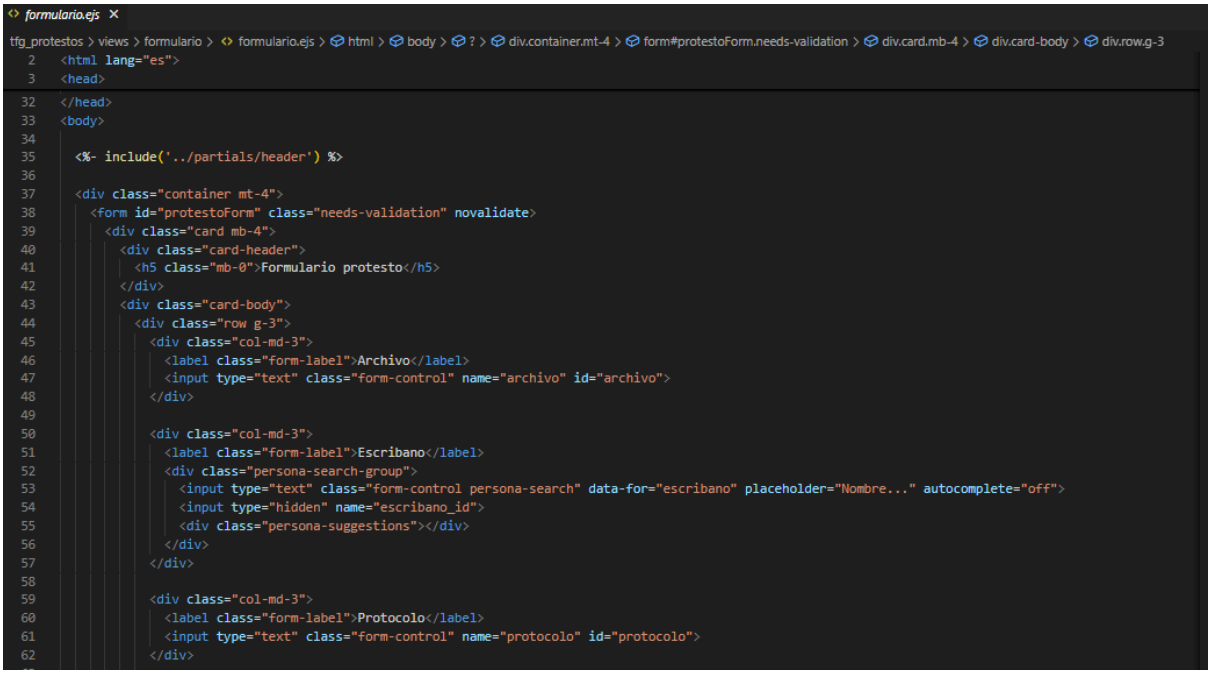


Figura 13 - Imagen inicio formulario.ejs

Autocompletado y búsqueda: El autocompletado facilita la selección de entidades como personas, ciudades, monedas o tipos de protesto. Cada campo visible tiene un input asociado y visible, un input oculto para almacenar el identificador real y un contenedor donde se muestran la lista sugerencias de la búsqueda.

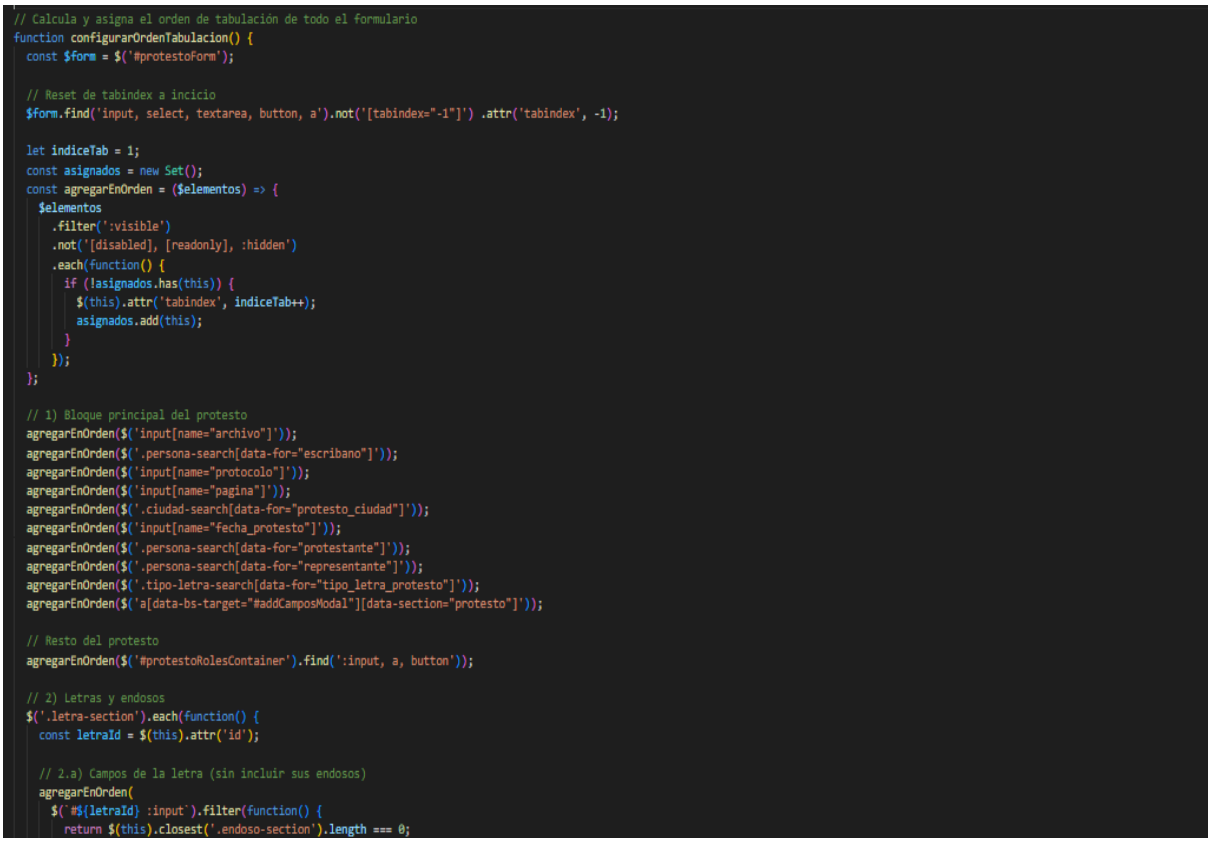
La configuración de estos buscadores se realiza en SearchConfigs, que define los endpoints de cada búsqueda. Al cargar la página, y cada vez que se añaden dinámicamente nuevas letras o endosos, el módulo GestorBusquedas inicializa los buscadores correspondientes a través de BuscadorEntidades.

Cuando el usuario escribe en el campo visible, tras dos segundos de retardo se consulta el servidor mediante una petición *get* con el contenido escrito hasta el momento. Si existe una coincidencia exacta, el sistema la auto selecciona, completando el nombre en el campo visible, el identificador en el oculto y mostrando el campo en verde. Si el usuario selecciona una sugerencia también se autocompleta. En caso de no encontrar resultados, el campo se marca como "nuevo" se muestra en azul y el backend se encarga de crear la entidad correspondiente si procede.

```
const SearchConfigs = {
    persona: {
        inputSelector: '.persona-search',
        suggestionsSelector: '.persona-suggestions',
        hiddenInputSelector: 'input[type="hidden"]',
        searchUrl: '/personas/search',
        entityName: 'persona',
        idField: 'id_persona',
        searchFields: ['nombre', 'apellidos'],
        formatDisplay: (p) => `${p.nombre} ${p.apellidos || ''}`,
        formatSuggestion: (p) => `${p.nombre} ${p.apellidos || ''} <small class="text-muted">${p.pais || ''}</small>`
    },
    ciudad: {
        inputSelector: '.ciudad-search',
        suggestionsSelector: '.ciudad-suggestions',
        hiddenInputSelector: 'input[type="hidden"]',
        searchUrl: '/ciudades/search',
        entityName: 'ciudad',
        idField: 'id_ciudad',
        searchFields: ['nombre_ciudad'],
        formatDisplay: (c) => c.nombre_ciudad,
        formatSuggestion: (c) => `${c.nombre_ciudad} <small class="text-muted">${c.pais || ''}</small>`
    },
    moneda: {
        inputSelector: '.moneda-search',
        suggestionsSelector: '.moneda-suggestions',
        hiddenInputSelector: 'input[type="hidden"]',
        searchUrl: '/monedas/search',
        entityName: 'moneda',
        idField: 'id_moneda',
        searchFields: ['nombre_moneda'],
        minLength: 1,
        formatDisplay: (m) => m.nombre_moneda,
        formatSuggestion: (m) => `${m.nombre_moneda} <small class="text-muted">${m.nombre_moneda.substring(0,3).toUpperCase()}</small>`
    },
```

Figura 14- Imagen implementación searchConfigs que define los endpoints y los formatos

Tabular: Gestiona el orden de tabulación por todo el formulario incluso cuando se añaden nuevas letras o endosos.



```
// Calcula y asigna el orden de tabulación de todo el formulario
function configurarOrdenTabulacion() {
  const $form = $('#protestoForm');

  // Reset de tabindex a incicio
  $form.find('input, select, textarea, button, a').not('[tabindex="-1"]') .attr('tabindex', -1);

  let indiceTab = 1;
  const asignados = new Set();
  const agregarEnOrden = ($elementos) => {
    $elementos
      .filter(':visible')
      .not('[disabled], [readonly], :hidden')
      .each(function() {
        if (!asignados.has(this)) {
          $(this).attr('tabindex', indiceTab++);
          asignados.add(this);
        }
      });
  };

  // 1) Bloque principal del protesto
  agregarEnOrden($('input[name="archivo"]'));
  agregarEnOrden($('.persona-search[data-for="escribano"]'));
  agregarEnOrden($('input[name="protocolo"]'));
  agregarEnOrden($('input[name="pagina"]'));
  agregarEnOrden($('.ciudad-search[data-for="protesto_ciudad"]'));
  agregarEnOrden($('input[name="fecha_protesto"]'));
  agregarEnOrden($('.persona-search[data-for="protestante"]'));
  agregarEnOrden($('.persona-search[data-for="representante"]'));
  agregarEnOrden($('.tipo-letra-search[data-for="tipo_letra_protesto"]'));
  agregarEnOrden($('a[data-bs-target="#addCamposModal"][data-section="protesto"]'));

  // Resto del protesto
  agregarEnOrden($('#protestoRolesContainer').find(':input, a, button'));

  // 2) Letras y endosos
  $('.letra-section').each(function() {
    const letraId = $(this).attr('id');

    // 2.a) Campos de la letra (sin incluir sus endosos)
    agregarEnOrden(
      $(`#${letraId} :input`).filter(function() {
        return $(this).closest('.endoso-section').length === 0;
```

Figura 15 - Imagen función configurarOrdenTabulacion de Tabulacion.js

Roles y campos adicionales: En los documentos antiguos era frecuente el cambio de el formato, anotaciones o actores que no están previstos, por lo que se necesita un mecanismo que de flexibilidad al protesto. Se pueden añadir campos mediante modales que permite añadir dinámicamente comentarios contextuales y roles adicionales en cualquier nivel del protesto: ya sea en el propio protesto, en una letra concreta o en un endoso específico de cada letra.

Se puede añadir un campo desde el botón “añadir campos adicionales” en el inferior de cada documento, esto desplegara una ventana modal implementada en anyadirMasCampos.ejs en que escoger el campo a añadir, un comentario(anyadirCampos.js) o añadir un rol (roles Adicionales.js)

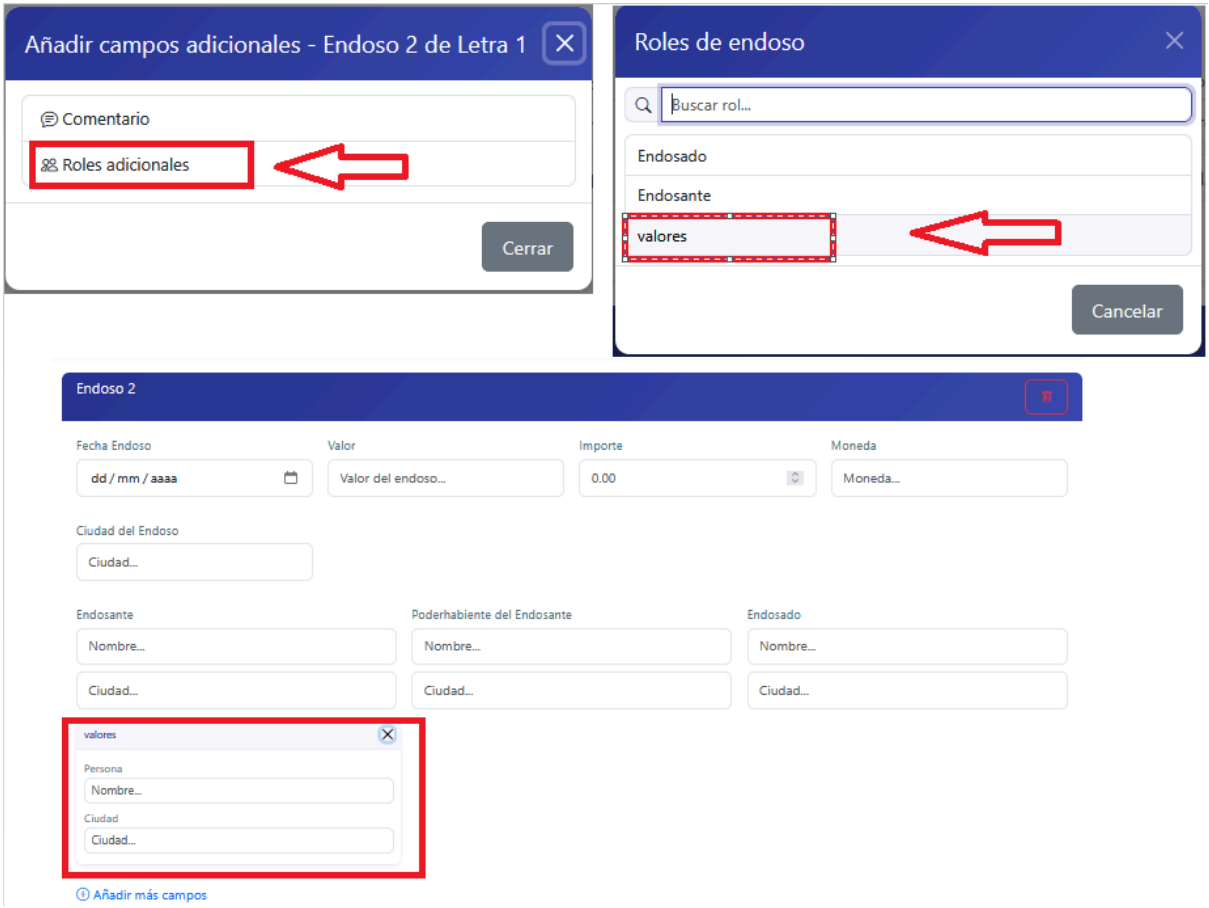


Figura 16 - Imagen proceso añadir rol adicional

Validaciones y envío: Comprobaciones de validez de fechas y de los campos del formulario antes de enviar el formulario.

Autorellenado de campos comunes, al crear un nuevo protestos se obtienen datos del anterior protesto guardado, si existe, y se rellenan los campos del protesto que se suelen repetir mucho de un protesto a otro, el archivo, la ciudad del protesto y el escribano. El autorellenado también se aplica en la creación de endosos. Cuando se crea un nuevo endoso dentro de una letra, si el endoso anterior tiene definido un endosado, este valor se copia automáticamente como endosante en el nuevo endoso.

5.4.2 Lógica guardado

Lógica SA y DAO: se encarga de garantizar la validez de la información y de gestionar la creación de todas las entidades asociadas al formulario en la base de datos. En particular, se asegura que entidades como ciudades, personas, monedas, tipos de letra y tipos de protesto se mantengan únicas, evitando duplicidades.

La función central en este proceso es DAOFormulario.save(), responsable de guardar en la base de datos todos los datos del formulario. Para ello, utiliza una transacción que ejecuta en bloque la inserción de protesto, letras, endosos y roles, aplicando un rollback automático en caso de error. Este mecanismo asegura la consistencia de la información incluso en operaciones complejas que implican múltiples llamadas a la base de datos.

El DAOFormulario accede directamente al resto de DAOs necesarios para completar la operación, sin recurrir a un transaction manager genérico. Esta decisión de diseño se tomó por motivos de simplicidad y eficiencia, ya que el formulario de protesto constituye el único caso de uso en el que resulta imprescindible coordinar varios DAOs dentro de una misma transacción.

```
class DAOFormulario {
  save(data, callback) {
    db.pool.getConnection((err, connection) => {
      if (err) return callback(err);

      connection.beginTransaction(err => {
        if (err) {
          connection.release();
          return callback(err);
        }

        // 1 Normalizar y resolver personas (si hay nombres (se crean nuevas personas) en vez de id (si se usan personas que ya estan en la bd))
        this.processAllPersonas(data, connection, (err, datosConPersonas) => {
          if (err) {
            return connection.rollback(() => {
              connection.release();
              callback(err);
            });
          }

          // 2 Insertar protesto y todo lo relacionado
          this._insertProtestoCompleto(connection, datosConPersonas, (err, result) => {
            if (err) {
              return connection.rollback(() => {
                connection.release();
                callback(err);
              });
            }

            connection.commit(commitErr => {
              if (commitErr) {
                return connection.rollback(() => {
                  connection.release();
                  callback(commitErr);
                });
              }
              connection.release();
              callback(null, { protestoId: result.protestoId });
            });
          });
        });
      });
    });
  }
```

Figura 17 - Imagen función save del formulario del protesto en DAOFormulario

5.5. Gestión de datos (CRUD de entidades)

El módulo de gestión se apoya en una vista común en EJS que carga dinámicamente las tablas según la entidad seleccionada. El fichero HTML define un panel con botones de entidad, un buscador y un contenedor para las tablas, mientras que el script consultar.js se encarga de renderizar, filtrar y actualizar los datos vía AJAX.

La está en entityConfig.js, donde cada entidad tiene configurados sus campos, columnas y endpoints REST. Esta configuración permite reutilizar el mismo

motor para operaciones CRUD alta, edición, búsqueda o eliminación, generando formularios dinámicos según la entidad.

```
function renderEntityTable(entityType, data) {
    const config = entityConfig[entityType];
    if (!config) return;

    let tableHTML = `
        <table class="table table-bordered table-hover" id="entityTable">
            <thead>
                <tr>
                    ${config.columns.map(col => `<th style="width: ${col.width || 'auto'}">${col.title}</th>`).join('')}
                </tr>
            </thead>
            <tbody>
        `;

    if (data && data.length > 0) {
        data.forEach(item => {
            tableHTML += '<tr>';
            config.columns.forEach(col => {
                if (col.field === '_actions') {
                    tableHTML += `
                        <td>
                            <div class="btn-group btn-group-sm" role="group">
                                <button type="button" class="btn btn-outline-primary btn-edit" data-id="${item[config.columns[0].field]}" title="Editar">
                                    <i class="bi bi-pencil"></i>
                                </button>
                                <button type="button" class="btn btn-outline-danger btn-delete" data-id="${item[config.columns[0].field]}" title="Eliminar">
                                    <i class="bi bi-trash"></i>
                                </button>
                            </div>
                        </td>
                    `;
                } else if (col.field === 'protesto_detalles' || col.field === 'letra_detalles' || col.field === 'endoso_detalles') {
                    const value = item[col.field] || '';
```

Figura 18 - Imagen renderizar la tabla entidades en ejs en consultar.js

```
const entityConfig = {
    protestos: {
        title: "Protestos",
        columns: [
            {title: "ID", field: "id_protesto", width: "80px"},
            {title: "Fecha", field: "fecha_protesto", width: "120px", format: "date"},
            {title: "Archivo", field: "archivo", width: "100px"},
            {title: "Protocolo", field: "protocolo", width: "100px"},
            {title: "Página", field: "pagina", width: "80px"},
            {title: "Ciudad", field: "ciudad_nombre", width: "120px"},
            {title: "Tipo Letra", field: "tipo_letra_protesto_nombre", width: "120px"},
            {title: "Tipo Protesto", field: "tipo_protesto_nombre", width: "120px"},
            {title: "Motivo", field: "motivo", width: "150px"},
            {title: "Acciones", field: "_actions", width: "120px"}
        ],
        endpoint: "/protesto",
        formFields: [
            {name: "fecha_protesto", label: "Fecha Protesto", type: "date", required: true},
            {name: "archivo", label: "Archivo", type: "text", required: false},
            {name: "protocolo", label: "Protocolo", type: "text", required: false},
            {name: "pagina", label: "Página", type: "text", required: false},
            {name: "id_ciudad", label: "Ciudad", type: "select", endpoint: "/ciudades", valueField: "id_ciudad", displayField: "nombre_ciudad", required: false},
            {name: "id_tipo_letra", label: "Tipo Letra", type: "select", endpoint: "/letra/tipos/letra", valueField: "id_tipo_letra", displayField: "nombre", required: false},
            {name: "id_tipo_protesto", label: "Tipo Protesto", type: "select", endpoint: "/protesto/tipos/protesto", valueField: "id_tipo_protesto", displayField: "nombre", required: false},
            {name: "motivo", label: "Motivo", type: "textarea", required: false},
            {name: "introduccion", label: "Introducción", type: "textarea", required: false},
            {name: "fuente", label: "Fuente", type: "text", required: false},
            {name: "importe", label: "Importe", type: "number", step: "0.01", required: false},
            {name: "id_moneda", label: "Moneda", type: "select", endpoint: "/monedas", valueField: "id_moneda", displayField: "nombre_moneda", required: false}
        ]
    },
    letras: {
        columns: [
            {title: "#", field: "id_letra", width: "80px"},
            {title: "Fecha Emisión", field: "fecha_letra", width: "120px", format: "date"},
            {title: "Vencimiento", field: "fecha_vencimiento", width: "120px"},
```

Figura 19 - Imagen configuración endpoints en entityconfig.js

5.6. Análisis de datos

El módulo de análisis permite a los investigadores explorar los datos de protestos, letras de cambio y endosos de forma interactiva desde diferentes perspectivas visuales. Su desarrollo se apoyó en tres librerías principales: Chart.js para las gráficas estadísticas, Leaflet para la representación geográfica sobre mapas, y D3.js para la visualización de redes de relaciones.

La interfaz se organiza en dos columnas: a la izquierda, un panel de filtros dinámicos que se adaptan a la entidad seleccionada (protestos, letras, endosos, personas, ciudades o la vista combinada “Todos”); a la derecha, el área de visualización principal con pestañas para cambiar entre las distintas representaciones. El backend expone endpoints específicos que devuelven datos ya agregados o normalizados para facilitar su consumo en cliente.

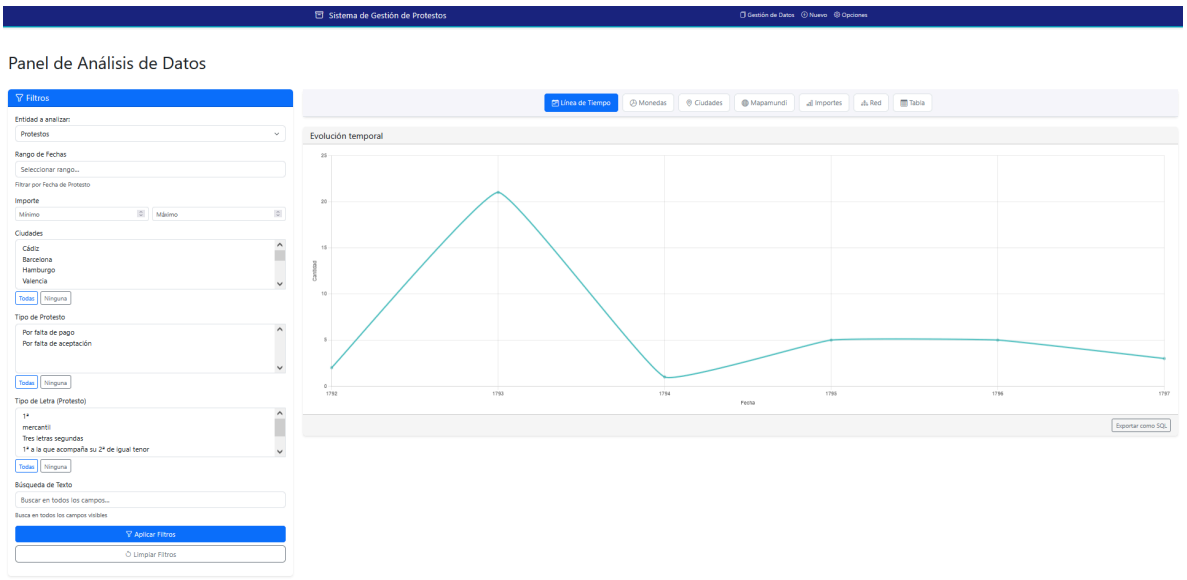
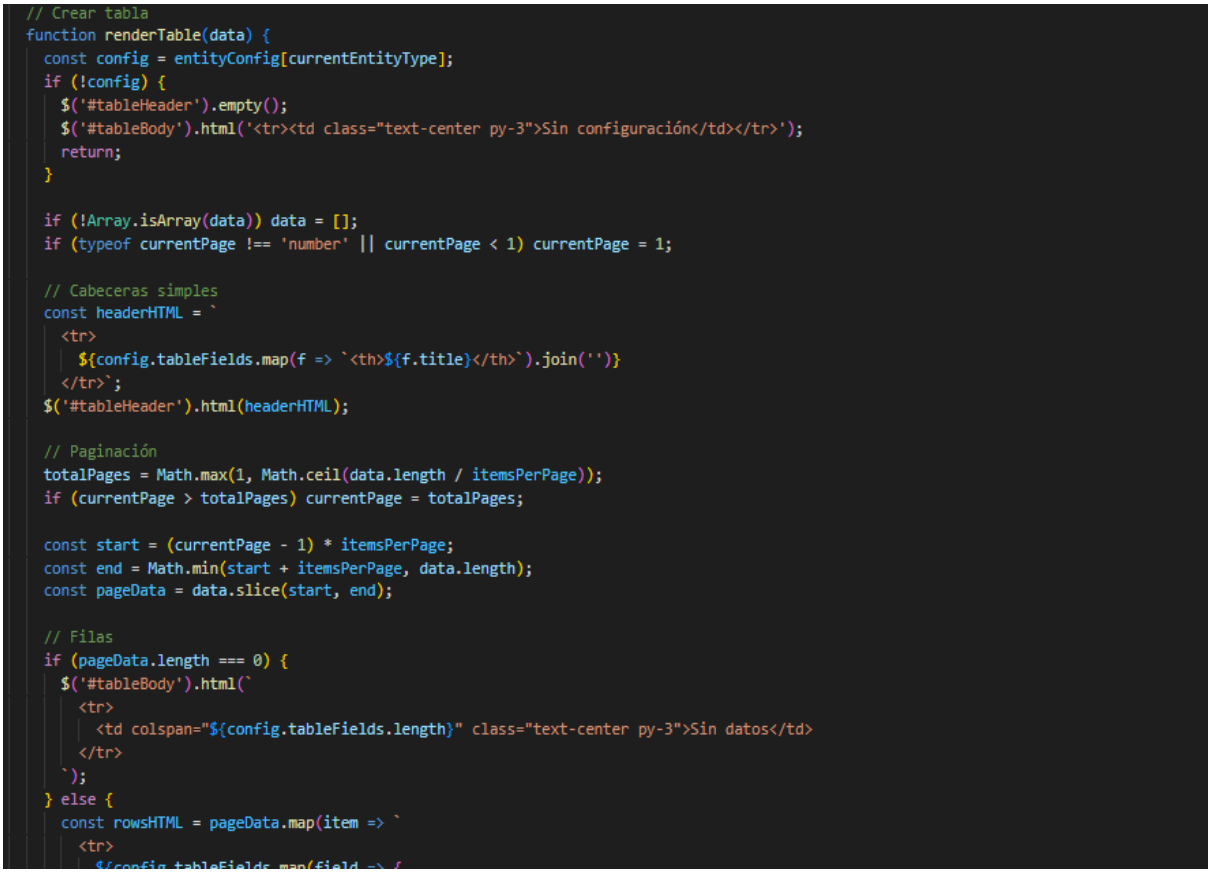


Figura 20 – Imagen panel de análisis de datos

5.6.1. Tablas

La vista de tabla se diseñó como una representación común a todas las entidades. El servidor devuelve los datos de las entidades filtradas por el usuario, y en el cliente se renderiza una tabla con cabeceras adaptadas a cada entidad (por ejemplo, fecha y motivo en protestos; vencimiento y valor en letras).

La búsqueda en tiempo real se implementó directamente sobre el conjunto filtrado en cliente, para evitar sobrecargar el servidor con consultas innecesarias. Asimismo, la tabla soporta paginación, ordenación en cabecera y un cuadro de búsqueda general.



```
// Crear tabla
function renderTable(data) {
  const config = entityConfig[currentEntityType];
  if (!config) {
    $('#tableHeader').empty();
    $('#tableBody').html('<tr><td class="text-center py-3">Sin configuración</td></tr>');
    return;
  }

  if (!Array.isArray(data)) data = [];
  if (typeof currentPage !== 'number' || currentPage < 1) currentPage = 1;

  // Cabeceras simples
  const headerHTML = `
    <tr>
      ${config.tableFields.map(f => `<th>${f.title}</th>`).join('')}
    </tr>`;
  $('#tableHeader').html(headerHTML);

  // Paginación
  totalPages = Math.max(1, Math.ceil(data.length / itemsPerPage));
  if (currentPage > totalPages) currentPage = totalPages;

  const start = (currentPage - 1) * itemsPerPage;
  const end = Math.min(start + itemsPerPage, data.length);
  const pageData = data.slice(start, end);

  // Filas
  if (pageData.length === 0) {
    $('#tableBody').html(`
      <tr>
        <td colspan="${config.tableFields.length}" class="text-center py-3">Sin datos</td>
      </tr>
    `);
  } else {
    const rowsHTML = pageData.map(item => `
      <tr>
        ${config.tableFields.map(field => {
```

Figura 21 - Función que crea la tabla y la puebla de datos

5.6.2. Filtros

El sistema de filtros es uno de los elementos clave. Se construye dinámicamente en función de la entidad seleccionada: si hay campo de fecha, se habilita un selector de rango; si hay campo numérico, se generan controles para mínimo y máximo; y si existen categorías (ciudades, monedas, tipos de protesto, tipos de letra…), se muestran listas desplegables múltiples. Además, todos los filtros se pueden limpiar rápidamente con un botón de “reset”.

La lógica de filtrado se concentra en una clase JavaScript (FiltrosAnalisis) que aplica los criterios de manera acumulativa sobre el conjunto de datos cargados. Los resultados filtrados se propagan automáticamente a todas las visualizaciones activas.

```
class FiltrosAnalisis {
  constructor() {
    this.filtros = {};
    this.entityConfig = {};
    this.currentData = [];
    this.callbacks = {
      onFilterChange: null
    };
  }

  iniciarFiltros(entityConfig, data, entityType) {
    this.entityConfig = entityConfig;
    this.currentData = data;
    this.currentEntityType = entityType;

    // Generar filtros según la entidad
    this.generateFilterUI(entityConfig[entityType], data);
    this.loadFilterOptions(entityConfig[entityType], data);
    this.bindFilterEvents();
  }
```

Figura 22 - Imagen función principal de analisis/filtros.js iniciarFiltros

5.6.3. Mapamundi (Leaflet)

La representación sobre mapa se implementó con Leaflet, que permite colocar marcadores sobre un mapa, se ha importado el mapa de 'Imagery © Esri'. Cada marcador corresponde a una ciudad y su tamaño y color varían según la métrica seleccionada (número de documentos, importe total o número de relaciones).

Los datos de coordenadas se obtienen de la tabla ciudad en la base de datos y se cachean en cliente para evitar peticiones redundantes. Además, cada marcador despliega un modal con estadísticas resumidas y un tooltip flotante.

Se añadió un selector para cambiar de métrica y un botón opcional para mostrar conexiones entre ciudades, lo que facilita visualizar redes de comercio o rutas de circulación de letras.

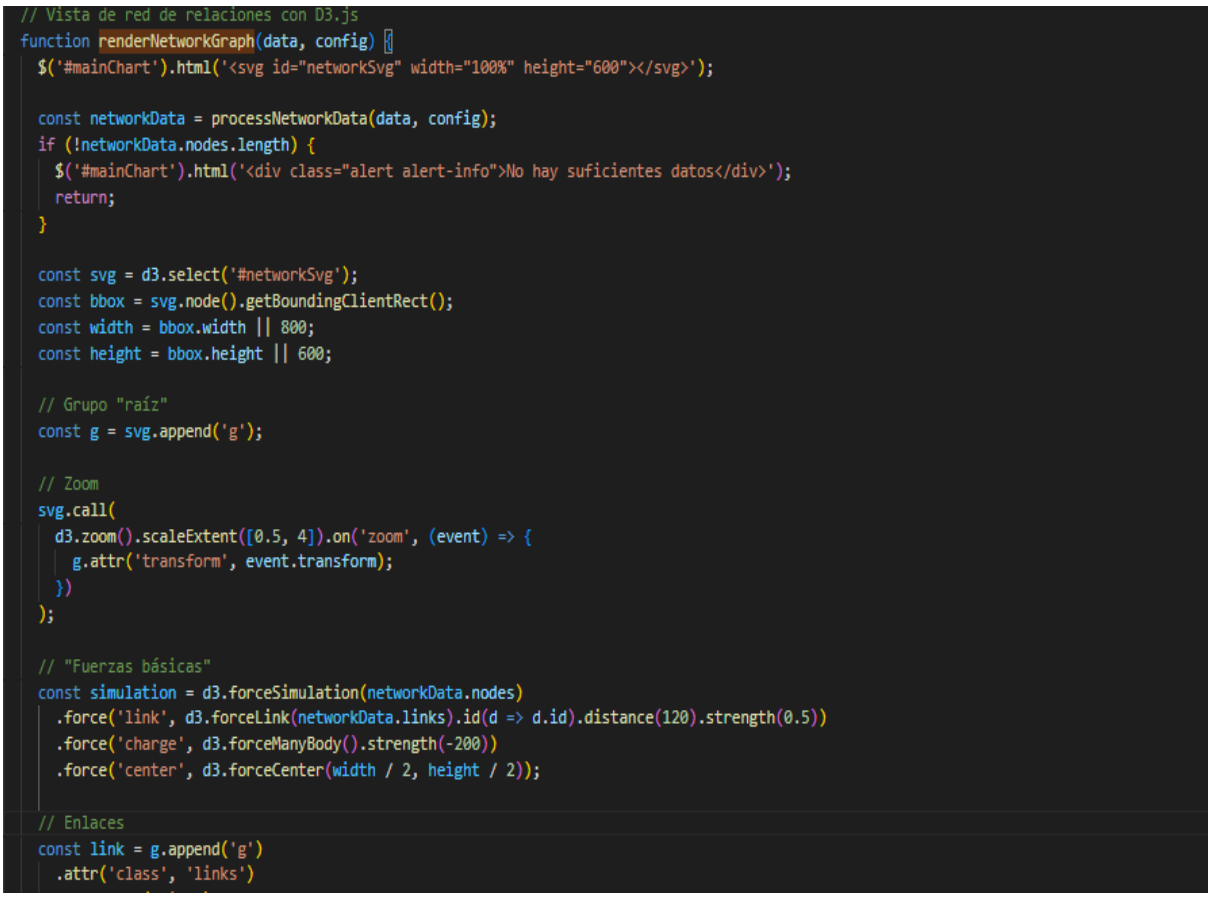
```
// Inicializar el mapa
initMap(containerId) {
    try {
        this.map = L.map(containerId, {
            center: [40.4168, -3.7038], //Centrado en Madrid
            zoom: 6,
            zoomControl: true,
            attributionControl: true
        });

        L.tileLayer('https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/tile/{z}/{y}/{x}', {
            attribution: 'Imagery © Esri',
            maxZoom: 18
        }).addTo(this.map);

        // Cargar datos de ciudades
        this.loadCityData();

    } catch (error) {
        console.error('Error inicializando el mapamundi:', error);
        this.showMapError('Error inicializando el mapamundi');
    }
}

// Cargar datos de ciudades
loadCityData() {
    $.get('/ciudades/estadisticas-mapa')
        .done((data) => {
            console.log('Datos mapa cargados:', data);
            this.cityData = data;
            this.updateMarcadoresMapa();
        })
        .fail((error) => {
            console.error('Error cargando datos de ciudades:', error);
            this.showMapError('Error cargando datos de ciudades');
        });
}
```

Figura 23 - Inicialización y población de marcadores de mapamundi.js

5.6.4. Grafo de relaciones (D3.js)

Para estudiar las interacciones entre ciudades, se implementó un grafo con D3.js. Cada nodo representa una ciudad y su tamaño depende del número de documentos en los que aparece. Los enlaces se generan cuando dos ciudades coexisten en un mismo protesto, letra o endoso, y su grosor se ajusta según la fuerza de la conexión.

La simulación utiliza fuerzas de atracción y repulsión para distribuir los nodos en un espacio equilibrado. Los nodos son interactivos: al situar el cursor sobre uno se destacan sus conexiones y se muestra un modal desplegable con datos de la ciudad.



```
// Vista de red de relaciones con D3.js
function renderNetworkGraph(data, config) {
  $('#mainChart').html('<svg id="networkSvg" width="100%" height="600"></svg>');

  const networkData = processNetworkData(data, config);
  if (!networkData.nodes.length) {
    $('#mainChart').html('<div class="alert alert-info">No hay suficientes datos</div>');
    return;
  }

  const svg = d3.select('#networkSvg');
  const bbox = svg.node().getBoundingClientRect();
  const width = bbox.width || 800;
  const height = bbox.height || 600;

  // Grupo "raíz"
  const g = svg.append('g');

  // Zoom
  svg.call(
    d3.zoom().scaleExtent([0.5, 4]).on('zoom', (event) => {
      g.attr('transform', event.transform);
    })
  );

  // "Fuerzas básicas"
  const simulation = d3.forceSimulation(networkData.nodes)
    .force('link', d3.forceLink(networkData.links).id(d => d.id).distance(120).strength(0.5))
    .force('charge', d3.forceManyBody().strength(-200))
    .force('center', d3.forceCenter(width / 2, height / 2));

  // Enlaces
  const link = g.append('g')
    .attr('class', 'links')
```

Figura 24 - Imagen función renderiza los nodos, sus enlaces, su zoom de un Grafo

5.6.5. Gráficas estadísticas (Chart.js)

El uso de Chart.js permite generar gráficas rápidas e interactivas en varios formatos: Línea temporal, Circular(o gráfico de tarta), Barras horizontales o Histogramas.

Cada gráfico se actualiza automáticamente según los filtros activos, de manera que el investigador puede explorar intervalos temporales o regiones concretas sin recargar la página.

```
// Gráfico circular de distribución de monedas
function renderCurrencyChart(data, config) {
  const currencyData = {};
  let hayDatos = false;

  data.forEach(item => {
    if (item.moneda_nombre) {
      hayDatos = true;
      if (!currencyData[item.moneda_nombre]) {
        currencyData[item.moneda_nombre] = 0;
      }
      currencyData[item.moneda_nombre]++;
    }
  });

  if (!hayDatos) {
    $('#mainChart').html('<div class="alert alert-info">No hay suficientes datos de monedas para mostrar un gráfico</div>');
    return;
  }

  const labels = Object.keys(currencyData);
  const values = labels.map(label => currencyData[label]);

  const ctx = document.createElement('canvas');
  ctx.id = 'currencyChart';
  $('#mainChart').html('').append(ctx);

  // Colores diferentes para cada moneda
  const backgroundColors = labels.map(() =>
    `rgba(${Math.floor(Math.random() * 255)}, ${Math.floor(Math.random() * 255)}, ${Math.floor(Math.random() * 255)}, 0.6)`
  );

  chartInstances.mainChart = new Chart(ctx, {
    type: 'pie',
    data: {
      labels: labels,
      datasets: [{
        data: values,
```

Figura 25 - Imagen función carga el gráfico circular de monedas

5.6.6. Exportación SQL

Como funcionalidad adicional, se añadió un botón de exportación que permite descargar un script SQL con las filas correspondientes a los datos filtrados en ese momento. La clase ExportacionAnalisis genera las sentencias INSERT necesarias respetando las relaciones entre tablas (personas, protestos, letras, endosos y roles).

Esto ofrece una utilidad doble: por un lado, el investigador puede conservar subconjuntos de datos en su propio entorno; por otro, se facilita la interoperabilidad con otras aplicaciones o bases de datos.

```
mapProtesto(p, acc) {
  // Tabla base "protesto"
  const protestoRow = {
    id_protesto: p.id_protesto,
    fecha_protesto: p.fecha_protesto,
    importe: this.numOrNull(p.importe),
    moneda_nombre: p.moneda_nombre,
    ciudad_nombre: p.ciudad_nombre,
    tipo_protesto_nombre: p.tipo_protesto_nombre,
    tipo_letra_protesto_nombre: p.tipo_letra_protesto_nombre,
    motivo: p.motivo
  };
  this.addRow(acc, 'protesto', protestoRow);

  // Roles protesto
  if (Array.isArray(p.roles)) {
    p.roles.forEach(r => {
      // Exportar persona
      if (r?.persona) { this.addRow(acc, 'persona', this.pick(r.persona, ['id_persona','nombre','apellidos','profesion'])); }
      //Exportar datos rol
      const rolRow = {
        id_protesto: p.id_protesto,
        id_persona: r?.persona?.id_persona ?? r?.id_persona,
        rol_nombre: r?.rol_nombre || r?.nombre_rol || r?.rol,
        ciudad_nombre: r?.ciudad_nombre || r?.ciudad || r?.ciudad_residencia
      };
      this.addRow(acc, 'rol_protesto', rolRow);
    });
  }

  // Mapera letras del protesto
  if (Array.isArray(p.letras)) {
    p.letras.forEach(l => {
      this.mapLetra({ ...l, id_protesto: p.id_protesto }, acc);
    });
  }
}
```

Figura 26 - Imagen obtener datos de protesto para exportación

5.7. Despliegue en Docker y Google Cloud

El despliegue se basa en contenedores: en desarrollo se usan docker-compose para levantar aplicación y base de datos; en producción, la imagen se publica en Artifact Registry y se ejecuta en Cloud Run con conexión a Cloud SQL. Variables de entorno aseguran portabilidad y se incorpora un health check para monitorizar el estado.

```
tfg_protestos > docker-compose.yml > ...
version: '3.8'
Run All Services
services:
  Run Service
  app:
    build: .
    ports:
      - "8080:8080"
    environment:
      - DB_HOST=db
      - DB_USER=root
      - DB_PASSWORD=root_password
      - DB_NAME=protestos
    depends_on:
      - db
    networks:
      - app-network

  Run Service
  db:
    image: mysql:8.0
    environment:
      - MYSQL_ROOT_PASSWORD=root_password
      - MYSQL_DATABASE=protestos
    volumes:
      - ./config/protestos.sql:/docker-entrypoint-initdb.d/protestos.sql
    networks:
      - app-network

networks:
  app-network:
    driver: bridge
```

Figura 27 - Imagen docker compose

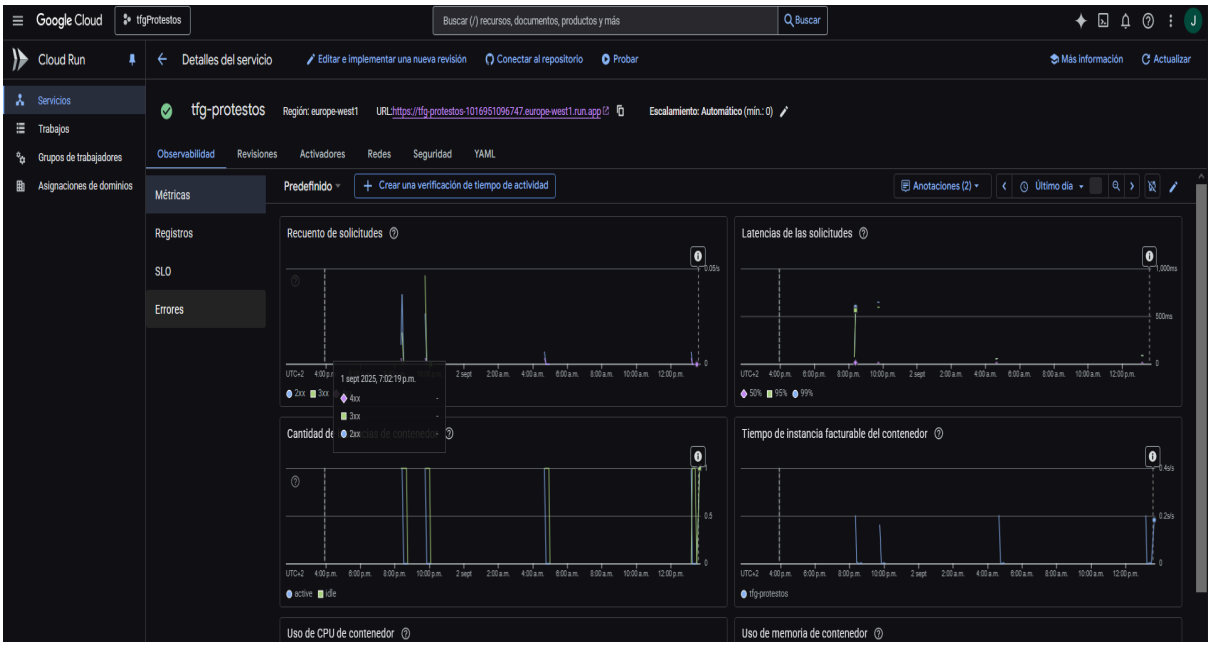
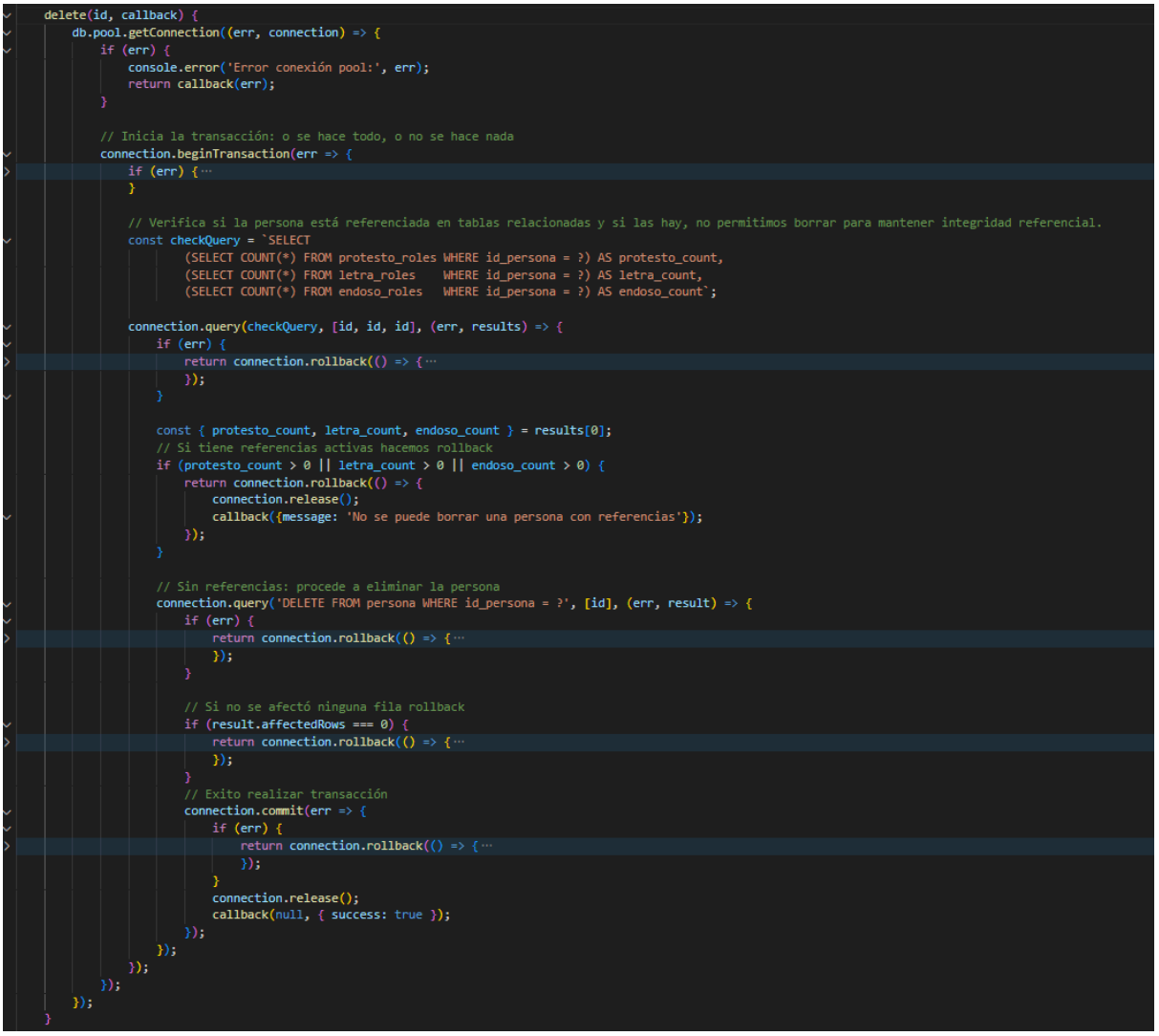


Figura 28 – Imagen consola de comandos de la aplicación web en Google Cloud

5.8. Acceso a datos (DAOs) y transacciones MySQL

Los **DAOs** (Data Access Object) son funciones que acceden a la base de datos fundamentales para mantener una arquitectura por capas. Por otro lado, las transacciones son operaciones que garantizan las que una serie de operaciones sobre una conexión a la base de datos se ejecuten de forma independiente y si se confirman se guardan en la base de datos los cambios (commit), pero si alguna se rechaza, se deshacen todas las operaciones(rollback).

En el caso de la *Figura 24* que contiene la función borrar una persona se obtiene una conexión a la base de datos mediante el pool, luego se inicia la transacción y verifica que esa persona que se intenta borrar no este relacionada con algun rol de un protesto, letra o endoso. Si no está relacionada con nada se borra y se confirma la eliminación con el commit, pero si está relacionada con algún rol, no se puede borrar porque dejaría información errónea en la relación, por lo que se descartan los cambios (rollback).



```
delete(id, callback) {
    db.pool.getConnection((err, connection) => {
        if (err) {
            console.error('Error conexión pool:', err);
            return callback(err);
        }

        // Inicia la transacción: o se hace todo, o no se hace nada
        connection.beginTransaction(err => {
            if (err) {…
            }

            // Verifica si la persona está referenciada en tablas relacionadas y si las hay, no permitimos borrar para mantener integridad referencial.
            const checkQuery = `SELECT
                    (SELECT COUNT(*) FROM protesto_roles WHERE id_persona = ?) AS protesto_count,
                    (SELECT COUNT(*) FROM letra_roles    WHERE id_persona = ?) AS letra_count,
                    (SELECT COUNT(*) FROM endoso_roles   WHERE id_persona = ?) AS endoso_count`;

            connection.query(checkQuery, [id, id, id], (err, results) => {
                if (err) {
                    return connection.rollback(() => {…
                    });
                }

                const { protesto_count, letra_count, endoso_count } = results[0];
                // Si tiene referencias activas hacemos rollback
                if (protesto_count > 0 || letra_count > 0 || endoso_count > 0) {
                    return connection.rollback(() => {
                        connection.release();
                        callback({message: 'No se puede borrar una persona con referencias'});
                    });
                }

                // Sin referencias: procede a eliminar la persona
                connection.query('DELETE FROM persona WHERE id_persona = ?', [id], (err, result) => {
                    if (err) {
                        return connection.rollback(() => {…
                        });
                    }

                    // Si no se afectó ninguna fila rollback
                    if (result.affectedRows === 0) {
                        return connection.rollback(() => {…
                        });
                    }
                    // Exito realizar transacción
                    connection.commit(err => {
                        if (err) {
                            return connection.rollback(() => {…
                            });
                        }
                        connection.release();
                        callback(null, { success: true });
                    });
                });
            });
        });
    });
}
```

Figura 29 - imagen función Borrar Persona en DAOPersona.

Conexión a la base de datos, todos los DAOs usan mysql2 y un pool de conexiones compartido. La configuración de la bd cambia según el entorno, si está en local o está en Google Cloud.

```
db.js
tfg_protestos > config > db.js > ...
const mysql = require('mysql2');

const config = process.env.NODE_ENV === 'production'
  ? {
    // Configuración Cloud SQL
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    database: 'protestos',
    socketPath: `/cloudsql/${process.env.INSTANCE_CONNECTION_NAME}`,
    waitForConnections: true,
    connectionLimit: 10,
    queueLimit: 0,
    connectTimeout: 60000
  }
  : {
      // Configuración XAMPP
      host: 'localhost',
      user: 'root',
      password: '',
      database: 'protestos',
      port: 3306,
      waitForConnections: true,
      connectionLimit: 10,
      queueLimit: 0
    };

const pool = mysql.createPool(config);

const promisePool = pool.promise();

module.exports = {
  pool: pool,
  promisePool: promisePool
};
```

Figura 30 - Código configuración base de datos

5.9. Modelo de datos: del E/R a tablas

El modelo E/R se traduce a un conjunto de tablas relacionales con claves foráneas y restricciones de integridad. Los índices se aplican en campos clave como fechas, ciudad o moneda, lo que permite responder con rapidez a las consultas más habituales. El diseño equilibra fidelidad histórica y eficiencia técnica, manteniendo trazabilidad hacia la fuente original.

```
CREATE TABLE `letracambio` (
  `id_letra` int(11) NOT NULL,
  `fecha_letra` date DEFAULT NULL,
  `fecha_vencimiento` varchar(255) DEFAULT NULL,
  `importe` decimal(10,2) DEFAULT NULL,
  `id_moneda` int(11) DEFAULT NULL,
  `id_tipo_letra` int(11) DEFAULT NULL,
  `id_tipo_plazo` int(11) DEFAULT NULL,
  `id_tipo_valor` int(11) DEFAULT NULL,
  `id_ciudad` int(11) DEFAULT NULL,
  `texto_indicacion` text DEFAULT NULL,
  `plazo_dias` int(11) DEFAULT NULL,
  `comentarios` text DEFAULT NULL,
  `uso` varchar(100) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

--
-- Volcado de datos para la tabla `letracambio`
--

INSERT INTO `letracambio` (`id_letra`, `fecha_letra`, `fecha_vencimiento`, `importe`, `id_moneda`, `id_tipo_letra`, `id_tipo_plazo`, `id_tipo_valor`, `id_ciudad`, `texto_indicacion`, `plazo_dias`, `comentarios`,
(1, '1793-10-01', 'Uso y medio', 796.00, 1, 7, NULL, NULL, 8, NULL, NULL, NULL, NULL),
(2, '1793-02-28', 'A la orden', 5215.00, 3, NULL, NULL, NULL, 526, NULL, NULL, NULL, NULL),
(3, NULL, '90 días fecha', 421.00, 4, 8, NULL, NULL, 2, NULL, NULL, NULL, NULL),
(4, '1792-11-12', '90 días fecha', 72.00, 8, NULL, NULL, NULL, 675, NULL, NULL, NULL, NULL),
(5, '1792-10-12', '90 días fecha', 83.12, 9, NULL, NULL, NULL, 675, NULL, NULL, NULL, NULL),
(6, '1792-10-12', '90 días fecha', 91.22, 10, NULL, NULL, NULL, 675, 'La primera en casa de los señores Courant Bridel y Cia\n', NULL, NULL, NULL),
```

Figura 31 - Imagen tabla letracambio en sql

5.10. Pruebas automatizadas en Jest

El conjunto de pruebas cubre servicios, persistencia y vistas principales:

- Servicios: SAFormulario.test.js verifica reglas de negocio (fechas coherentes, moneda obligatoria, roles mínimos) y rutas felices/erróneas del guardado.
- Persistencia: formulario.persistence.test.js mockea el pool MySQL; fuerza fallos intermedios para asegurar rollback.
- Vistas: formulario_vista.test.js monta la EJS con un DOM simulado y comprueba la presencia y orden de campos, así como validaciones esenciales.

Se automatiza en CI/CD para cada push.

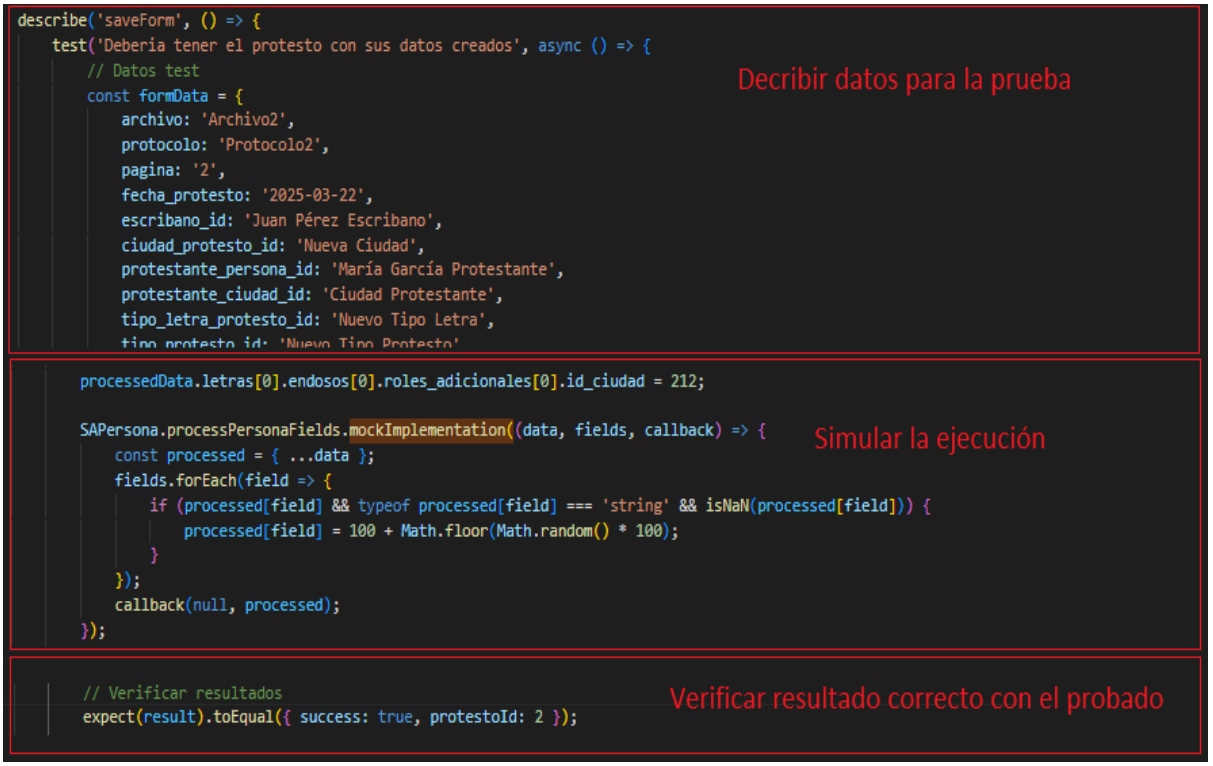


Figura 32 - Imagen explicación test de jest en SAFormulario.test.js

Capítulo 6 - Conclusiones y trabajo futuro

Conclusiones

Este Trabajo de Fin de Grado ha dado como resultado una aplicación web que cumple los objetivos iniciales y ofrece al investigador una herramienta práctica para digitalizar, gestionar y analizar protestos de letras de cambio.

En primer lugar, se ha diseñado un modelo de datos relacional que representa con claridad protestos, letras, endosos y el resto de entidades relacionadas, garantizando la integralidad y la trazabilidad de los datos, indispensable para gestionar y analizar miles de protestos.

En segundo lugar, se ha implementado un formulario de entrada, que aunque manual, sus funciones de autocompletado, validación y tabulación convierten la tarea de registro en un proceso más ágil y consistente. Además esta implementación puede ser la base para un futuro sistema OCR que pre-rellene automáticamente datos en el formulario.

En tercer lugar, se ha desarrollado un módulo de análisis y visualización que permite explorar los datos a través de tablas, varios tipos de gráficos, mapas interactivos y grafos, el investigador puede observar patrones, comparar magnitudes y detectar conexiones que serían difíciles de identificar mediante un análisis manual. Aunque en esta primera versión el análisis se centra en documentos, ciudades y monedas, la arquitectura implementada sienta las bases para ampliar estas funcionalidades a cualquier entidad o campo de la base de datos. Además el investigador puede exportar en sql sus consultas.

La aplicación, además, adopta una arquitectura en capas que facilita su mantenimiento y escalabilidad. El uso de pruebas automatizadas con Jest refuerza la fiabilidad del sistema, mientras que el despliegue mediante contenedores Docker en Google Cloud garantiza la portabilidad y el acceso desde distintas ubicaciones y dispositivos.

En definitiva, este Trabajo de Fin de Grado ha dado lugar a una herramienta estable y útil que combina digitalización, gestión y análisis de protestos de letras de

cambio en un único entorno. Su diseño modular, basado en un modelo de datos relacional, un formulario optimizado y un conjunto de visualizaciones interactivas, proporciona al investigador una solución práctica que mejora de manera notable la productividad frente a métodos manuales tradicionales.

La aplicación cumple con los objetivos planteados, y además sienta unas bases sólidas para su evolución futura. Gracias a la arquitectura en capas, el despliegue en la nube y la interoperabilidad con herramientas externas, el sistema puede crecer en funcionalidades sin comprometer la estabilidad. A partir de aquí, el reto no es tanto construir la infraestructura básica (ya consolidada), sino avanzar hacia una mayor automatización de la digitalización, enriquecer las capacidades analíticas y habilitar un mayor uso colaborativo.

Trabajo futuro

Aunque los resultados alcanzados son satisfactorios, la aplicación abre la puerta a diversas líneas de mejora que podrían enriquecerla en futuras fases:

1. **OCR + extracción asistida por IA**

   Integrar herramientas de reconocimiento de manuscritos como Transkribus para convertir las imágenes de protestos manuscritos en texto digitalA partir de este texto, un modelo de lenguaje (LLM) podría pre-rellenar el formulario, de esa manera el investigador solo tenga que revisar si han sido introducidos correctamente. También sería interesante resaltar visualmente en el texto digitalizado de dónde se han extraído los valores de cada campo.

2. **Análisis más profundo y flexible**

   Actualmente se pueden analizar protestos, letras, endosos, ciudades y monedas, pero todavía queda pendiente ampliar el módulo de análisis a las entidades de personas y roles.

   Por otro lado, los filtros y visualizaciones ofrecen un gran margen de mejora: lo ideal sería que cada campo de cualquier entidad pudiera analizarse de

forma independiente o en combinación con otros. Con ello, el investigador podría construir vistas analíticas avanzadas sin necesidad de modificar el código.

3. **Exportación en más formatos**

   Además de SQL, convendría añadir JSON y CSV. JSON para intercambios con otras herramientas como Gephi y CSV para hojas de cálculo rápidas. La implementación no sería compleja, ya que se construiría sobre la exportación en sql que hay actualmente.

4. **Trabajo colaborativo y control de acceso**

   La incorporación de gestión de usuarios, roles y permisos (editor, revisor, solo lectura…), añadir historial de cambios (quién editó qué y cuándo) y comentarios en línea. De esta manera se podría trabajar en equipo o colaborar con otros investigadores con trazabilidad y auditoría.

5. **Modelo de datos adaptable por el usuario**

   Permitir que el investigador cree o modifique entidades y campos desde la interfaz, sin intervención de un programador. Esto facilita aplicar la herramienta a otros corpus (p. ej., escrituras notariales, padrones, libros de contabilidad) manteniendo el núcleo técnico. Es un cambio potencialmente muy útil inspirado en plataformas como Salesforce o Odoo.

6. **Personalización de formularios y vistas**

   Otra línea de mejora sería permitir al investigador crear o modificar entidades y campos directamente desde la interfaz, sin programador. Este enfoque, inspirado en plataformas como Salesforce u Odoo, permitiría editar documentos actuales o crear otros tipos de documentos históricos com testamentos, préstamos o libros de contabilidad sin alterar el núcleo de la aplicación.

7. **Importación de datos**

   Finalmente, junto a la exportación, sería conveniente ofrecer la posibilidad de importar datos en formatos como CSV o JSON. Esto permitiría integrar fácilmente información ya digitalizada en otros contextos o proyectos.

En conjunto, estas líneas de trabajo posibilitarían que la aplicación no solo mantenga su utilidad a corto plazo, sino que pueda crecer como un entorno integral para la investigación histórica.

Repositorio de la aplicación:
https://github.com/jjencina/tfg-digitalizacion-protestos.git

Conclusions and future work

Conclusions

This project delivers a web application that turns historical protestos key to understanding the late-eighteenth-century crisis in Galicia into structured, queryable, and analyzable data. The relational model faithfully captures the domain (bills, endorsements, protests, people, roles, cities, currencies, and typologies) and enforces referential integrity, with the attempt to improve traceability and data quality. The application is uses an optimized data-entry workflow speeding up digitization while reducing errors and duplication. On this basis, the analysis module application offers tables, charts, interactive maps, and network views that open new avenues for inquiry in the field of academic Economic History.

From a software-engineering perspective, the system follows a clear layered architecture—Node.js and Express for the backend, EJS and Bootstrap for the presentation layer, and MySQL for persistence—containerized with Docker and deployed on Google Cloud. Automated tests with Jest strengthen reliability and make future evolution safer. Overall, the tool meets the stated goals: it is usable in practice, technically solid, and illustrates how computational methods can unlock the research potential of dispersed historical sources.

Future Work

Notwithstanding the fact that the results achieved are satisfactory, the application opens the door to several lines of improvement that could enrich and supplement it in future stages:

1. HTR/OCR integration. Add handwriting recognition to extract fields directly from scans of protests, reducing manual entry and speeding up academic efforts and performance.
2. Increased analytical functionality: add modules for advanced statistics, pattern detection algorithms, and automated report generation to enable more specific conclusions to be drawn from the data.
3. Interface and user friendly environment: improve the user experience through a more intuitive and accessible design, including smart forms, autocomplete options, and mass-editing features for the application.
4. The application is able to be adapted to other contexts: extend the application to other languages or adapt the data model to different documentary sources, thereby broadening its scope to research in other regions or historical periods.
5. Multi-user collaborative work: evolve the application towards a platform with user viewer and role management that enables team-based digitization and analysis, fostering collaborative work among historians, economists, and computer scientists.
6. Integration with external systems: establish direct connections with applications such as Gephi, QGIS, or R, enabling real-time interactive analysis without the need for intermediate exports.

Taken together, these lines of work ensure that the application will not only remain useful in the short term but can grow into a comprehensive environment for academic historical research, consolidating itself as a solid example of the convergence between computer science and the social sciences

ANEXOS

ANEXO 1 - MANUAL DE USUARIO

Pantalla principal

Pantalla principal permite acceder a los tres módulos principales:

- Añadir un protesto para digitalizar manualmente los documentos
- Gestión de Datos es el CRUD de todas las entidades de la aplicación
- Análisis de datos es el módulo donde se puede explorar y exportar la información

Figura 33 - Imagen ventana principal del programa

Formulario (Añadir protestos)

El formulario es la pantalla principal para introducir manualmente protestos completos, con sus letras y endosos. Replica la estructura de los documentos de protesto reales para introducir los datos en la aplicación a medida que se leen.
Al entrar al formulario se cargan los datos y se crean por defecto una letra y un endoso y, se autorellenan ciertos campos con datos del último protesto(archivo, protocolo, escribano, ciudad del protesto).

La mayoría de campos tienen búsqueda y autocompletado en vivo para agilizar, además tiene ayudas visuales para identificar datos nuevos o datos ya

guardados, en azul se muestran datos nuevos, en verde se muestran datos que ya incluidos en la base de datos.

Ha sido optimizado mediante una tabulación optimizada para poder introducir los datos usando únicamente el teclado y evitando al máximo el uso del ratón, volviéndolo mucho más ágil.

Los botones “Añadir nuevo campo” permiten añadir un nuevo rol y al documento o un nuevo comentario.

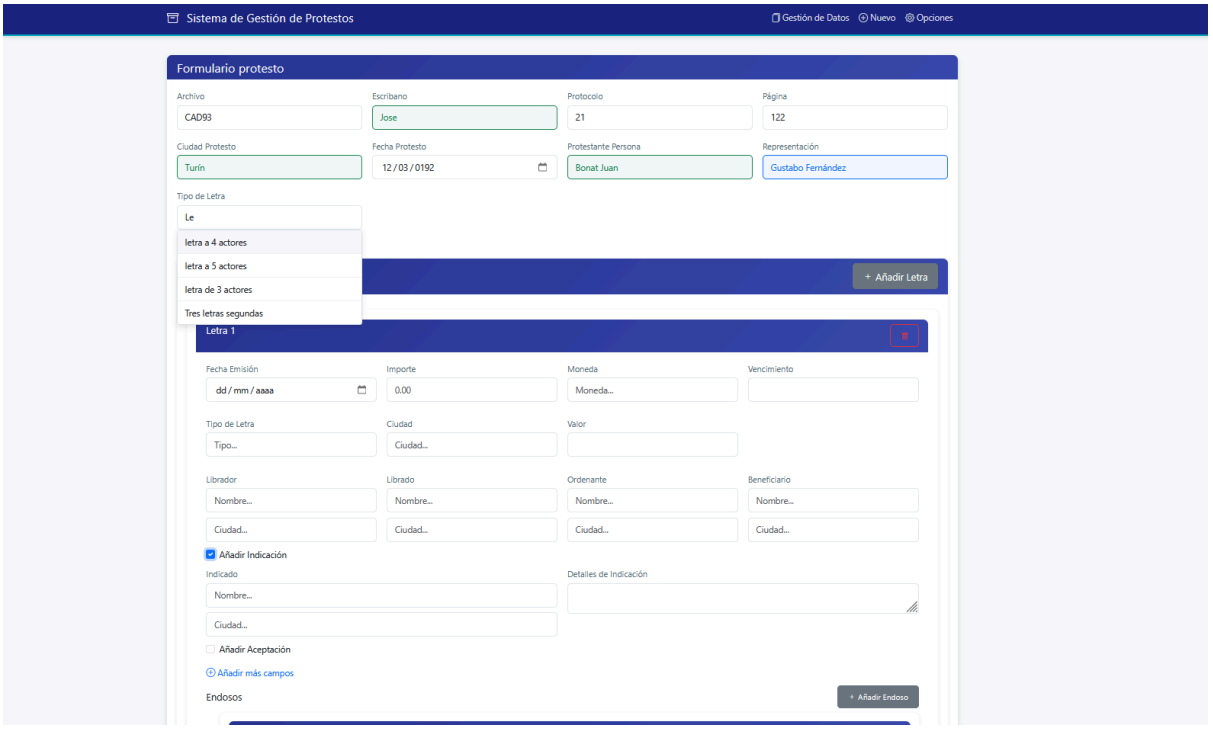


Figura 34 – Imagen formulario del protesto

Gestión de datos

Consiste en un módulo de gestión que permite administrar todas las entidades y relaciones almacenadas en la base de datos. Esta sección está concebida como una interfaz CRUD (crear, leer, actualizar y eliminar), a la que se accede desde el menú principal, y que ofrece listados de todas las entidades y

relaciones que se usan: protestos, personas, ciudades, monedas, tipos de letras o protestos, entre otros.

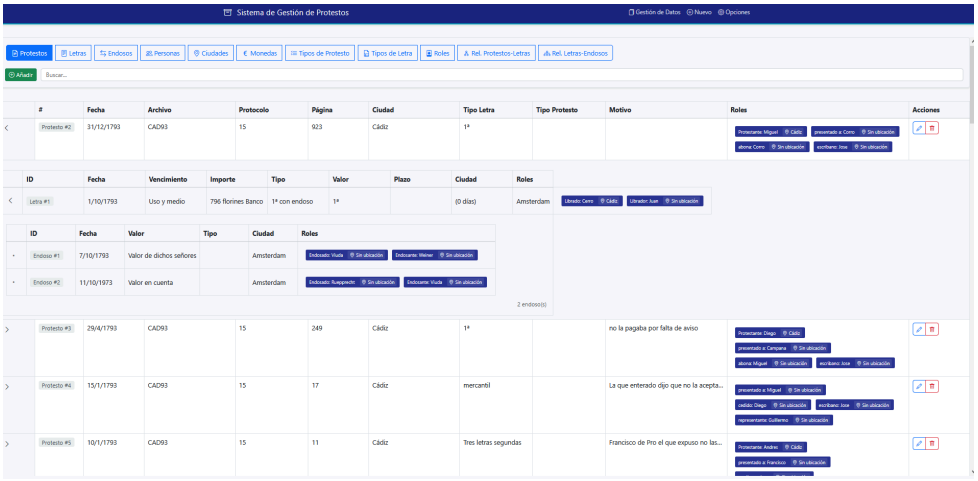


Figura 35 – Imagen ventana de gestión, tabla de entidades y botones CRUD

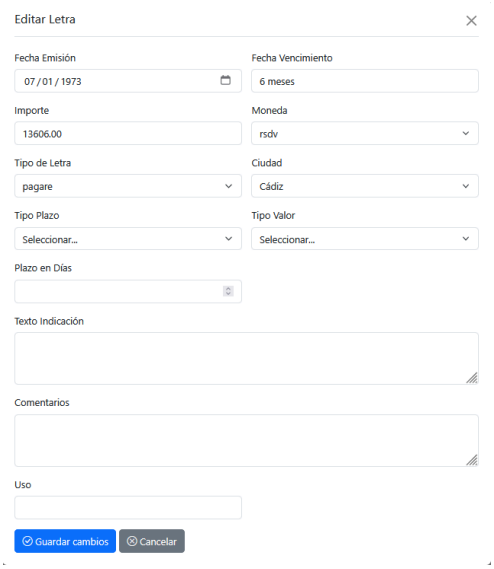


Figura 36 – Imagen ventana modal editar Letra de Cambio

Análisis de datos

El módulo de análisis representa la “ventana” principal para la explotación de la información ya digitalizada. Desde una única vista, el investigador puede aplicar filtros, consultar tablas, generar gráficas, explorar mapas interactivos y visualizar

grafos de relaciones. Todos estos componentes comparten un mismo estado de filtros, de manera que los resultados se actualizan de forma sincronizada.

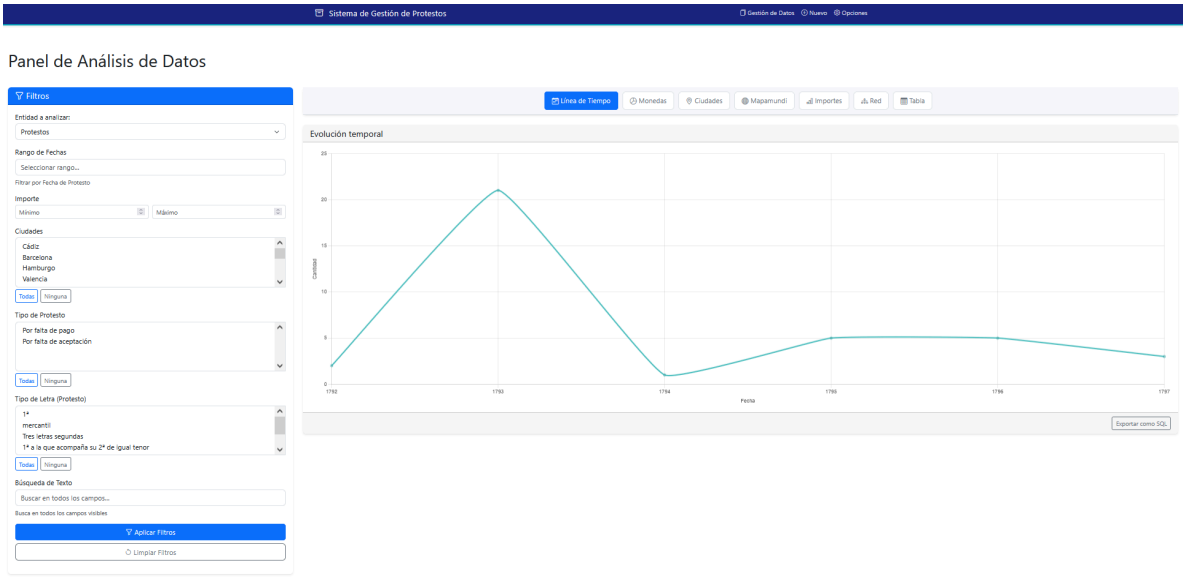
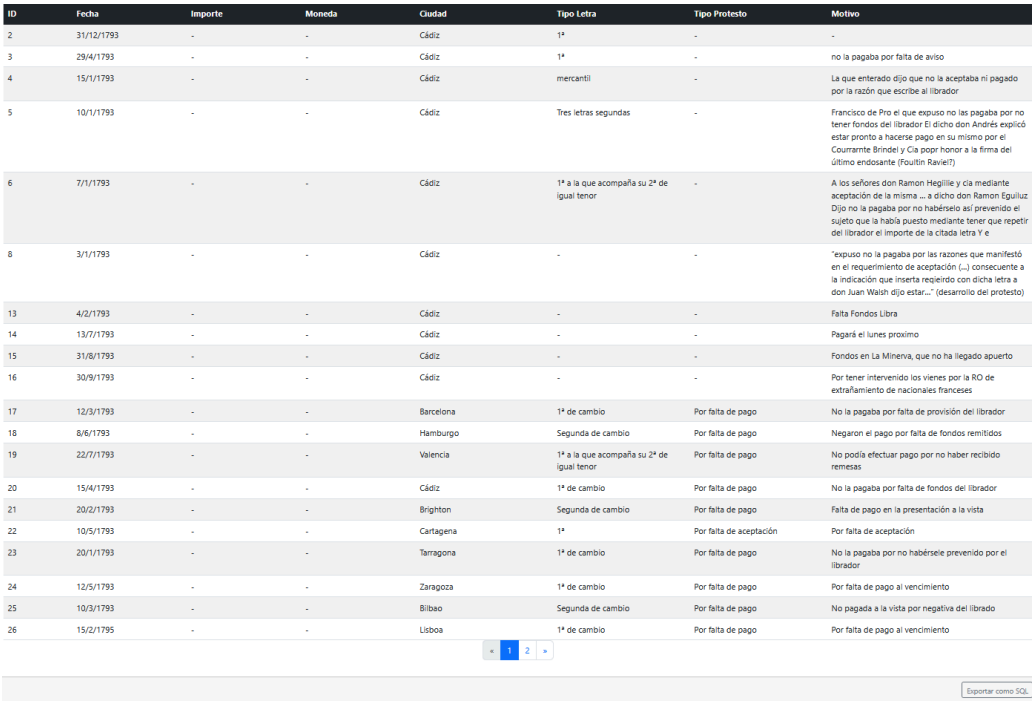


Figura 37 – Imagen panel de análisis de datos

Tablas

Las tablas permiten examinar campos y contadores de protestos, letras y endosos que hayan sido filtrados por el usuario, además la función de exportación permite guardar las entidades de la tabla en formato sql.



ID	Fecha	Importe	Moneda	Ciudad	Tipo Letra	Tipo Protesto	Motivo
2	31/12/1793	-	-	Cádiz	1ª	-	-
3	29/4/1793	-	-	Cádiz	1ª	-	no la pagaba por falta de aviso
4	15/1/1793	-	-	Cádiz	mercantil	-	La que enterado dijo que no la aceptaba ni pagado por la razón que escribe al librador
5	10/1/1793	-	-	Cádiz	Tres letras segundas	-	Francisco de Pro el que expuso no las pagaba por no tener fondos del librador El dicho don Andrés explicó estar pronto a hacerse pago en su mismo por el Courrarnte Brindel y Cia popr honor a la firma del último endosante (Foultin Raviel?)
6	7/1/1793	-	-	Cádiz	1ª a la que acompaña su 2ª de igual tenor	-	A los señores don Ramon Hegillie y cia mediante aceptación de la misma ... a dicho don Ramon Eguiluz Dijo no la pagaba por no habérselo así prevenido el sujeto que la había puesto mediante tener que repetir del librador el importe de la citada letra Y e
8	3/1/1793	-	-	Cádiz	-	-	"expuso no la pagaba por las razones que manifestó en el requerimiento de aceptación (...) consecuente a la indicación que inserta reqieirdo con dicha letra a don Juan Walsh dijo estar..." (desarrollo del protesto)
13	4/2/1793	-	-	Cádiz	-	-	Falta Fondos Libra
14	13/7/1793	-	-	Cádiz	-	-	Pagará el lunes proximo
15	31/8/1793	-	-	Cádiz	-	-	Fondos en La Minerva, que no ha llegado apuerto
16	30/9/1793	-	-	Cádiz	-	-	Por tener intervenido los vienes por la RO de extrañamiento de nacionales franceses
17	12/3/1793	-	-	Barcelona	1ª de cambio	Por falta de pago	No la pagaba por falta de provisión del librador
18	8/6/1793	-	-	Hamburgo	Segunda de cambio	Por falta de pago	Negaron el pago por falta de fondos remitidos
19	22/7/1793	-	-	Valencia	1ª a la que acompaña su 2ª de igual tenor	Por falta de pago	No podía efectuar pago por no haber recibido remesas
20	15/4/1793	-	-	Cádiz	1ª de cambio	Por falta de pago	No la pagaba por falta de fondos del librador
21	20/2/1793	-	-	Brighton	Segunda de cambio	Por falta de pago	Falta de pago en la presentación a la vista
22	10/5/1793	-	-	Cartagena	1ª	Por falta de aceptación	Por falta de aceptación
23	20/1/1793	-	-	Tarragona	1ª de cambio	Por falta de pago	No la pagaba por no habérsele prevenido por el librador
24	12/5/1793	-	-	Zaragoza	1ª de cambio	Por falta de pago	Por falta de pago al vencimiento
25	10/3/1793	-	-	Bilbao	Segunda de cambio	Por falta de pago	No pagada a la vista por negativa del librado
26	15/2/1795	-	-	Lisboa	1ª de cambio	Por falta de pago	Por falta de pago al vencimiento

« 1 2 »

Exportar como SQL

Figura 38 – Imagen tabla en la ventana de análisis

Filtros

Los filtros permiten seleccionar ciertas entidades como protesto, letras de cambio, endosos, personas y ciudades filtrándolos según rangos de fechas, importes, ciudades, monedas, tipos de letra o protesto y búsquedas por texto de todos las entidades seleccionadas.

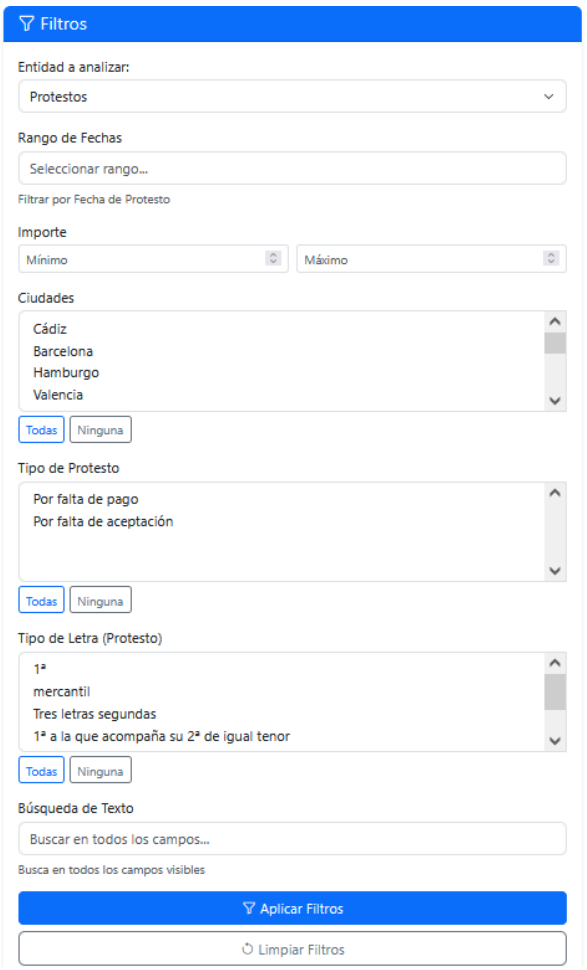


Figura 39 – Imagen filtros de la ventana de análisis

Mapa interactivo

El mapa interactivo, desarrollado con Leaflet, muestra las ciudades según la intensidad de Número de documentos firmados en esa ciudad, la cantidad de dinero o número de documentos y personas que participan en documentos desde la ciudad. Cada ciudad aparece representada con un marcador que despliega información al hacer seleccionar, como el número de documentos asociados y la suma de importes. Opcionalmente, pueden visualizarse conexiones entre ciudades, representando rutas comerciales o financieras, cuyo grosor y color indican la intensidad de la conexión entre las dos ciudades.

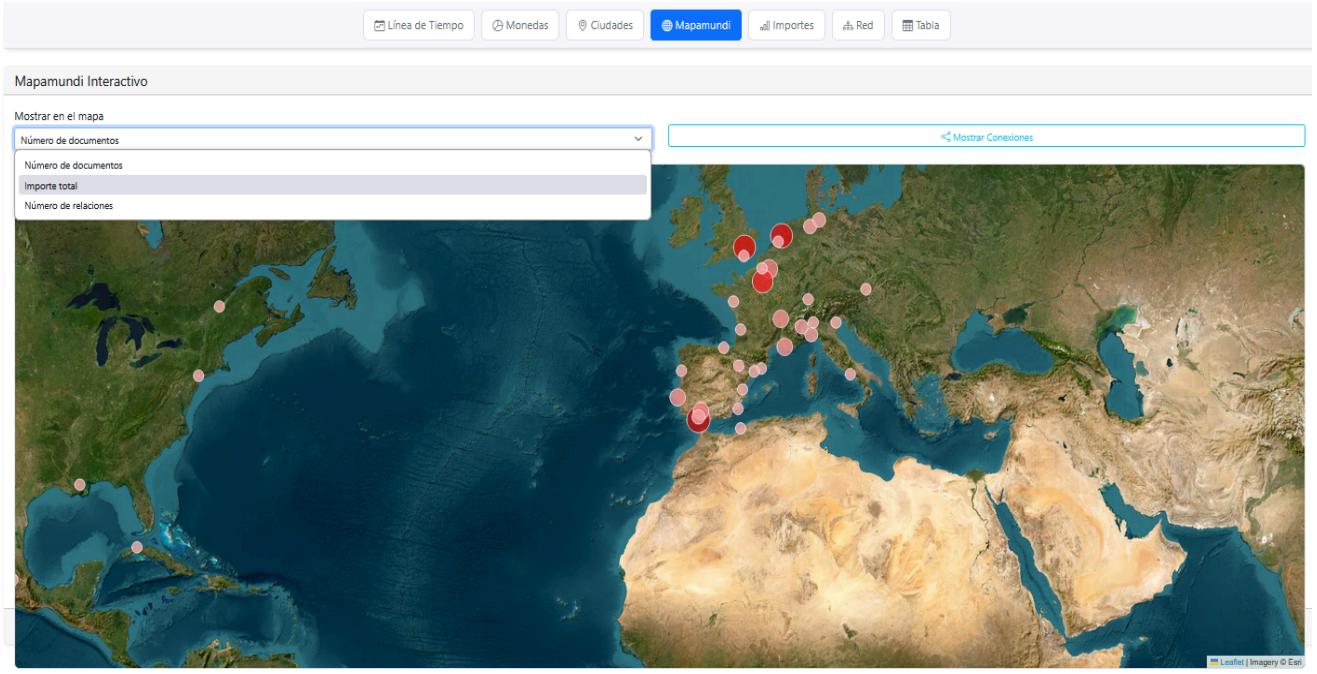


Figura 40 – Imagen mapa interactivo ventana de análisis

Grafo

El grafo, implementado en D3.js, permite analizar las redes de relaciones, ya sea entre ciudades o entre personas y lugares. Los nodos se distribuyen dinámicamente, y su tamaño y color reflejan el grado de conexión. De esta forma, se identifican con facilidad los actores o centros urbanos con mayor protagonismo en la red. Al seleccionar un nodo se puede observar con que otros nodos tiene relación y cuanta.

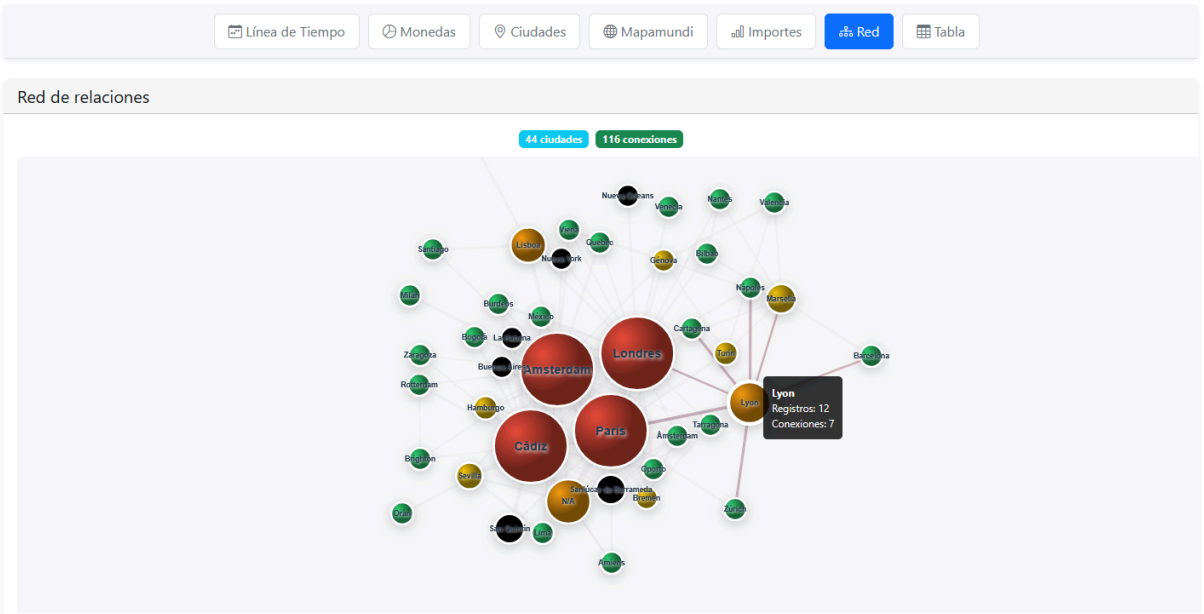


Figura 41 – Imagen red de relaciones en la ventana de análisis

Gráficas

Las gráficas están elaboradas con Chart.js complementan el filtrado mediante el análisis con representación temporal, comparativa por distribución de monedas y por importes en un periodo.

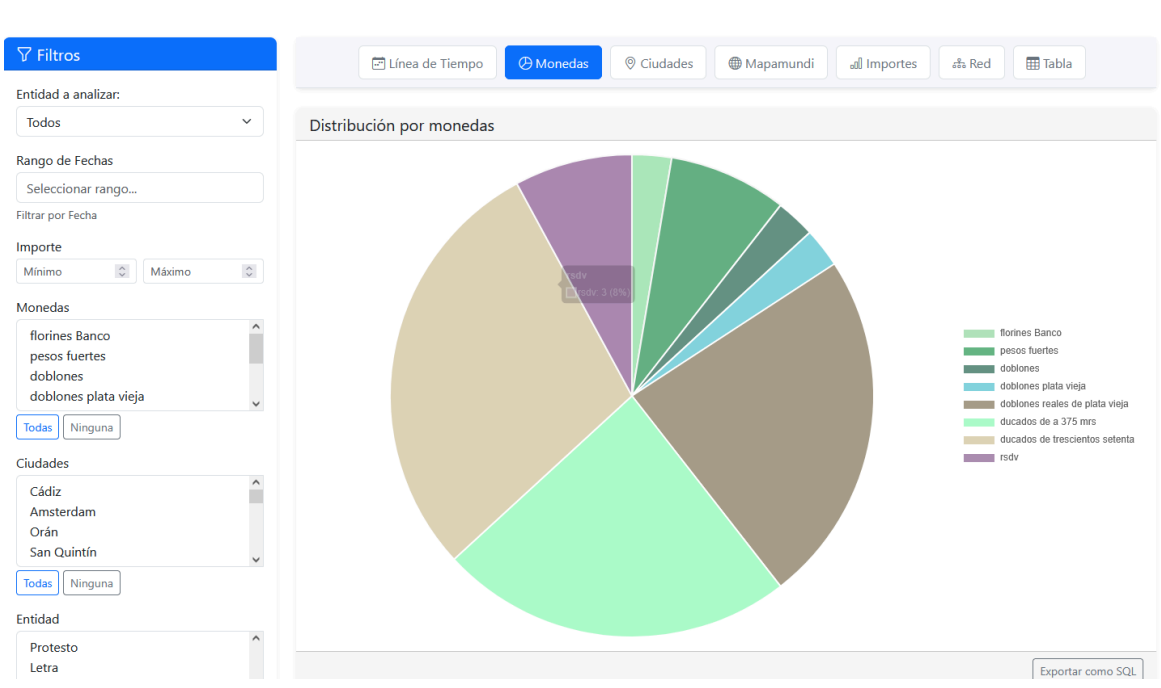


Figura 42 – Imagen gráfica distribución de monedas

Exportar sql

El módulo incorpora también un sistema de exportación que permite descargar los resultados en formatos SQL, respetando los filtros aplicados. De este modo, los investigadores pueden continuar su análisis en otras herramientas de visualización.

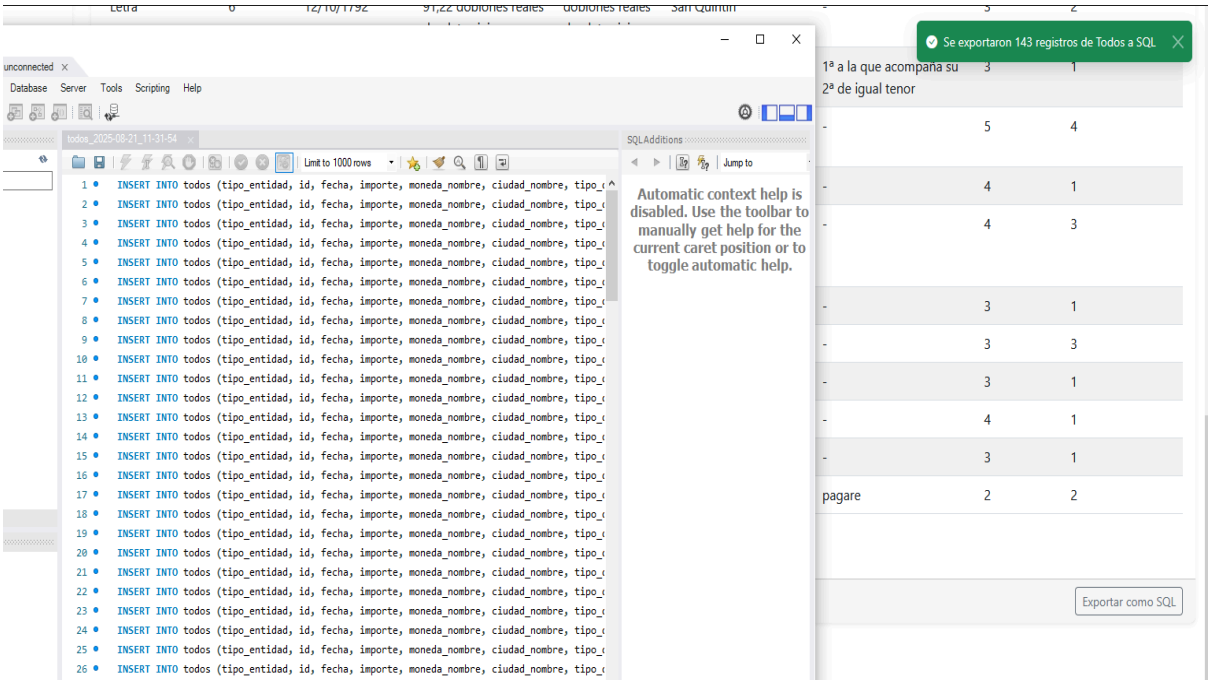


Figura 43– Imagen de proceso y resultado de exportación en sql de una tabla

ANEXO 2 - EJEMPLO DE USO DE USUARIO

Añadir protesto, editar ciudad, realizar análisis ¿En qué ciudades se usaban los ducados durante el 1793? y guardar el análisis en sql

Quiero añadir un protesto con una letra y dos endosos asociados.

Añado datos que ya estaban incluidos en la base de datos que se muestran en verde y los datos nuevos que no existían se muestran en azul.

Creo desde el formulario un tipo de letra y una moneda nuevas

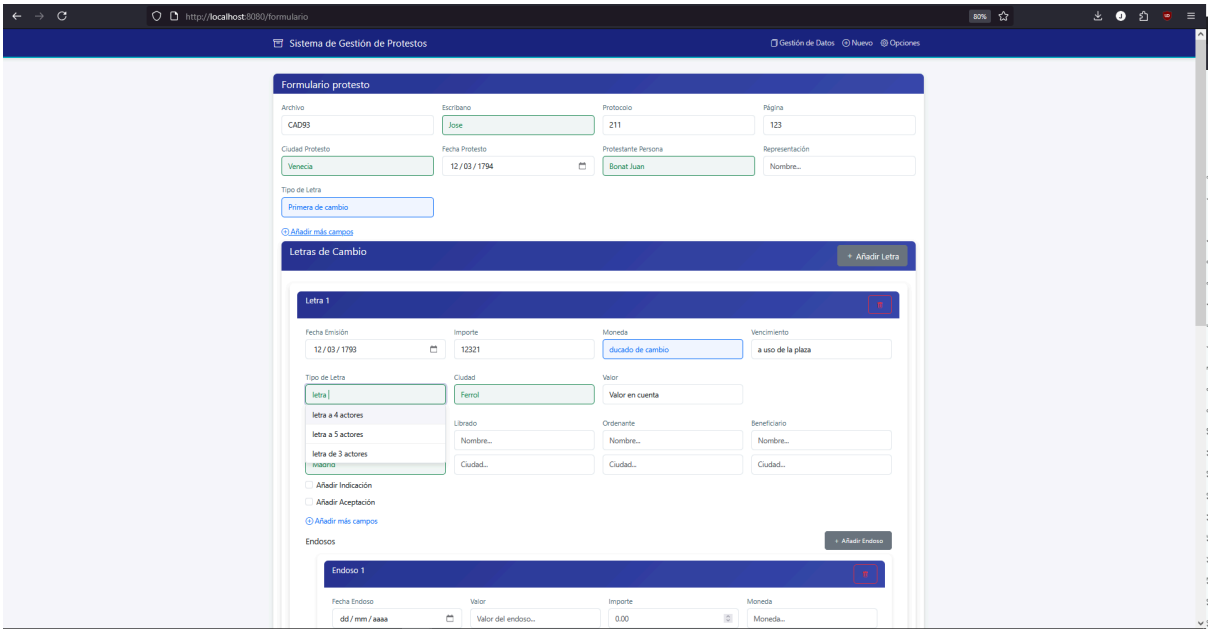


Figura 44 – Imagen proceso de guardado de un protesto

Añado un rol extra a la Letra 1,el rol Aviso

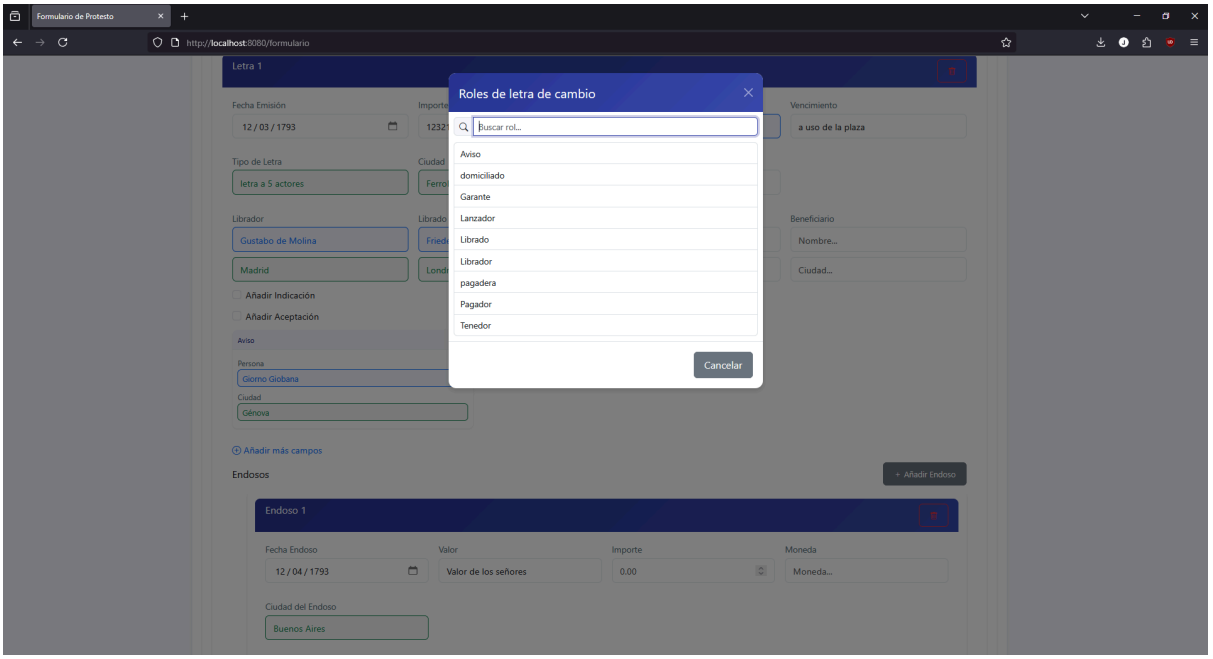


Figura 45 – Imagen añadir a la Letra 1 un campo adicional de rol “Aviso”

Una vez guardado lo veo en la sección “Gestión de datos”

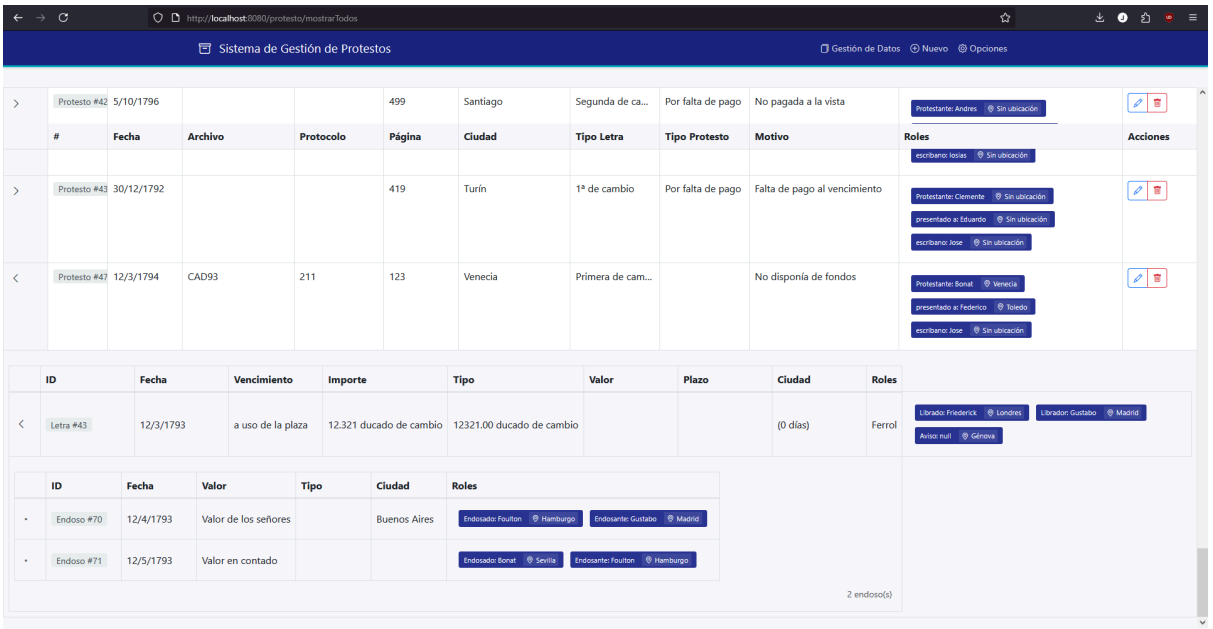


Figura 46 – Imagen Tabla sección “Gestión” Protesto, letra y endosos guardados

Y edito el protesto para cambiar la ciudad de Venecia a Madrid

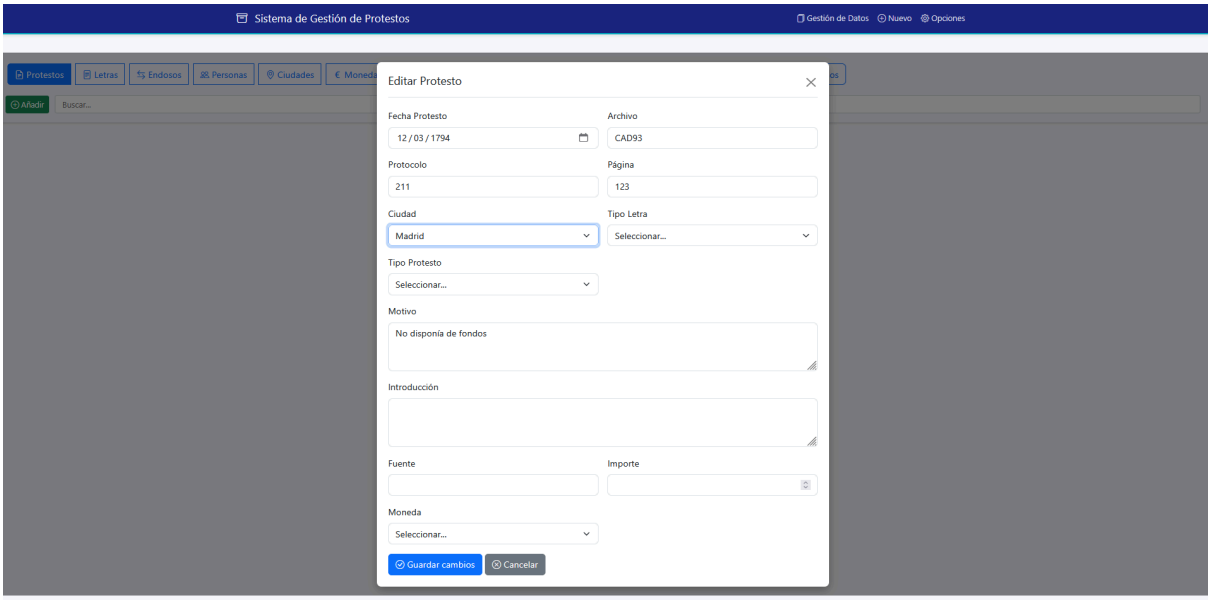


Figura 47 – Imagen edición protesto en sección de “Gestión”

A continuación para analizar los datos seleccionó la sección de “Análisis de Datos” los Selección de filtros, fechas del 01/01/1792 al 31/12/1793 y monedas todos los tipos de dudados

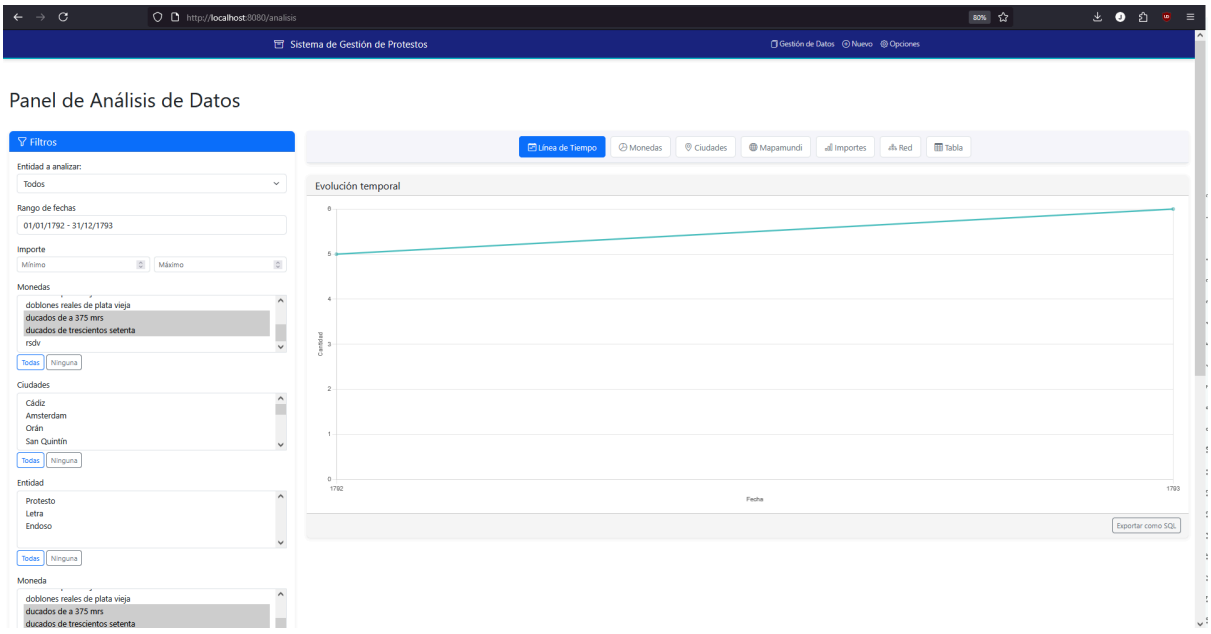


Figura 48 – Imagen sección Análisis. Filtros y gráfica temporal con número de documentos

Ubicación de las ciudades

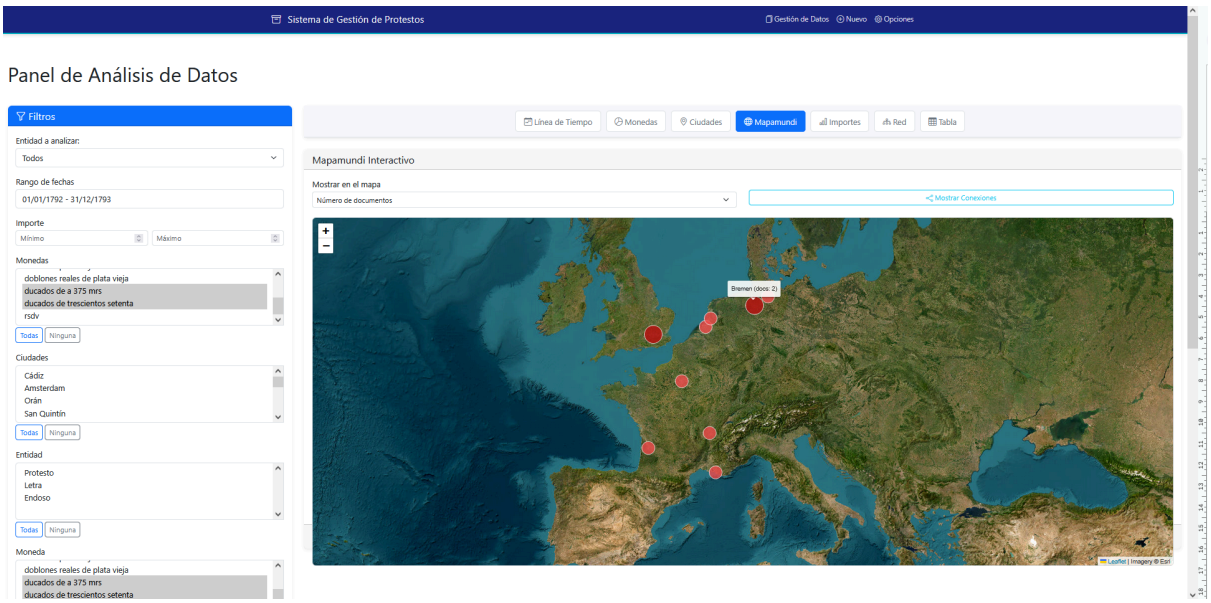


Figura 49 – Imagen sección Análisis. Mapa ciudades con según cantidad de documento

Que ducado se usaba más

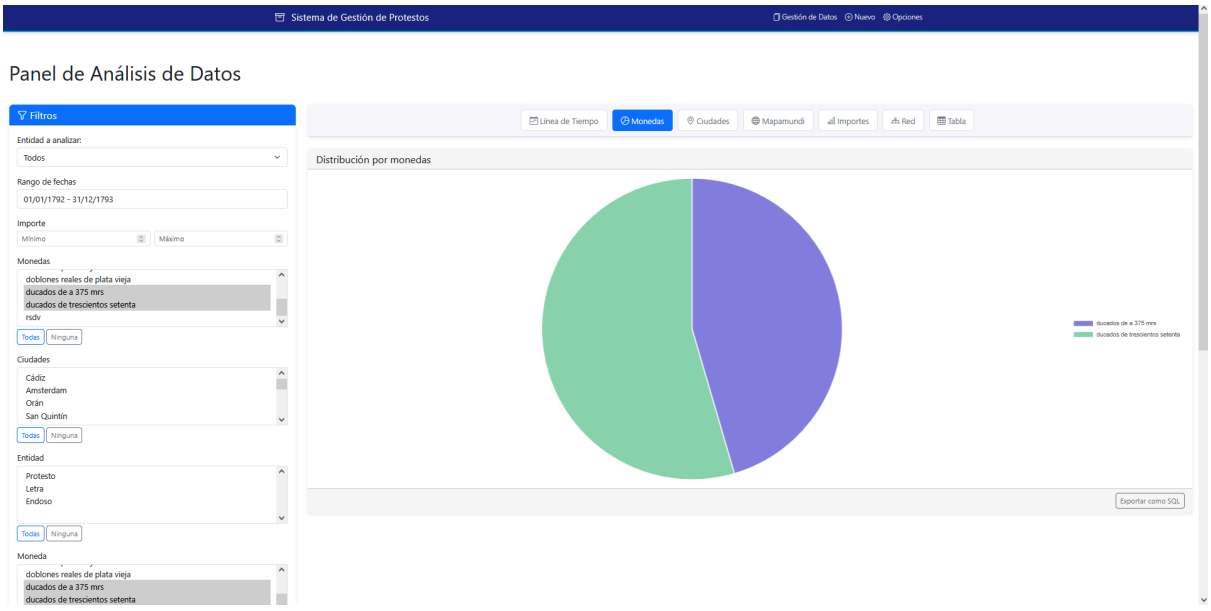


Figura 50 – Imagen sección Análisis. gráfico de tarta con peso de cada tipo de moneda

Entre las ciudades que participaban en documentos en los que se usaban doblones, tenían más conexiones

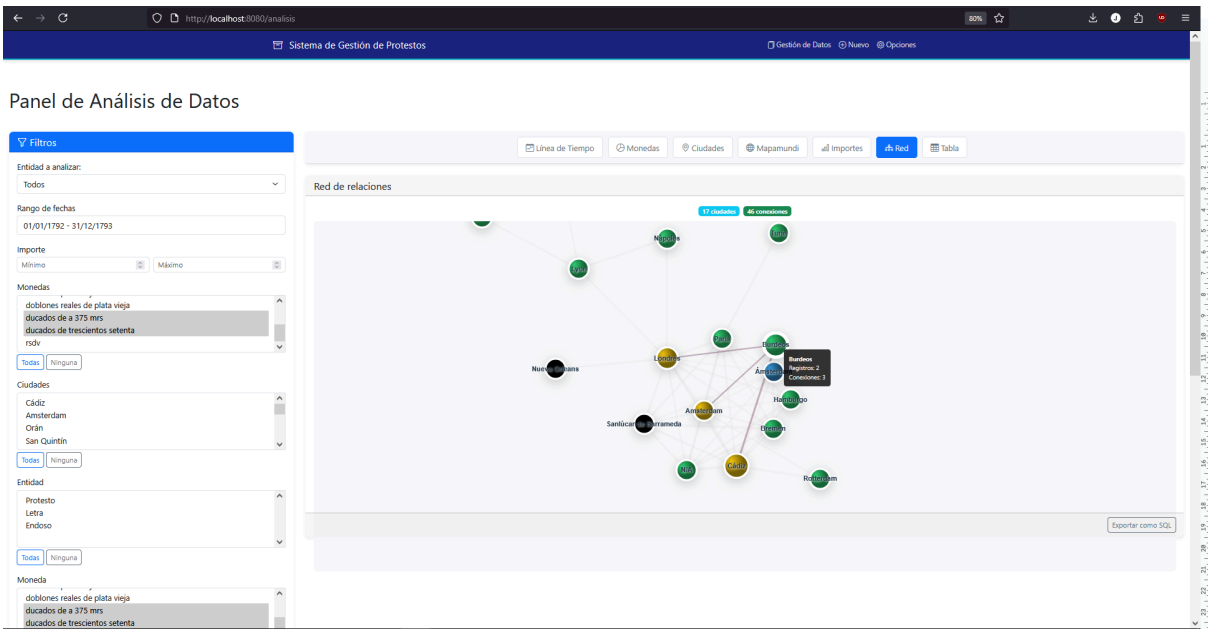


Figura 51 – Imagen sección Análisis. Grafo ciudades conectadas en los documentos seleccionados

Tabla de los Documentos que usan doblones:

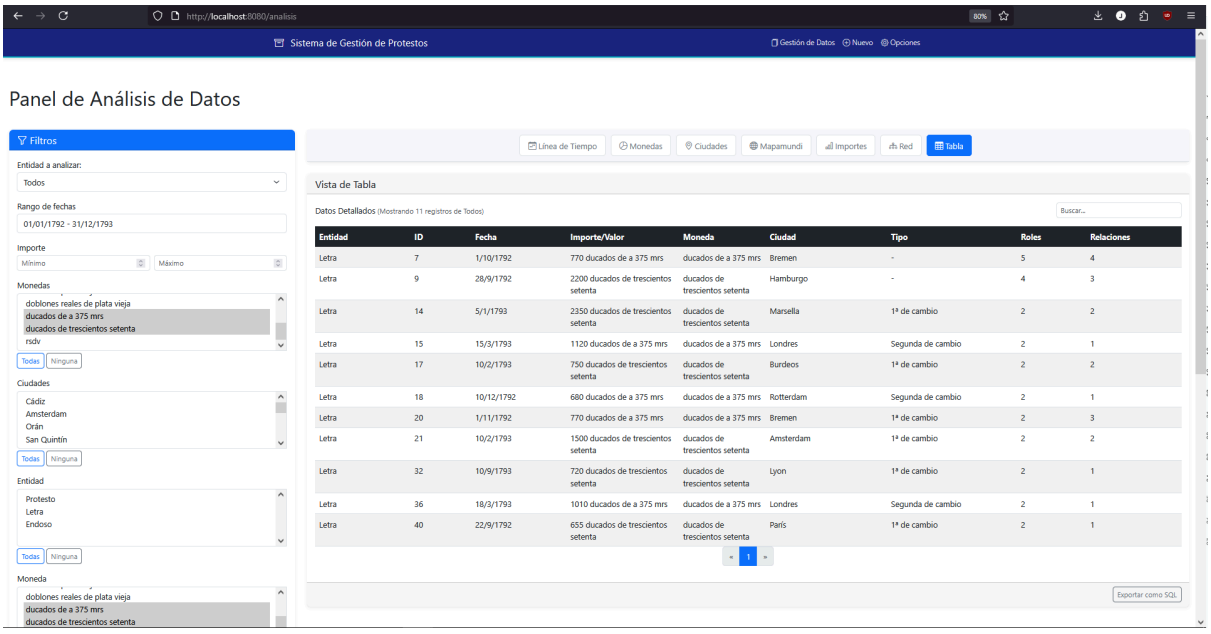


Figura 52 – Imagen sección Análisis. Tabla datos de los documentos seleccionados

Exportar documentos de la consulta en sql:

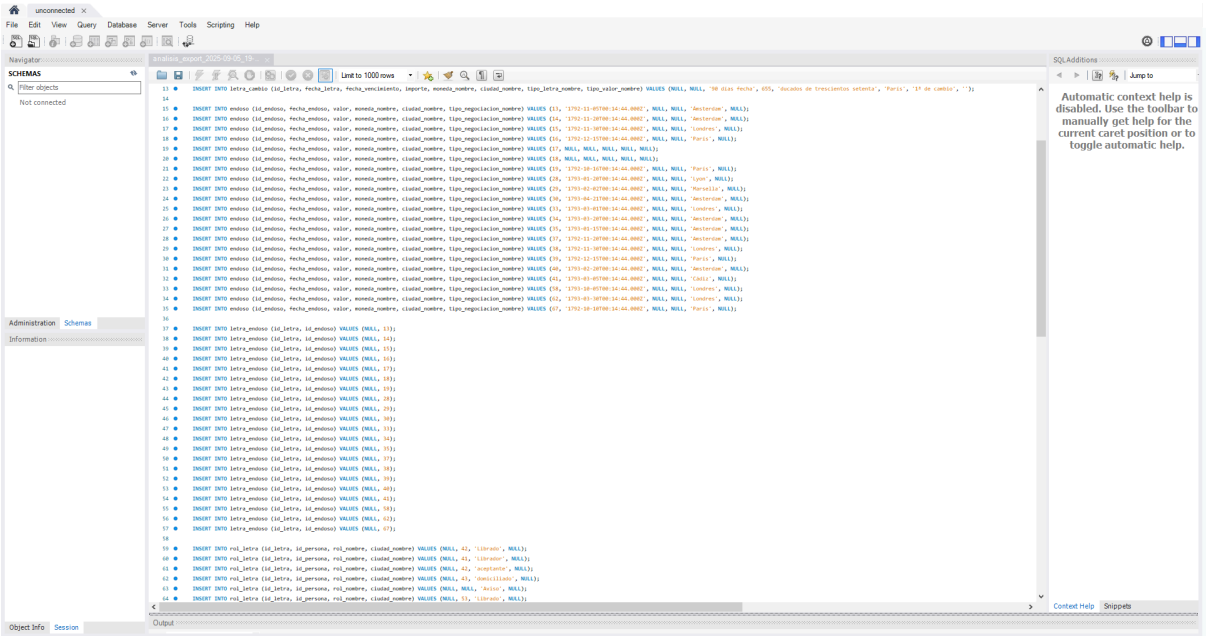


Figura 53 – Imagen sección Análisis. Consulta de documentos exportada en sql

Bibliografía

Libros y documentos compartidos por Francisco acerca de los protestos y el tema a estudiar

[1] L. Caruana de las Cagigas y C. Larrinaga Rodríguez, “De comerciantes, pagos y crisis: elementos para una unidad didáctica en torno a la letra de cambio,” en Superando el COVID-19 en las aulas de Historia Económica, cap. 4.3. Material facilitado y explicado por F. Cebreiro Ares, s. l., s. f.

[2] A. S. Ribeiro, F. L. Pinheiro, F. C. Santos, A. Polónia y J. M. Pacheco, “Structural and temporal patterns of the first global trading market,” Royal Society Open Science, vol. 5, no. 8, p. 180577, 2018, doi: 10.1098/rsos.180577.

[3] O. Accominotti, D. Lucena-Piquero y S. Ugolini, “The origination and distribution of money market instruments: sterling bills of exchange during the first globalization,” Economic History Review, vol. 74, no. 4, pp. 892–921, 2021.

[4] I. Schnabel y H. S. Shin, “Liquidity and Contagion: The Crisis of 1763,” Journal of the European Economic Association, vol. 2, no. 6, pp. 929–968, 2004.

[5] M. Flandreau, C. Galimard, C. Jobst y P. Nogués-Marco, “Monetary geography before the Industrial Revolution,” Cambridge Journal of Regions, Economy and Society, vol. 2, no. 2, pp. 149–171, 2009, doi: 10.1093/cjres/rsp009.

Plataformas y proyectos del estado de la cuestión

[6] Europeana: europeana.eu

[7] PARES: cultura.gob.es

[8] Transkribus: transkribus.org

[9] Omeka: omeka.org

[10] Heurist Network: heurist.huma-num.fr

[11] Nodegoat: nodegoat.net

[12] Gephi: gephi.org

[13] Cytoscape: cytoscape.org

[14] QGIS: qgis.org

Tecnologías empleadas

[15] Node.js: nodejs.org

[16] Express: expressjs.com

[17] EJS: ejs.co

[18] Bootstrap: getbootstrap.com

[19] Leaflet: leafletjs.com

[20] D3.js: d3js.org

[21] Chart.js: chartjs.org

[22] MySQL: mysql.com

[23] Docker: docker.com

[24] Google Cloud: cloud.google.com

[25] OpenJS Foundation, Jest: jestjs.io