Universidad de Valladolid

Escuela Técnica Superior de Ingenieros de Telecomunicación

Trabajo de Fin de Grado

Grado en Ingeniería de Tecnologías de Telecomunicación

Request To Pay frente a la domiciliación bancaria: propuesta de mejora e implementación de un prototipo

Autor:
Alonso Sandoval Martínez

Tutor:
Federico Simmross Wattenberg

Valladolid, junio 2025

Agradecimientos

Agradezco, en primer lugar, la orientación y el seguimiento de mi tutor, cuya experiencia ha sido decisiva para completar este trabajo.
Extiendo mi gratitud a mi familia por su respaldo constante y a mis amigos por el apoyo práctico y la paciencia mostrada durante el desarrollo del proyecto.
Su ayuda conjunta ha permitido que este trabajo llegue a buen término.

Resumen

Este Trabajo de Fin de Grado explora el potencial del esquema **SEPA**[**Ban23**] ***Request-to-Pay*** **(SRTP)** como alternativa de próxima generación al *SEPA Direct Debit* (SDD). Parte de la premisa de que la domiciliación bancaria actual limita la inmediatez, la trazabilidad y el control del pagador. Para validar la propuesta se ha diseñado y construido un prototipo end-to-end que reproduce el ciclo completo de una petición de pago: creación, presentación, decisión, ejecución y cierre. El prototipo incluye un back-end orientado a eventos, una interfaz web en tiempo real y un banco de pruebas que emula a los participantes del ecosistema SRTP. Sobre esta base se han realizado pruebas funcionales y de rendimiento, escalabilidad y robustez frente a fallos. Los resultados confirman que SRTP puede reducir los tiempos de conciliación y mejorar la visibilidad de las operaciones sin comprometer la seguridad. Finalmente, se discuten las implicaciones de negocio, los requisitos regulatorios y las líneas de trabajo futuro para su industrialización.

Abstract

This Final Degree Project investigates the potential of the **SEPA[Ban23] *Request-to-Pay* (SRTP)** scheme as a next-generation alternative to traditional *SEPA Direct Debit* (SDD). The study starts from the premise that current direct-debit processes limit immediacy, traceability and payer control. To validate the proposal, a full end-to-end prototype has been designed and built that reproduces the entire payment-request life-cycle: creation, presentation, decision, execution and settlement. The prototype comprises an event-driven back-end, a real-time web interface and a test harness that emulates the SRTP ecosystem participants. Functional and performance tests were conducted, measuring scalability and fault tolerance. The results confirm that SRTP can shorten reconciliation times and improve operational visibility without sacrificing security. Business implications, regulatory requirements and future lines of work needed for industrialisation are also discussed.

Contents

Lista de acrónimos

1. API – Application Programming Interface
2. CDN – Content Delivery Network
3. CSS3 – Cascading Style Sheets 3
4. EBA – Euro Banking Association
5. EDS – Electronic Data Submission
6. EDS-Dir – EPC Directory Service
7. EPC – European Payments Council
8. ERPB – Euro Retail Payments Board
9. ES6 – ECMAScript 6
10. HTML5 – HyperText Markup Language 5
11. HTTP – Hypertext Transfer Protocol
12. IBAN – International Bank Account Number
13. ISO20022 – International Organization for Standardization 20022
14. JSON – JavaScript Object Notation
15. NFC – Near Field Communication
16. ORM – Object-Relational Mapping
17. POS – Point of Sale
18. PSD2 – Payment Services Directive 2
19. PSP – Payment Service Provider
20. QR – Quick Response
21. RTP – Request-to-Pay
22. SCA – Strong Customer Authentication
23. SCT – SEPA Credit Transfer
24. SCT Inst – SEPA Instant Credit Transfer
25. SEPA – Single Euro Payments Area
26. SDD – SEPA Direct Debit
27. SRTP – SEPA Request-to-Pay
28. TCP – Transmission Control Protocol
29. TFG – Trabajo de Fin de Grado
30. TIPS – TARGET Instant Payment Settlement
31. TPV – Terminal Punto de Venta
32. WSGI – Web Server Gateway Interface

1 Introducción

Para comprender el entorno actual de los pagos en Europa, conviene arrancar por la *Single Euro Payments Area* (SEPA): un espacio comunitario en el que todos los pagos en euros se rigen por los mismos estándares técnicos y normas operativas, de modo que enviar dinero de un país a otro resulta tan ágil y claro como una transferencia nacional[Ban23]. SEPA estableció protocolos de mensajería comunes, armonizó los plazos de liquidación y fijó reglas uniformes de protección al usuario, creando la base sobre la que se despliegan hoy los servicios de pago más innovadores.

En los últimos diez años, la digitalización de los servicios financieros ha cambiado por completo cómo particulares y empresas gestionan sus transacciones dentro de ese marco SEPA. Las transferencias instantáneas, las API abiertas de los bancos y el auge del comercio electrónico han disparado la demanda de procesos de cobro que sean sencillos, transparentes y en tiempo real. No obstante, los instrumentos de pago tradicionales —tarjetas, transferencias convencionales o domiciliaciones— nacieron en un contexto muy distinto y todavía arrastran limitaciones que penalizan tanto la experiencia de usuario como la eficiencia operativa.

Aquí es donde entra en juego *Request-to-Pay* (RTP). Este servicio de mensajería permite al beneficiario enviar al pagador una solicitud de pago digital estructurada, con todos los detalles (importe, concepto, vencimiento), y recibir en segundos una respuesta —aceptación, rechazo o aplazamiento— antes de iniciar el movimiento de fondos. RTP no sustituye los métodos de pago existentes, sino que actúa como una capa de orquestación sobre la infraestructura SEPA (y, en especial, los pagos inmediatos) y los canales de banca online, facilitando la conciliación, reduciendo la fricción en el cobro y modernizando la experiencia tanto para empresas como para consumidores.

1.1 Motivación

La domiciliación bancaria, regulada por el esquema *SEPA Direct Debit* (**SDD**)[Cou24a] (instrumento paneuropeo de cargo en cuenta regulado por el *European Payments Council* (EPC)) desde 2014, sigue siendo el método principal para cobros recurrentes en España. No obstante, su estructura, pensada para un entorno de procesos *offline*—genera hoy inconvenientes que chocan con las demandas de inmediatez, seguridad y experiencia de usuario fluida que caracterizan la economía digital actual. Así se detectan una serie de ineficiencias, que se describen en la sección 1.1.1, y una oportunidad de introducir un nuevo método de pago que solvente dichas ineficiencias, el cual se presenta en la sección 1.1.2.

1.1.1 Ineficiencias operativas detectadas

Tras analizar la operativa SDD nacional se han identificado una serie de ineficiencias que afectan tanto a los usuarios como a las entidades participantes en el proceso de pago resumidas en los siguientes **5** puntos:

1. **Modelo offline y necesidad de mandato físico**

El proceso de pago mediante el esquema SDD opera bajo un modelo offline, lo que implica una ausencia total de interacción en tiempo real entre las partes involucradas. Para iniciar el cobro, el deudor debe firmar y enviar un **mandato SEPA** en formato físico. Este mandato es un documento mediante el cual el deudor autoriza al acreedor a realizar cobros automáticos a través de la domiciliación bancaria. El documento debe ser conservado por el acreedor durante toda la duración del contrato y hasta 14 meses después de la última transacción realizada. Aunque la digitalización ha avanzado en muchos ámbitos, aún no existe un estándar único y interoperable para los **eMandates**, que son la versión electrónica de los mandatos SEPA que permitem autorizar cobros de manera digital, lo que lleva a que cada entidad bancaria implemente su propio sistema. Esta falta de uniformidad genera inconsistencias y dificulta la estandarización del proceso.

Consecuencias:

- Fricciones significativas en los procesos de venta digital, ya que los usuarios deben completar pasos adicionales que rompen con la inmediatez esperada en el comercio electrónico actual.
- Costes operativos considerables asociados a la gestión administrativa, como el archivado, las auditorías y el mantenimiento de los mandatos físicos.
- Riesgos legales y financieros en caso de disputa, como devoluciones costosas o conflictos prolongados con los deudores, debido a la ausencia de un mandato válido.

2. **Derecho a devolución prolongado**
   El esquema SDD otorga al deudor un derecho a devolución excepcionalmente amplio, lo que genera incertidumbre en la gestión de los ingresos por parte de los acreedores. En el caso de un *cobro autorizado*, el deudor puede solicitar la devolución del importe sin necesidad de justificar su decisión durante un periodo de **ocho semanas**, bajo la política conocida como *"no-questions-asked"*. Por otro lado, si el cobro se clasifica como *no autorizado* —por ejemplo, si el banco emisor no puede probar la existencia de un mandato válido—, el plazo para reclamar se extiende hasta **trece meses**.

   Consecuencias:

   - Notable inseguridad para los acreedores, quienes deben mantener reservas de liquidez y provisiones contables para cubrir posibles devoluciones tardías.
   - Facilitación de prácticas como el *friendly fraud*[1], donde los deudores reclaman reembolsos injustificados tras haber recibido el producto o servicio.
   - Impacto directo en la rentabilidad de las empresas debido a las devoluciones inesperadas.

3. **Ciclos de cobro lentos**
   Los tiempos de procesamiento en el esquema SDD son significativamente prolongados, lo que compromete tanto la eficiencia operativa como la experiencia del usuario. En el esquema Core [Cou25b], el acreedor debe enviar la orden de cobro al banco con una antelación de **D-5 días** para la primera domiciliación y de **D-2 días** para las domiciliaciones recurrentes. A esto se suman **dos días adicionales** para la liquidación

[1] El *friendly fraud* ocurre cuando un usuario consume un bien o servicio y, posteriormente, solicita una devolución sin justificación, aprovechando las políticas de devolución laxas.

interbancaria. En total, el proceso puede demorar entre seis y ocho días naturales desde que se solicita el cobro hasta que se confirma el abono, un plazo incompatible con las expectativas de inmediatez en la venta de bienes o servicios digitales.

Consecuencias:

- Afectación en la planificación financiera de las empresas, ya que los ingresos no están disponibles de manera inmediata, generando una tesorería imprevisible.
- Riesgo de prestar servicios o entregar productos sin la certeza de que el pago se completará con éxito.
- Pérdidas económicas significativas debido a la falta de confirmación inmediata del pago.

4. **Costes y complejidad de las R-transactions**
   Las **R-transactions**, que son transacciones de rechazo, devolución o reembolso asociadas a pagos fallidos o no autorizados, representan una fuente notable de complicaciones y costes adicionales. Estas transacciones se clasifican mediante diversos códigos, cada uno asociado a un flujo y reglas específicas, lo que dificulta su gestión y seguimiento. Los acreedores deben dedicar recursos a identificar las causas de cada R-transaction y aplicar las medidas correctivas correspondientes, un proceso que frecuentemente requiere intervención manual debido a la falta de automatización.

   Consecuencias:

   - Necesidad de equipos especializados en conciliación y recobro, incrementando los costes operativos.
   - Reducción de la eficiencia general del sistema debido a la complejidad de los procesos.
   - Posibilidad de errores o retrasos que afectan la productividad y la confianza en el esquema SDD.

5. **Ausencia de autorización fuerte**
   El esquema SDD se basa en un consentimiento previo otorgado mediante el mandato SEPA, pero no incorpora la **autorización fuerte del cliente (SCA)**, que es un requisito de seguridad que exige la verificación del usuario mediante al menos dos factores de autenticación [Eur18] en el momento de cada transacción. Una vez firmado el mandato, los cobros se ejecutan automáticamente sin que se solicite al deudor una autenticación adicional para cada operación.

   Consecuencias:

   - Elevado riesgo de disputas por cargos no autorizados, lo que puede derivar en conflictos y devoluciones.
   - Pérdida de una oportunidad clave para fortalecer la seguridad y la confianza en el proceso de cobro mediante métodos de autenticación modernos.

En conclusión. Estas ineficiencias tienen un gran impacto en la operativa y la competitividad de las empresas y entidades financieras que lo utilizan y se traducen en:

- Una **estructura de costes elevada**, derivada de la alta frecuencia de devoluciones y la necesidad de personal especializado para gestionarlas, lo que incrementa los gastos operativos.

- **Liquidez incierta**, ya que los ingresos no se confirman de inmediato y pueden ser revertidos incluso meses después de haberse registrado, dificultando la gestión financiera.
- Un **freno al desarrollo de la economía digital**, puesto que el SDD no está diseñado para ofrecer experiencias de pago instantáneas y fluidas, como las que proporcionan métodos alternativos como las tarjetas de crédito, los monederos electrónicos o plataformas como Bizum.

1.1.2 Oportunidad de un esquema RTP

El estándar *SEPA RTP* (**SRTP**) aborda de manera efectiva las limitaciones técnicas y operativas del SDD, ofreciendo una laternativa más ágil y adaptada al entorno digital.

A continuación se describen las principales ventajas del SRTP frente al SDD:

a) **Autenticación reforzada y consentimiento digital inmediato**
El SRTP reemplaza el mandato físico del SDD por una solicitud de pago que el deudor aprueba directamente desde su banca en línea o wallet digital mediante SCA. Este proceso genera una prueba eléctronica de consentimiento, firmada y resgistrada en el sistema del Payment Service Provider (PSP) del pagador, eliminando la dependencia de documentos en papel y simplificando la gestión de autorizaciones.

b) **Irrevocabilidad y mitigación de fraude *post-servicio***
Una vez aceptada la solicitd, el pago se ejecuta mediante SCT Inst, que es una transferencia inmediata SEPA con liquidación en menos de 10 s [Cou24b]. A diferencia del SDD que permite devoluciones automáticas en plazos amplios, el SRTP no admite reversiones sin causa justificada. Esto minimiza el riesgo de *friendly fraud* y reduce la necesidad de provisiones por impagos.

c) **Liquidez *real-time* y conciliación automática**
Con fondos disponibles en menos de 10 segundos, las empresas pueden gestionar su tesorería con mayor precisión. Además, el uso de identificadores únicos y referencias estructuradas según el estándar ISO 20022 [Sta13], asegura que la información del pago se transmita íntegramente de extremo a extremo, permitiendo una conciliación automática y eliminando los retrasos y errores típicos del SDD.

d) **Simplificación operativa**
El SRTP elimina las R-transactions, la custodia de mandatos físicos y las tareas administrativas asociadas. El flujo se reduce a dos mensajes principales -solicitud y aceptación-, con la opción de una transferenciua instantánea, ofreciendo una trazabilidad clara y directa.

e) **Flexibilidad comercial y costes reducidos**
Este esquema soporta cobros únicos, recurrentes o fraccionados a través de canales digitales como enlaces profundos, códigos QR o APIs. Al estar basado en SCT Inst, las comisiones bancarias son bastante menores a las de las tarjetas o la gestión de devoluciones del SDD, lo que mejora la eficiencias y amplía su aplicabilidad en el comercio electrónico.

En conjunto, el SRTP conserva los puntos fuertes del SDD pero los adapta a las necesidades actuales, proporcionando una solución más rápida, segura y eficiente. Al superar las

ineficiencias del SDD se convierte en una herramienta clave para modernizar los sistema de pago en la zona SEPA y, en concreto, en España.

Cuadro 1: Comparativa entre SDD y SRTP con SCT Inst

Aspecto	**SDD**	**SRTP (+ SCT Inst)**
Autorización	Mandato off-line	Consentimiento digital (SCA)
Plazo de devolución	8 semanas / 13 meses	No aplica (irrevocable)
Disponibilidad de fondos	5–8 días	Menos de 10 segundos
Coste operativo	Alto (mandatos, R-CODES)	Bajo (mensajería ISO 20022)
Cobertura *e-commerce*	Limitada	Amplia (API / móvil)
Riesgo de fraude	Medio-Alto (devoluciones)	Bajo (SCA + irreversibilidad)

1.2 Objetivos técnicos fundamentales

El propósito de este Trabajo de Fin de Grado es **demostrar**, de forma general, **cómo el esquema SRTP puede subsanar las ineficiencias operativas del SDD** y adaptarse al contexto digital actual. Para ello **se plantea el desarrollo de un prototipo de referencia que reproduzca el ciclo completo de una petición de pago** y permita evaluar sus ventajas frente al modelo tradicional. De manera deliberada, en esta sección los objetivos se formulan en términos generales; los detalles de implementación (arquitectura, tecnologías y herramientas concretas) se presentan más adelante, en el capítulo de Diseño (3).

A continuación se recogen los objetivos principales, ordenados de lo general a lo particular:

- **Demostrar la viabilidad del SRTP como alternativa al SDD**
  Validar que un flujo basado en solicitudes de pago inmediatas ofrece mayor agilidad, trazabilidad y control que la domiciliación bancaria, sin comprometer la seguridad ni la interoperabilidad dentro del ecosistema SEPA.
- **Emular de extremo a extremo el ciclo operativo de un proveedor RTP**
  Construir un prototipo funcional que reproduzca los pasos esenciales de una operación SRTP —creación, presentación, decisión, ejecución y cierre— implicando a los actores y mensajes definidos por el *rulebook* [Cou25c].
- **Garantizar integridad, confidencialidad y trazabilidad de las transacciones**
  Incorporar salvaguardas de seguridad y gobierno del dato que permitan certificar el origen de cada mensaje, registrar su histórico de estados y conservar evidencias de consentimiento digital.
- **Facilitar la interacción en tiempo real entre los participantes**
  Asegurar que los eventos relevantes del ciclo (solicitud, aceptación, rechazo, expiración, etc.) se propaguen al instante, de modo que pagador y beneficiario dispongan siempre de información actualizada para la toma de decisiones.

- **Proveer una base flexible y extensible para futuras integraciones**
  Diseñar la solución de manera modular, de modo que pueda evolucionar hacia escenarios de producción y aplicaciones reales o incorporar mejoras como autenticación reforzada y analítica de eventos.
- **Evaluar el desempeño y las limitaciones del prototipo mediante pruebas controladas**
  Someter la solución a distintos escenarios (casos favorables, errores operativos, etc) y contrastar los resultados con los objetivos de negocio y los requisitos técnicos del esquema SRTP.
- **Documentar los aprendizajes y proponer líneas de mejora**
  Reflejar de forma crítica las decisiones tomadas, las diferencias respecto a la especificación oficial y las oportunidades para optimizar, escalar o industrializar la solución en trabajos posteriores.

En definitiva, este TFG busca construir una solución práctica que demuestre el potencial del SRTP para superar las trabas del SDD, usando tecnologías modernas y un enfoque riguroso. Este prototipo no solo debe cumplir con los estándares técnicos, sino que también inspire confianza en una nueva forma de gestionar pagos en Europa.

1.3 Fases y Métodos

El TFG se ha estructurado en 3 fases principales:

Fase 1 – Análisis y planificación En esta primera etapa se estudiará el mundo de los pagos en la zona SEPA, revisando los documentos emitidos por el EPC [Cou25a] para identificar las posibles mejoras que el RTP podría suponer.

Luego, se estudiaron los casos de uso del RTP, identificando qué necesitan hacer los actores principales y se planificará el prototipo.

Fase 2 – Diseño e implementación Una vez claro el contexto, se comenzará a diseñar la estructura del prototipo definiendo los roles de los actores y cómo interactúan entre si. La implementación se llevará a cabo usando herramientas que se detallarán en el capítulo de diseño (3) e implementación (4).

Fase 3 – Pruebas y validación Por último, una vez implementado el prototipo se realizarán una serie de pruebas y comprobaciones para verificar que todo funciona correctamente, como veremos en el capítulo de validación (5).

2 Antecedentes y estado del arte

2.1 Evolución de los medios de pago hacia SEPA

El ecosistema europeo de pagos ha experimentado una transformación en las últimas décadas, pasando de sistemas nacionales heterogéneos a un marco unificado bajo la iniciativa **SEPA**. Antes de SEPA, cada país operaba infraestructuras y normas propias para transferencias bancarias y adeudos, lo que complicaba los pagos transfronterizos dentro de Europa. Con la introducción del euro y el objetivo de un mercado único, surgió la necesidad de armonizar los instrumentos de pago. El Consejo Europeo impulsó la creación de SEPA a través del Reglamento UE 260/2012 [Eur12], que fijó la migración obligatoria a los nuevos esquemas paneuropeos de transferencia y adeudo en fechas límite (febrero de 2014 para la zona euro).

Así, en *2008* se lanzó el esquema ***SEPA Credit Transfer (SCT)*** [Cou23c] para transferencias de crédito en euros, y en *2009* el **SDD** para adeudos domiciliados. Estos esquemas sustituyeron progresivamente a los medios nacionales, unificando formatos (por ejemplo, el uso obligatorio de *IBAN* [Com01]) y reglas de funcionamiento en todos los países SEPA. Posteriormente, para atender las demandas de inmediatez, la transferencia instantánea ***SCT Inst*** entró en funcionamiento en *2017*, permitiendo abonar al beneficiario en menos de 10 s. La implantación de *SCT Inst* ha sido voluntaria hasta ahora, aunque recientemente la UE ha aprobado su obligatoriedad progresiva en 2025 para acelerar su adopción. [Cle17].

En la actualidad, los esquemas SEPA (transferencias estándar e inmediatas, adeudos básicos y B2B) concentran la mayoría de pagos bancarios en euros dentro de Europa. Este salto hacia la unificación de pagos fue liderado por la propia industria bancaria europea. En *2002* los bancos constituyeron el ***European Payments Council*** *(EPC)*, órgano de autorregulación que diseña y gestiona los esquemas SEPA. El EPC publicó las primeras *rulebooks* de SCT y SDD en 2008–2009, estableciendo estándares comunes de mensaje *ISO 20022* y calendarios de liquidación. Cabe destacar que el EPC no es un organismo legislativo de la UE ni un regulador, sino una asociación del sector bancario que actúa de facto como ente normalizador: especifica las reglas de los esquemas utilizados por los **PSP** y coopera con los bancos centrales para operar las infraestructuras de compensación. Gracias a esta colaboración público-privada, a partir de *2014* se completó con éxito la migración de millones de pagos nacionales al formato SEPA, eliminando diferencias entre pagos domésticos y transfronterizos en euros.

2.2 El papel del EPC en la estandarización y el surgimiento de Request-to-Pay

El EPC ha desempeñado un rol central en la estandarización de los instrumentos de pago SEPA. Tras la implementación de SCT y SDD, el EPC continuó explorando mejoras para la era digital, en línea con las iniciativas del Eurosistema para fomentar pagos electrónicos

paneuropeos más eficientes. En noviembre de 2018, el *Euro Retail Payments Board* (**ERPB**) —órgano del BCE que orienta la estrategia de pagos minoristas [Ban20] — lanzó un llamado a la acción para desarrollar el concepto de *Request to Pay* como nuevo servicio en la zona SEPA.

Atendiendo esta petición, el EPC creó un grupo de trabajo y comenzó a diseñar un esquema formal de Request to Pay durante 2019–2020. Tras una consulta pública, en noviembre de 2020 se publicó el primer *SRTP Scheme Rulebook* (versión 1.0) y se abrió el registro de participantes. El esquema SRTP entró en vigor el *15 de junio de 2021*, marcando un hito en la evolución de SEPA más allá de los instrumentos tradicionales. Al igual que SCT Inst, la adhesión al esquema RTP es voluntaria; sin embargo, su desarrollo cuenta con fuerte apoyo institucional al considerarse un potenciador de los pagos instantáneos y digitales en Europa. El EPC continúa gestionando y actualizando el esquema (versión 4.0 en 2025), con la expectativa de que Request-to-Pay se integre gradualmente como componente clave del panorama de pagos europeos.

2.3 Funcionamiento técnico de SEPA Request-to-Pay (SRTP)

RTP es un servicio de mensajería financiera que actúa como capa de solicitud previa al pago. A diferencia de los instrumentos tradicionales (transferencias o adeudos) que mueven fondos, RTP no mueve dinero por sí mismo: permite a un beneficiario (*Payee*) enviar electrónicamente una solicitud de pago a un pagador (*Payer*), quien puede aceptarla o rechazarla antes de iniciarse la transacción monetaria. El servicio funciona 24×7 y añade al flujo de pago un intercambio estructurado de datos (importe, concepto, vencimiento, identidad de las partes, etc.) previo al envío de fondos.

Los mensajes SRTP viajan en tiempo real formateados según ISO 20022, lo que facilita su integración con las plataformas SEPA existentes.

Modelo de cuatro esquinas. SRTP adopta la clásica arquitectura *4-corner model* [con25] de la figura 1 presente en el *rulebook* [**ecp014**]

La *Operational Scheme Manager (OSM)* es el elemento central que mantiene vivo y accesible a todo el ecosistema SRTP. Es un gran directorio seguro y siempre actualizado, gestionado por el EPC, donde quedan registrados todos los PSP y demás actores autorizados: sus identificadores, los certificados TLS que usan para cifrar las comunicaciones, las claves de firma y las URLs de sus endpoints(puntos finales de API donde se reciben y se envían RTP). Gracias a la OSM, cada PSP puede, en cualquier momento, localizar de forma sencilla y confiable a otro PSP: comprueba automáticamente que el receptor está homologado, que su endpoint está operativo, y que la conexión será segura antes de intercambiar solicitudes o respuestas de pago. [Cou21].

Su misión va más allá de un simple directorio estático: la OSM supervisa y valida de forma continua la disponibilidad y el correcto funcionamiento de todos los endpoints adheridos, publica esta información en el *EPC Directory Service (EDS)* y notifica de inmediato cualquier incidencia. De este modo, se garantiza que las transacciones RTP fluyan sin interrupciones, con altos estándares de seguridad y cumplimiento de ISO 20022, y que todos los participantes puedan interoperar con total confianza.

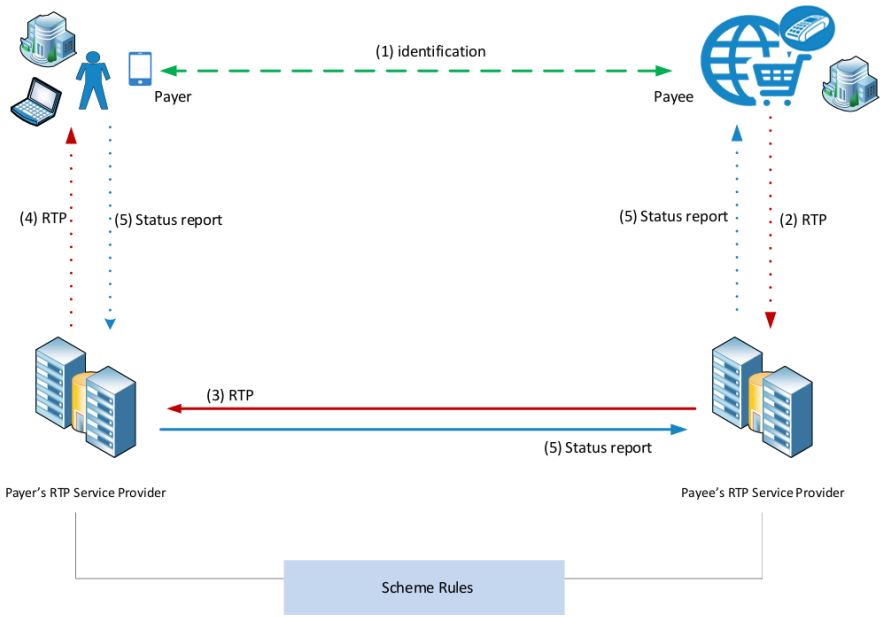

Figura 1: Modelo de 4 esquinas

Cuadro 2: Pasos del flujo del esquema SEPA Request-to-Pay (SRTP) y su descripción

Paso	Descripción
1. Identificación	Una primera interacción que establece la comunicación entre pagador y beneficiario.
2. Envío de la SRTP al PSP del beneficiario	El beneficiario envía la solicitud de pago SRTP a su PSP, incluyendo todos los datos esenciales del esquema (importe, concepto, vencimiento...).
3. Transmisión al PSP del pagador	El PSP del beneficiario reenvía la SRTP al PSP del pagador.
4. Presentación al pagador	La solicitud se muestra al pagador en el canal acordado (app móvil, web, etc.), permitiendo que revise los detalles.
5. Informe de estado	El resultado (aceptación o rechazo) se comunica de vuelta al beneficiario mediante los PSP correspondientes.

En este TFG no se va a simular ni modelar este registro previo en la OSM, asumiremos que todos los actores se encuentran correctamente dados de alta en el directorio. Así, podremos centrarnos exclusivamente en las dinámicas de la solicitud, aceptación, rechazo yu cancelación de pagos, dando por hecho que el acceso a los endpoints y la homologación técnica ya están resueltos.

El flujo básico de un intercambio RTP se encuentra en la figura 2.

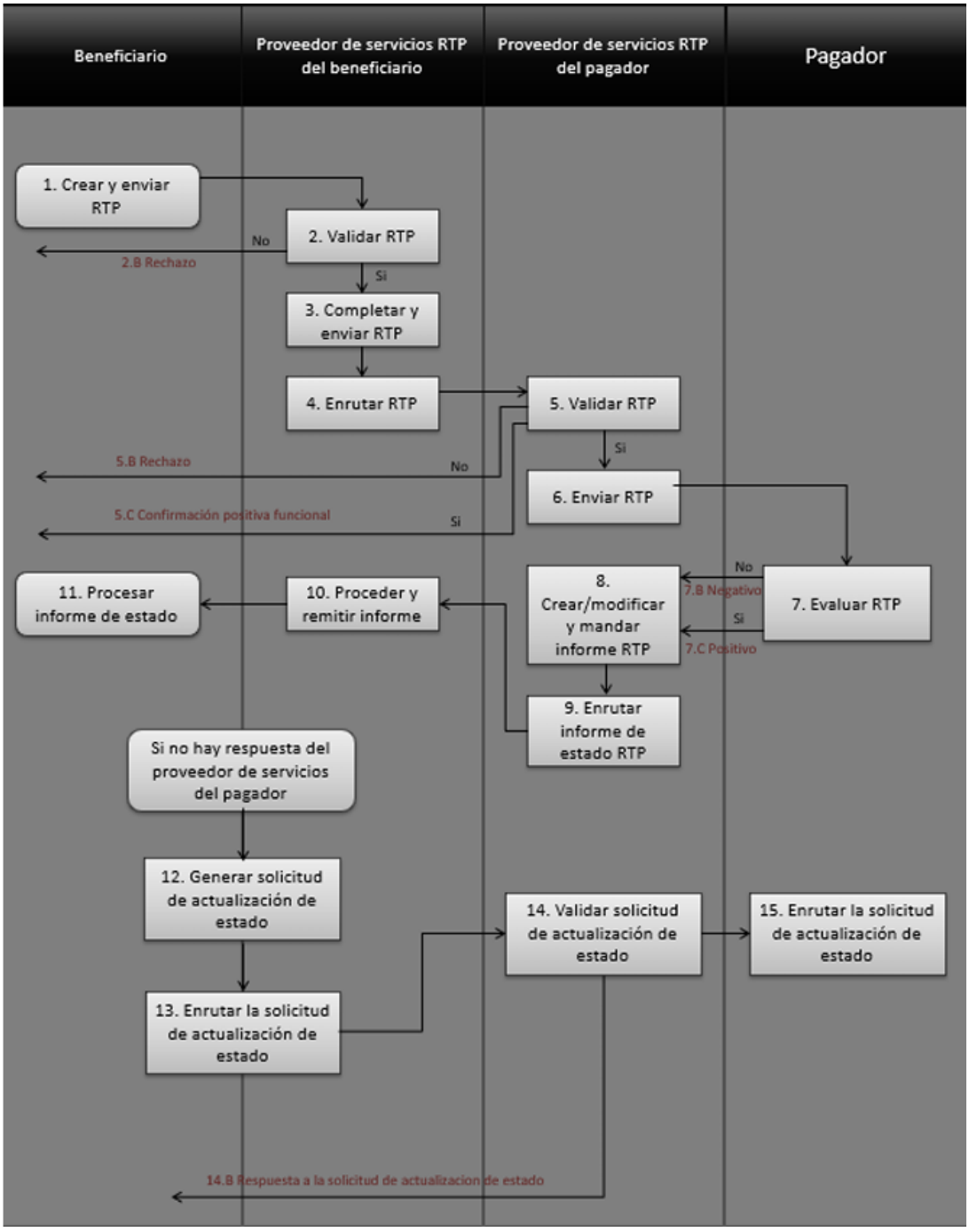

Figura 2: Diagrama de flujo de intercambio RTP

Cuadro 3: Pasos del flujo del esquema SRTP y su descripción

Paso/Función	**Descripción**
1 Crear y enviar RTP	El beneficiario crea el SRTP en el formato normalizado (en un formato acordado bilateralmente con su proveedor). Contiene todos los elementos obligatorios y elementos opcionales que puedan ajustarse al flujo en función de las condiciones comerciales. El beneficiario lo envía al proveedor de servicios SRTP del beneficiario.
2 Validar RTP	El proveedor de servicios SRTP del beneficiario realiza una primera validación del SRTP. Esto incluye, por ejemplo, validación técnica, de seguridad y de formato (por ejemplo, comprobación del IBAN).
2B Rechazo	Si la validación en el paso 2 no tiene éxito, el proveedor de servicios SRTP del beneficiario notifica al beneficiario el rechazo del SRTP, crea un informe de estado negativo y lo envía al beneficiario en el formato acordado con este.
3 Completar y enviar RTP	En caso de validación correcta en el paso 2, el proveedor de servicios SRTP del beneficiario enriquece el SRTP con los elementos necesarios para el enrutamiento en el espacio entre proveedores de servicios SRTP y añade un sello de tiempo.
4 Enrutar RTP	El SRTP se envía al proveedor de servicios SRTP del pagador en función de los mecanismos de enrutamiento establecidos por el PSP.
5 Validar RTP	El proveedor de servicios SRTP del pagador valida el SRTP, incluye la comprobación del identificador del pagador. Esto puede incluir la validación específica del pagador (por ejemplo, si el pagador ha optado por no participar en el servicio, el SRTP es rechazado por defecto).
5B Rechazo	Si la validación en el paso 5 no tiene éxito, el proveedor de servicios SRTP del pagador rechaza el SRTP. El proveedor de servicios SRTP del beneficiario y el beneficiario son informados de este rechazo mediante un código de motivo de no aceptación del RTP.
5C Confirmación positiva funcional	Después de una validación externa en el paso 5, el proveedor de servicios SRTP confirma al pagador que el proveedor de servicios SRTP ha completado con éxito el procedimiento beneficial. Esta confirmación es obligatoria solo en el caso de que el beneficiario o el proveedor de servicios SRTP no haya confirmado previamente la positividad funcional.
6 Enviar RTP	En el caso de validación correcta en el paso 5, el proveedor de servicios SRTP envía el documento SRTP al pagador en el formato acordado (el SRTP puede ser convertido en este paso).
7 Evaluar RTP	El pagador decide si aceptar o rechazar el SRTP, determinando el próximo curso de acción.

(continúa en la siguiente página)

Paso/Función	Descripción
7B Positivo	Si el pagador decide aceptar el SRTP, se envía una respuesta positiva al proveedor de servicios SRTP por parte del pagador.
7C Negativo	Si el pagador rechaza el SRTP, se envía una respuesta negativa al proveedor de servicios SRTP por parte del pagador.
8 Crear/modificar y mandar informe RTP	El proveedor de servicios SRTP crea un informe informativo basado en la decisión del pagador. Si la decisión es negativa (7B/7C), el informe se envía de vuelta al pagador para su revisión. Si el SRTP ya ha sido aceptado o rechazado, surge un caso excepcional donde no se espera un acuse de recibo. El informe debe considerar la fecha de expiración del SRTP. En este caso, el proveedor de servicios SRTP es responsable de notificar al beneficiario la decisión del proveedor de servicios SRTP, incluyendo el código correspondiente. Como resultado, el beneficiario debe representar el SRTP o utilizar otro canal.
9 Enviar informe de estado	El proveedor de servicios SRTP envía un informe de estado actualizado (positivo o negativo) al pagador a través del mismo canal utilizado para el SRTP original, utilizando mecanismos establecidos para la actualización.
10 Proceder y remitir informe	El proveedor de servicios SRTP del beneficiario procesa el informe de estado recibido (positivo o negativo), informa al beneficiario y decide los pasos siguientes previo acuerdo con el beneficiario.
11 Procesar informe de estado	El beneficiario ejecuta las acciones finales tras la recepción del informe de estado: actualización del estado final del registro SRTP, preparación del pago SRTP conciliación, etc.
12 Generar solicitud de actualización de estado	El beneficiario y el proveedor de servicios SRTP del beneficiario pueden enviar una Solicitud de Actualización de Estado si no se ha recibido respuesta hasta la Fecha/Hora de Expiración.
13 Enrutar la solicitud de actualización de estado	La solicitud de actualización de estado al proveedor de servicios SRTP del pagador se enruta a través de la misma vía utilizada para el SRTP original basándose en los mecanismos de enrutamiento establecidos.
14 Validar solicitud de actualización de estado	Tras la recepción de la solicitud de actualización de estado, el Proveedor de Servicios SRTP del pagador comprueba la validez de la Solicitud.
14B Respuesta a la solicitud de actualización de estado	El Proveedor de Servicios SRTP del pagador responde al proveedor de servicios SRTP del beneficiario y, si procede (a través del proveedor de servicios SRTP del beneficiario), al beneficiario (por ejemplo, respuesta del SRTP original no recibida, el pagador aún no ha respondido, etc.).

(continúa en la siguiente página)

Paso/Función	Descripción
15 Enrutar la solicitud de actualización de estado	En caso de que el pagador aún no haya respondido al SRTP inicial, el proveedor de servicios SRTP del pagador puede enviar la solicitud de actualización de estado al Pagador.
Fin de la tabla	

Hay algunos detalles acerca del esquema que conviene aclarar:

1. **Rechazo de un SRTP o una Solicitud de Cancelación (Reject)**
   Un *reject* se produce cuando un SRTP o una Solicitud de Cancelación no es aceptada antes de ser enviada al siguiente participante en la cadena de pago. El mensaje de rechazo sigue la misma ruta que el SRTP original sin modificar ningún dato, y se incluye un registro con los detalles necesarios para asegurar un rastro de auditoría. Además, el mensaje de rechazo lleva un código de motivo que explica la razón del rechazo. La identificación del SRTP original se hace mediante la referencia única incluida por el proveedor de servicios del receptor. Los rechazos se envían de manera instantánea por el proveedor de servicios SRTP que no puede procesar la solicitud, y estos rechazos pueden ser generados automáticamente en función de comprobaciones técnicas o comerciales, sin intervención del pagador.
2. **Respuestas a un SRTP (positiva o negativa)**
   Cuando un pagador responde a un SRTP, puede aceptar (respuesta positiva) o rechazarlo (respuesta negativa). En ambos casos, el mensaje sigue la misma ruta que el SRTP original, sin alterar los datos, y se envía instantáneamente entre los proveedores de servicios SRTP. Las respuestas negativas incluyen un código de motivo que especifica la razón del rechazo, mientras que las respuestas positivas simplemente confirman la aceptación de la solicitud. Como en el caso de los rechazos, se mantiene un registro detallado de los datos relevantes para asegurar la trazabilidad del proceso y la transparencia en la comunicación entre los proveedores de servicios.
3. **Solicitud de Cancelación del SRTP**
   Una "Request for cancelation" RfC puede ser iniciada por el y se transmite al pagador a través de los proveedores de servicios SRTP. La solicitud sigue la misma ruta que el SRTP original, sin modificar los datos, y debe incluir un código de motivo que justifique la cancelación. Esta solicitud puede realizarse hasta la fecha de expiración del SRTP, a menos que ya haya sido rechazado, cancelado o expirado. El proveedor de servicios SRTP del pagador verifica la validez de la solicitud antes de reenviarla, y si no se puede procesar, envía una respuesta negativa. Si la cancelación se ejecuta correctamente, el proveedor del pagador envía una respuesta positiva de manera instantánea, manteniendo siempre un registro para garantizar la trazabilidad del proceso.

Casos de uso representativos. El **SRTP** se concibió como un servicio versátil, capaz de cubrir desde pagos cotidianos de bajo valor hasta cobros empresariales complejos. No

obstante, el caso de uso considerado más transformador y sobre el que se centra este TFG es la sustitución del SDD. Aun así, existen otros escenarios relevantes que merece la pena describir para contextualizar el alcance potencial de SRTP.

1. **Punto de Venta Físico**
   - **Descripción:** este caso de uso de RTP se emplea en tiendas físicas, donde el beneficiario (el comercio) envía una solicitud de pago al pagador (el cliente) utilizando un código QR o una tecnología *Near Field Communication(NFC).* Al escanear el código QR con su móvil o usar NFC en la terminal de pago, el cliente es redirigido a su aplicación bancaria para autorizar el pago de manera inmediata.
   - **Proceso de identificación:**
     - *Identificación del pagador:* el pagador se autentica directamente en su aplicación bancaria, generalmente mediante su número de cuenta bancaria, número de tarjeta, o métodos de autenticación biométrica o PIN, según lo permita la aplicación del banco.
     - *Identificación del beneficiario:* el beneficiario está identificado en el sistema mediante un ID de comercio asociado a la terminal de pago o al código QR/NFC escaneado por el cliente. Estos elementos proporcionan los datos necesarios para vincular la solicitud de pago al comercio correspondiente.
   - **Proceso de pago:**
     - El cliente escanea el código QR o se conecta mediante NFC a la terminal de pago.
     - La aplicación bancaria del cliente recibe la solicitud de pago con el monto y la referencia.
     - El cliente revisa la información y autoriza el pago, un proceso rápido y conveniente, crucial en entornos físicos donde la velocidad es esencial.
   - **Diferencias frente a otros casos de uso:**
     - La rapidez y conveniencia son fundamentales en este caso de uso. El proceso de identificación y autorización es sencillo y rápido, requiriendo solo la confirmación del cliente a través de su banco, típicamente con autenticación biométrica o PIN.
     - Es ideal para transacciones de bajo valor donde la experiencia del cliente es un factor determinante.
2. **Comercio electrónico (*E-commerce*)**
   - **Descripción:** en este caso, el beneficiario (el comercio electrónico) envía una solicitud de pago al pagador (el cliente) durante el proceso de *checkout* o mediante un enlace de pago enviado a través de una aplicación bancaria o correo electrónico. El cliente es redirigido a su aplicación bancaria, donde debe autenticarse y revisar los detalles antes de aprobar el pago.
   - **Proceso de identificación:**
     - *Identificación del pagador:* la identificación implica un proceso de autenticación multifactor (por ejemplo, contraseña, *token* de seguridad o autenticación biométrica) para garantizar la seguridad de la transacción, realizado en la aplicación bancaria o la página de pago del comercio.

  - *Identificación del beneficiario:* el beneficiario se identifica mediante un ID de comercio electrónico que incluye su nombre comercial, identificador fiscal, número de cuenta o un identificador único en la plataforma de pagos.
- **Proceso de pago:**
  - El cliente recibe la solicitud de pago a través de un enlace en el checkout o en su aplicación bancaria.
  - El cliente se autentica en su aplicación bancaria, revisa el monto, la referencia y los detalles del comerciante, y aprueba el pago.
  - Una vez validada, el dinero se transfiere de manera segura.
- **Diferencias frente a otros casos de uso:**
  - A diferencia del punto de venta físico, el proceso en comercio electrónico es más lento debido a la autenticación adicional y la revisión en línea.
  - La seguridad es prioritaria, dado el mayor riesgo de fraude en transacciones digitales.

3. **Facturación electrónica (*E-invoicing*)**
   - **Descripción:** común en empresas que envían facturas electrónicas, el beneficiario (la empresa emisora) envía una solicitud de pago con los detalles de la factura al pagador (el cliente). El cliente recibe la solicitud en su correo electrónico o aplicación bancaria, pudiendo revisar los detalles antes de decidir cuándo pagar.
   - **Proceso de identificación:**
     - *Identificación del pagador:* el pagador se identifica mediante su número de cliente, correo electrónico o número de cuenta bancaria vinculado a la empresa emisora.
     - *Identificación del beneficiario:* el beneficiario se identifica por su NIF (Número de Identificación Fiscal) o un ID de facturación electrónica único, asegurando la verificación de la entidad receptora.
   - **Proceso de pago:**
     - El cliente recibe la factura y la solicitud de pago por correo o notificación bancaria.
     - Tras validar la información, autoriza el pago a través de su aplicación bancaria, completando la transacción.
   - **Diferencias frente a otros casos de uso:**
     - Ofrece flexibilidad, permitiendo al cliente revisar la factura antes de pagar.
     - La identificación del payee incluye detalles fiscales, incrementando la seguridad en la validación.
4. **Pagos recurrentes**
   - **Descripción:** ideal para suscripciones o pagos periódicos (streaming, software en la nube, gimnasios), el payor autoriza la primera transacción para suscribirse. Los pagos posteriores se gestionan automáticamente según la periodicidad acordada, sin intervención adicional.
   - **Proceso de identificación:**
     - *Identificación del pagador:* el pagador autoriza la primera transacción con autenticación en su aplicación bancaria (cuenta, contraseña o biométrica).

- *Identificación del beneficiario:* el beneficiario se identifica por su ID de suscripción y número de cuenta para configurar los pagos recurrentes.
- **Proceso de pago:**
  - El cliente autoriza el pago inicial.
  - Los cobros posteriores se realizan automáticamente según la periodicidad (mensual, anual, etc.).
- **Diferencias frente a otros casos de uso:**
  - Se distingue por la automatización tras la autorización inicial, reduciendo la intervención del cliente.
  - La comodidad y la automatización son sus principales ventajas.

5. **Pagos de grandes montos**
   - **Descripción:** utilizado en transacciones de alto valor (compras importantes, bienes de gran valor), este caso requiere métodos adicionales de autenticación para garantizar la seguridad.
   - **Proceso de identificación:**
     - *Identificación del pagador:* el pagador usa autenticación robusta (OTP, *tokens* de seguridad o multifactor como PIN y biométrica) para validar la transacción.
     - *Identificación del beneficiario:* el beneficiario es una entidad registrada con un identificador único (ID de comerciante), asegurando el destinatario correcto.
   - **Proceso de pago:**
     - El pagador revisa el monto y autoriza el pago con autenticación adicional.
     - Tras verificar la identidad, el pago se completa.
   - **Diferencias frente a otros casos de uso:**
     - Requiere autenticación más fuerte debido al alto riesgo de fraude en grandes sumas.
     - La seguridad es el factor más crítico, priorizando la protección contra fraudes y errores.

2.4 RTP dentro del ecosistema de pagos: arquitectura en capas

El ecosistema de pagos SEPA se puede describir mediante capas jerárquicas, desde los usuarios finales hasta las infraestructuras financieras que ejecutan las transacciones. A continuación se detallan las principales capas del modelo actual de pagos SEPA presente en la figura 3:

Capa de usuarios finales. En la capa más alta se encuentran el *pagador* y el *beneficiario*. Son quienes inician y reciben los pagos, respectivamente, ya sea personas o empresas involucradas en una transacción.

Capa de iniciación o interacción. Es donde se produce la solicitud o autorización del pago a través de algún canal o interfaz. Por ejemplo, en pagos SEPA tradicionales el pagador suele autorizar una transferencia mediante la banca online o una aplicación móvil. En entornos de comercio electrónico o físico, esta capa correspondería al proceso de *checkout* (por ejemplo, introduciendo datos en una pasarela de pago online o mediante un TPV/POS en tienda).

Capa de proveedores de servicios de pago (PSP). Aquí actúan las entidades financieras o PSP de cada parte. Típicamente, el banco del pagador y el banco del beneficiario son quienes ofrecen las cuentas bancarias y servicios de pago a sus clientes. Estas entidades facilitan la emisión y recepción de órdenes de pago en nombre de los usuarios finales.

Capa de esquemas de pago. Es el conjunto de reglas y estándares comunes que permiten la interoperabilidad entre todos los PSP. En SEPA, por ejemplo, existen los esquemas de transferencia crediticia SCT o SCT Inst y adeudo directo SDD, definidos por el EPC (explicados anteriormente). Cada esquema asegura que, independientemente del banco involucrado, un pago en euros siga las mismas normas y formatos. Un esquema de pago no es el movimiento del dinero en sí, sino el conjunto de reglas, mensajes e infraestructura técnica que orquesta cómo se inician y gestionan esos pagos.

Capa de infraestructuras de compensación. Son las plataformas interbancarias que procesan las transacciones según las reglas del esquema. Estas infraestructuras se encargan de intercambiar las órdenes de pago entre el banco emisor y el banco receptor, aplicando compensación multilateral si procede.

Capa de liquidación. Es la capa más baja, donde se realiza la liquidación final de fondos entre bancos. Normalmente ocurre en los bancos centrales u organismos de compensación: por ejemplo, las obligaciones netas resultantes de muchas transacciones SEPA pueden liquidarse en TARGET2 (sistema del Banco Central Europeo). En pagos inmediatos, la liquidación suele ser casi en tiempo real (por ejemplo, mediante TIPS [Ban18] que liquida cada operación al momento en cuentas del banco central).

Figura 3: Esquema por capas

Esta arquitectura por capas garantiza que un pago iniciado por un usuario en un banco pueda llegar a otro usuario en distinto banco de forma segura, eficiente e interoperable en toda la zona euro. Cada capa agrega funciones específicas: los usuarios generan órdenes, los PSP las gestionan, los esquemas proporcionan las reglas comunes, y las infraestructuras las ejecutan y asientan los fondos en última instancia.

RTP es un nuevo servicio incorporado en la arquitectura SEPA que se sitúa principalmente en la capa de esquema de pago, actuando como una capa adicional de mensajería sobre los instrumentos de pago existentes.

Ejemplo comparado: pago con tarjeta vs. RTP

Como muestra el esquema comparativo de la figura 4 los pagos con tarjeta (ej. Visa/Mastercard) históricamente han operado con una arquitectura de funciones similar a la de SEPA, pero con diferencias en los actores y procesos de cada capa:

Usuarios (pagador/beneficiario). En tarjetas, el pagador es el *titular de la tarjeta* y el beneficiario es el *comercio* que recibe el pago. En esencia es equivalente al ordenante y beneficiario de una transferencia, con la diferencia de que el pagador utiliza un instrumento distinto (su tarjeta en lugar de una cuenta bancaria directa).

PSP / entidades. En el modelo de cuatro partes de las tarjetas interviene el *banco emisor* (emite la tarjeta al pagador) y el *banco adquirente* (procesa pagos para el comerciante). Estos roles son análogos a la entidad del pagador y del beneficiario en SEPA, pero en el mundo tarjeta suelen implicar acuerdos específicos (p. ej. el comerciante contrata un adquirente para aceptar Visa/Mastercard). En cambio, en SEPA cualquier banco puede enviar o recibir transferencias para un cliente sin acuerdos individuales con cada comercio, ya que todos siguen el esquema común.

Esquema de pago. Las tarjetas operan bajo esquemas propietarios como Visa, Mastercard, etc., que definen reglas, formatos de mensajes (p. ej. mensajes de autorización y liquidación) y que actúan también como redes de procesamiento. Son equivalentes a los esquemas SEPA en cuanto a que proveen interoperabilidad, pero controlados por empresas particulares. El esquema Visa/Mastercard indica cómo se autoriza una compra, cómo se liquida posteriormente y fija también aspectos comerciales como las tasas de intercambio entre emisor y adquirente. En SEPA, el esquema SRTP + SCT Inst provee una funcionalidad comparable de solicitud y pago, pero dentro de un marco colaborativo paneuropeo.

Infraestructura de compensación. En los pagos con tarjeta, la red del esquema (p. ej. VisaNet) se encarga de la autorización instantánea de la transacción y de la compensación/*clearing* de las transacciones entre emisores y adquirentes. Visa o Mastercard centralizan el intercambio de mensajes financieros. En SEPA, por el contrario, las compensaciones suelen realizarse a través de múltiples infraestructuras (por ejemplo, cámaras como STEP2 o servicios inmediatos como TIPS) que no pertenecen a una sola empresa sino que son parte del ecosistema colaborativo europeo. Con RTP, la mensajería de solicitud viaja por la red designada (p. ej. la plataforma R2P de EBA Clearing) y el pago resultante se compensa a través de los *rails* SEPA existentes (p. ej. RT1/TIPS para instantáneas).

Liquidación final. En ambos casos, finalmente hay un traspaso de fondos entre bancos. En las tarjetas, las marcas de tarjeta calculan las obligaciones netas entre cada banco emisor y adquirente y típicamente las liquidan al final del día a través de cuentas en un banco central u otros mecanismos interbancarios. En SEPA, cada transferencia individual (especialmente si es instantánea) puede liquidarse inmediatamente en el banco central. Desde el punto de vista de capas, ambos mundos terminan convergiendo en la necesidad de que el dinero se ajuste entre las cuentas de los bancos participantes.

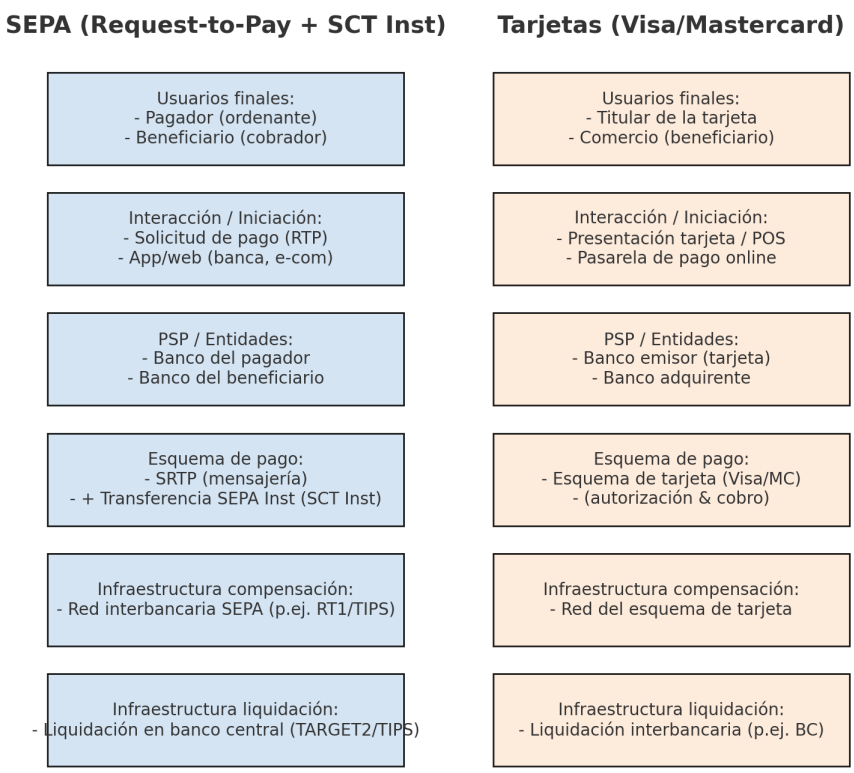

Figura 4: LayerComp

¿Qué simplifica o elimina RTP respecto al modelo de tarjetas?

Principalmente, elimina intermediarios dedicados y procesos redundantes. Por ejemplo, en un pago SRTP + SCT Inst no es necesario un procesador específico ni una red de tarjetas propietaria, ya que los propios bancos de pagador y beneficiario se comunican directamente mediante el esquema común. Esto puede reducir costes de aceptación para el comercio (evitando comisiones elevadas de tarjetas) y simplifica la integración: el comercio recibe el dinero directamente en su cuenta bancaria vía SEPA, sin pasos intermedios de recibir fondos a través de entidades de tarjeta y luego liquidarlos. Además, no se requiere que el pagador proporcione datos sensibles como el PAN de tarjeta o incluso su IBAN al comercio; la solicitud llega por canales bancarios seguros y el cliente simplemente autoriza en su entorno bancario. En resumen, RTP se apoya en la infraestructura bancaria existente (cuentas y pagos inmediatos) para ofrecer una experiencia similar a la de tarjeta, pero con

menos capas propietarias, aprovechando la red SEPA ya desplegada en toda Europa.

3 Diseño

En este apartado se detalla la estrategia adoptada para *emular, con un prototipo funcional, el comportamiento extremo-a-extremo del servicio SEPA RTP*. Se parte de los requisitos funcionales publicados en el *EPC RTP Rulebook 4.0* y se traducen a casos de uso concretos, modelo de datos y flujos de interacción que servirán de base a la implementación descrita en el apartado 4. Se persigue, ante todo, una visión que facilite al lector identificar primero el *qué* resuelve la solución y, a continuación, el *cómo* se materializa.

3.1 Objetivos técnicos derivados

El diseño e implementación del prototipo desarrollado responden a los objetivos técnicos fundamentales definidos en el capítulo 1.2. Estos objetivos buscan demostrar que el esquema SEPA RTP puede superar las limitaciones del sistema tradicional de domiciliación bancaria, ofreciendo una solución moderna, eficiente y adaptada a las demandas de la economía digital actual. A continuación se detallan estos objetivos en orden de prioridad, explicando qué se pretende conseguir con cada uno, cómo se implementaría y por qué son esenciales para el éxito del prototipo.

1 **Reproducir el ciclo RTP extremo a extremo**

   El propósito fundamental del prototipo es desarrollar un sistema que emule de manera completa y fiel el ciclo del esquema RTP, abarcando todas las etapas definidas en la tabla de pasos del SRTP 3. Esto implica que el prototipo debe ser capaz de replicar el flujo completo del proceso, desde el momento en que el beneficiario inicia una solicitud de pago hasta la resolución final por parte del pagador, incluyendo tanto escenarios exitosos como aquellos en los que se producen errores o rechazos.

   Quiero que el sistema emule todas las operaciones clave del RTP: la creación de la solicitud por parte del beneficiario, la validación por parte de los PSP y la decisión final del pagador. Esto incluye manejar situaciones reales, como la falta de fondos del pagador o errores en los datos de la solicitud, para garantizar que el prototipo sea robusto y represente fielmente cómo funcionaría un proveedor de RTP en un entorno real.

   El sistema implementará una API basada en HTTP/JSON que cubra las operaciones principales de RTP. Además se diseñarán mecanismos para simular errores como IBAN inválidos o saldos insuficientes, y se registrarán los resultados de cada etapa para analizar su comportamiento.

   Reproducir el ciclo completo es esencial para demostrar que el RTP puede ser una alternativa viable al SDD, resolviendo los problemas como la lentitud o la falta de interacción en tiempo real y proporcionando una base sólida para futuras implementaciones reales.

2 **Ofrecer notificación en tiempo real**

   Dado que el RTP es un esquema diseñado para operar en un entorno digital y online, se busca garantizar que todas las partes involucradas reciban información inmediata sobre cualquier cambio o evento relacionado con la solicitud de pago.

Quiero que el sistema notifique instantáneamente a los actores cada vez que ocurra un evento significativo, como la creación de una solicitud RTP, su validación por un PSP o la decisión final del pagador. Esto simulará la experiencia que un usuario tendría al recibir una notificación en su aplicación bancaria, permitiendo al pagador reaccionar rápidamente y al beneficiario conocer el estado de su solicitud sin demoras.

Se utilizarán eventos de WebSocket, una tecnología que permite una comunicación bidireccional en tiempo real entre el servidor y los clientes. Por ejemplo, cuando el beneficiario crea una solicitud, el sistema enviará una notificación al pagador a través de una sala específica WebSocket; de manera similar, cuando el pagador tome una decisión, el beneficiario será informado de inmediato. Este enfoque asegura que las actualizaciones sean push en lugar de depender de consultas manuales.

La inmediatez es una de las principales ventajas de RTP frente a SDD, que opera en un modelo offline con retrasos de días. Este objetivo refleja la necesidad de una experiencia de usuario fluida y ágil, alineada con las expectativas actuales de rapidez en el comercio digital.

3 Garantizar seguridad y trazabilidad

La seguridad y la capacidad de rastrear todas las acciones realizadas son pilares fundamentales para cualquier sistema de pagos, especialmente uno que aspire a ser adoptado en un contexto real.

Quiero que cada cambio de estado en una solicitud RTP quede registrado de forma segura e inalterable en la base de datos, permitiendo reconstruir el historial completo de cualquier solicitud en cualquier momento. Esto servirá para auditar el sistema, prevenir fraudes y resolver disputas entre las partes.

Cada transición de estado se almacenará junto con un hash SHA-256, que asegura la integridad de los datos, y una marca de tiempo en formato UTC, que indica el momento exacto de la acción. Por ejemplo, si un PSP valida una solicitud, se generará un registro con estos elementos, y cualquier intento de modificar los datos será detectable gracias al hash. Además se implementarán controles de acceso por roles para limitar quién puede realizar cada acción.

Sin seguridad y trazabilidad, el sistema será vulnerable a manipulaciones o ataques, lo que comprometería su credibilidad. Este objetivo asegura que el prototipo cumpla con los requisitos de confianza y auditoría, esenciales para su evolución hacia un sistema de producción.

4 Persistir la información de forma ligera

Es importante diseñar un sistema de almacenamiento que sea práctico para las fases iniciales del desarrollo y pruebas, pero que también permita crecer en el futuro si el prototipo se expande.

Quiero que la información generada por las solicitudes RTP se guarde de forma sencilla y sin requerir recursos excesivos durante el desarrollo, utilizando herramientas que no dependan de configuraciones complejas o servidores externos. Al mismo tiempo, quiero que el diseño sea flexible para adaptarse a necesidades mayores más adelante.

Se empleará SQLite como base de datos, gestionada mediante SQLAlchemy, una biblioteca que facilita la interacción con los datos. SQLite es ideal para este prototipo porque es ligera, no requiere instalación adicional y funciona bien en entornos locales.

Sin embargo, el sistema se estructurará de manera modular, permitiendo una migración futura a PostgreSQL u otra base de datos más robusta si el volumen de transacciones o usuarios aumenta.

Un almacenamiento eficiente reduce la complejidad del desarrollo inicial y asegura que el prototipo sea fácil de instalar y probar. La flexibilidad para escalar es clave para que el sistema no quede obsoleto si se decide llevarlo a un entorno real.

5 **Facilitar pruebas automatizadas e integración continua**

Para garantizar la calidad y fiabilidad del prototipo, este objetivo busca establecer un proceso de verificación continua que detecte errores y asegure que el sistema funciona correctamente a medida que evoluciona.

Quiero crear un conjunto de pruebas que revisen automáticamente las funcionalidades del servidor y que estas pruebas se ejecuten de manera recurrente cada vez que se realicen cambios en el código. Esto asegurará que el sistema permanezca estable y funcional durante todo el desarrollo.

Se diseñarán pruebas automáticas utilizando Postman, una herramienta que permite simular peticiones HTTP y validar las respuestas de la API. Estas pruebas cubrirán los endpoints principales y se integrarán en un flujo de integración continua.

Las pruebas automatizadas son esenciales para mantener la calidad del software, especialmente en un prototipo que simula un sistema crítico como el RTP. Este objetivo reduce el riesgo de errores no detectados y facilita la incorporación de nuevas funcionalidades sin comprometer la estabilidad.

3.2 Actores y alcance

La emulacion del SRTP en este prototipo se basa en el modelo de cuatro esquinas expuesto anteriormente en la figura 1. Este modelo es ampliamente utilizado en sistemas de pagos europeos porque proporciona un marco claro y seguro para gestionar transacciones, dividiendo las responsabilidades entre cuatro actores principales. Estos actores trabajan juntos para garantizar que una solicitud de pago fluya desde quien la inicia hasya quien debe responderla, pasando por intermediarios financieros que facilitan el proceso. A continuación se describen en detalle los actores y cómo interactúan, seguidos por una explicación del alcance de la emulación en el prototipo.

Actores del modelo de cuatro esquinas

Los cuatro actores que conforman el modelo son los siguientes:

Beneficiario. Es el comercio, empresa o persona que indica la solicitud de pago (RTP). Este actor es quien tiene la necesidad de recibir un pago, como un comercio enviando una factura a un cliente, una empresa de servicios solicitando el pago de una cuenta o incluso un individuo pidiendo dinero a otr. En el prototipo, el beneficiario es simulado para dar comienzo al proceso, enviando la solicitud de pago a su proveedor de servicios para que sea procesada.

PSP Beneficiario. Este es el PSP que da soporte al beneficiario. Un PSP es tipicamente un banco, una entidad financiera o una plataforma de pagos que actúa en nombre del beneficiario. Su rol principal consiste en recibir la solicitud de pago iniciada por el

beneficiario, verificar que sea válida y luego enrutarla hacia el PSP del pagador. En el prototipo, este actor simula esta función de validación y enrutamiento, asegurando que la solicitud avance al siguiente paso.

PSP Pagador. Es el PSP que atiende al pagador final. Recibe la solicitud de pago enviada por el PSP beneficiario, la valida y la comunica al pagador para que tome una decisión. En un sistema real, si el pagador acepta la solicitud, el PSP pagador también ejecutaría la transferencia de fondos, pero en este prorotipo, como se detalla más adelante, esta parte no está incluida. Aquí, el PSp pagador simula la entrega de la solicitud y la recepción de la respuesta del pagador.

Pagador. Es el cliente bancario o usuario final que recibe la solicitud de pago y decide si la acepta o la rechaza. En la vida real, esto podría ser una persona que recibe una notificación en su banca en línea o aplicación móvil, donde se le pide autorizar un pago. En el prototipo, el pagador es simulado para cerrar el ciclo de la solicitud, tomando la decisión final de aceptación o rechazo.

Estas interacciones entre los actores de ilustran en la figura 1.

Alcance de la emulación. El prototipo tiene como objetivo simular el flujo del esquema SRTP en un entorno controlado, pero con un alcance limitado para enfocarse en los aspectos esenciales del proceso de solicitud de pago. Esto significa que no replica todas las funcionalidades de un sistema RTP real, sino que prioriza ciertas partes para cumplir con los objetivos del proyecto. A continuación, se explica qué incluye la emulación, qué se deja fuera y las razones detrás de estas decisiones.

Registro en la OSM. En un sistema RTP real, los actores deben estar registrados y autenticados a través de la OSM, una entidad central que asegura que todas las partes sean legítimas y cumplan con las reglas del esquema. Sin embargo, en este prototipo, no se aplica el registro en la OSM. En lugar de implementar este proceso, se asume que los actores ya están registrados y autenticados de antemano. Esta simplificación elimina la necesidad de desarrollar una lógica compleja de registro y autenticación, lo que reduce el esfuerzo de desarrollo y permite centrarse en el flujo de la solicitud de pago. Sin embargo, esto implica que el prototipo no abroda aspectos de seguridad relacionados con la verificación de identidad de los actores, una limitación aceptable para los fines de esta simulación.

Materialización de la orden de pago SCT Inst. En un sistema RTP completo, cuando el pagador acepta la solicitud del pago, se genera una orden de pago mediante SEPA Inst, un esquema que permite transferenciuas de fondos instantáneas entre cuentas bancarias en la zona SEPA. En este prototipo, no se materializa la órden de pago SCT Inst después de que la solicitud es aceptada. Esto significa que el sistema simula todo el proceso hasta la aceptación o rechazo de la solicitud, pero no ejecuta la transferencia de dinero real. Esta decisión se basa en las limitaciones discutidas en el SRTP Scheme Rulebook [Cou25c], y se justifica porque el objetivo principal del prototipo es demostrar la interacción entre los actores y la lógica de la solicitud de pago, no integrar un sistema de pagos real. Incluir la ejecución de SCT Inst requeriría conectar el prototipo a servicios bancarios o simular un sistema financiero completo, lo que añadiría una complejidad innecesaria a este proyecto.

En resumen, la emulación sigue el modelo de cuatro esquinas de la EPC, con cuatro actores claramente definidos que interactúan en una red local. El prototipo se centra en simular la solicitud de pago y las respuestas entre los actores, omitiendo deliberadamente el registro en la OSM y la materialización de la orden de pago SCT Inst para mantener el enfoque en los aspectos clave del RTP, alineándose con los objetivos del proyecto y las limitaciones establecidas.

3.3 Requisitos

3.3.1 Requisitos funcionales (RF)

Los requisitos funcionales detallan las operaciones específicas que el sistema debe realizar para emular el esquema SRTP. Estos requisitos aseguran que el prototipo implementado cubra el flujo completo de una solicitud de pago, desde su creación hasta la decisión final, incluyendo las interacciones entre los actores del modelo de cuatro esquinas.

Id.	**Descripción**
RF-01	El Beneficiario debe poder crear una solicitud RTP indicando el IBAN del pagador, el importe en una moneda compatible con ISO 4217 (por ejemplo, euros) y un concepto descriptivo en formato UTF-8. La solicitud debe ser enviada al PSP Beneficiario para su procesamiento.
RF-02	El PSP Beneficiario debe validar la sintaxis y el formato de la solicitud RTP recibida, asegurando que cumpla con los estándares del esquema (por ejemplo, IBAN válido, importe correcto). Una vez validada, debe enrutar la solicitud al PSP pagador correspondiente.
RF-03	El PSP pagador debe aplicar reglas básicas de KYC (*Know Your Customer*) y AML (*Anti-Money Laundering*), como verificar que el pagador esté registrado y tenga capacidad para responder a la solicitud. Tras la validación, debe presentar la solicitud al pagador para que tome una decisión.
RF-04	El pagador debe poder aprobar o rechazar la solicitud RTP con un tiempo máximo, simulando la inmediatez del esquema RTP. La decisión debe ser comunicada de vuelta al PSP pagador.
RF-05	El Beneficiario debe recibir una notificación inmediata sobre la decisión del pagador (aceptación o rechazo) a través del sistema de notificaciones en tiempo real.
RF-06	Cualquier parte involucrada debe poder emitir una solicitud de cancelación (*cancellation request*) antes de que se tome la decisión final sobre la solicitud RTP.

Cuadro 4: Requisitos funcionales

3.3.2 Requisitos no funcionales (RNF)

Los requisitos no funcionales especifican las cualidades y restricciones del sistema que garantizan su rendimiento, seguridad y mantenibilidad. Estos requisitos aseguran que el prototipo sea eficiente y adecuado para simular un sistema de pagos en tiempo real, incluso en un entorno controlado.

Id.	Descripción
RNF-01	El sistema debe incluir documentación detallada sobre su arquitectura, componentes y cómo desplegarlo, para facilitar su comprensión y mantenimiento.
RNF-02	Toda la comunicación entre los actores debe estar cifrada. Además, cada transición de estado de la solicitud RTP debe registrarse con una huella digital SHA-256 para garantizar la integridad y trazabilidad.
RNF-03	El prototipo debe depender exclusivamente de tecnologías portables como Python 3.12 para el backend y SQLite para la persistencia de datos, asegurando su ejecución en cualquier entorno compatible.

Cuadro 5: Requisitos no funcionales

Los requisitos enlazan con la implememtación 4 y con pruebas de validación incluidas en el apartado 5. Además, cada transición de estado se acompaña de un identificador único para facilitar el seguimiento en los logs.

3.4 Casos de uso

Los casos de uso describen las interacciones clave entre los actores y el sistema para cumplir con los requisitos funcionales del esquema. A continuación, se presentan dos casos de uso: CU-01 para la creación de una solicitud RTP y CU-02 para la decisión sobre dicha solicitud.

3.4.1 CU-01 Crear solicitud RTP

Este caso de uso describe cómo el *Beneficiario* inicia una solicitud de pago (*Request To Pay*, RTP) y cómo esta es procesada por el *PSP Beneficiario* para su enrutamiento al *PSP pagador*.

Actor primario *Beneficiario*

Flujo principal

1. El *Beneficiario* envía una solicitud HTTP `POST /rtp` con los datos requeridos: IBAN del pagador, importe y concepto.
2. El *PSP Beneficiario* valida la sintaxis y el formato de la solicitud.
3. Si es válida, el *PSP Beneficiario* enruta la solicitud al *PSP pagador*.

4. El sistema responde con `RTP creado con exito` y un identificador único (*RTP-ID*).

Escenarios alternativos

- Si el IBAN es inválido, el sistema devuelve `RTP no creado con éxito (IBAN inválido)`.

Requisitos cubiertos

- **RF-01:** Creación de la solicitud RTP por el *Beneficiario*.
- **RF-02:** Validación y enrutamiento por el *PSP Beneficiario*.

3.4.2 CU-02: Decidir solicitud RTP

Este caso de uso detalla cómo el *pagador* recibe la solicitud RTP a través del *PSP pagador*, toma una decisión (*accept* o *reject*), y cómo esta decisión se notifica a los demás actores.

Actor primario *pagador*

Flujo principal

1. El *PSP pagador* presenta la solicitud RTP al *pagador*.
2. El *pagador* selecciona *accept* o *reject* dentro del tiempo límite.
3. El *PSP pagador* registra la decisión y notifica a los demás actores mediante eventos.

Escenarios alternativos

- Si el *pagador* no responde a tiempo, la solicitud se rechaza automáticamente.

Requisitos cubiertos

- **RF-03:** Presentación de la solicitud al *pagador* por el *PSP pagador*.
- **RF-04:** Decisión del *pagador* en un tiempo máximo.
- **RF-05:** Notificación de la decisión al *Beneficiario*.

Los flujos completos se ilustran en los diagramas de secuencia (Fig. 5 y Fig. 6).

3.5 Modelo de datos

El esquema de la base de datos, representado en el modelo entidad-relación (ER) de la Figura 7, está diseñado para soportar el flujo del esquema SRTP. Consta de tres tablas principales: `Actor`, `RTP` y `Log`, cada una con atributos específicos para almacenar la información necesaria del sistema.

Las tablas que componen el modelo de datos son las siguientes:

- **Actor** (`id`, `username`, `password`, `name`, `role`, `photo_url`, `iban`, `balance`, `psp_id`) — Almacena información sobre los actores involucrados en el sistema, como beneficiarios, pagadores y proveedores de servicios de pago (PSP). Sus atributos son:
  - `id`: Identificador único del actor (clave primaria).
  - `username`: Nombre de usuario único para autenticación.

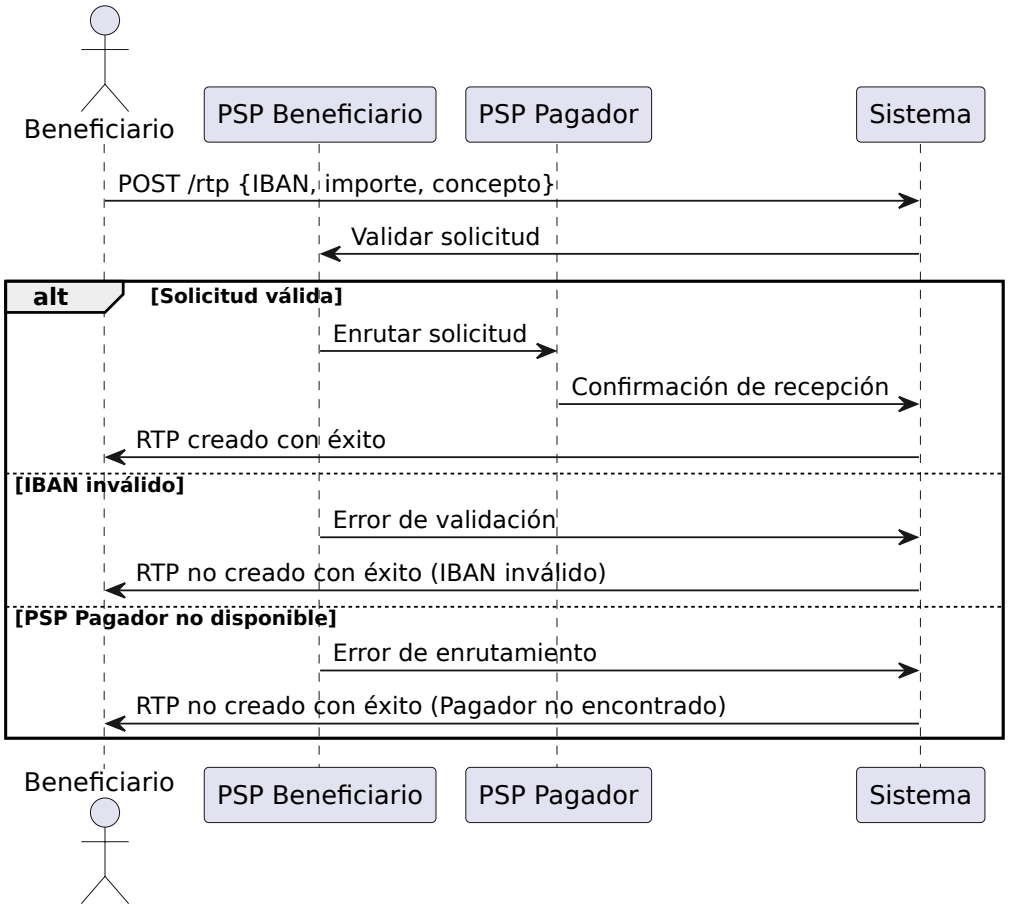

Figura 5: Diagrama de secuencia para CU-01 (crear solicitud RTP).

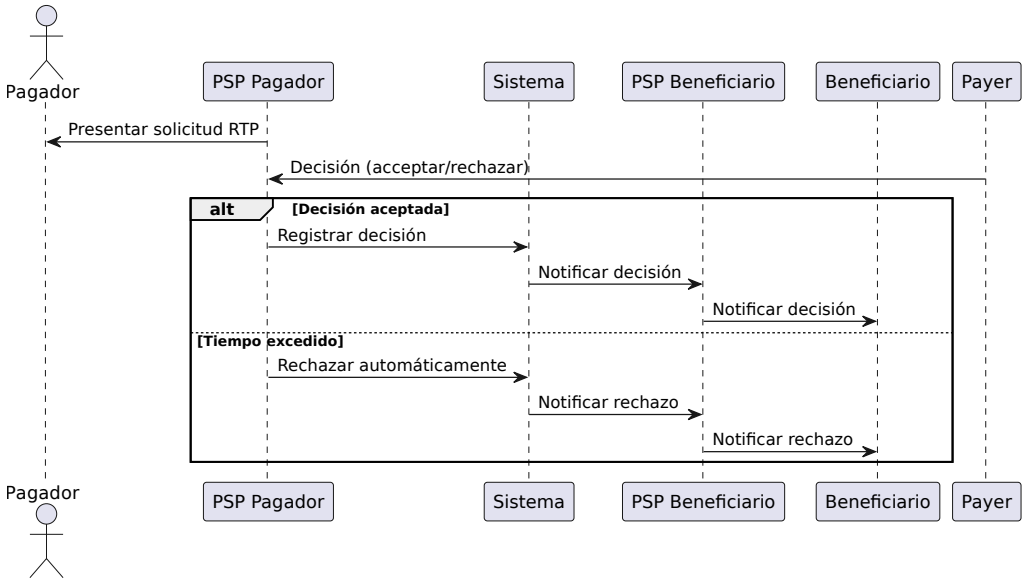

Figura 6: Diagrama de secuencia para CU-02 (decisión final RTP).

  - `password`: Contraseña del actor.
  - `name`: Nombre completo del actor.
  - `role`: Rol del actor, que puede ser *beneficiary*, *psp_beneficiary*, *psp_payer* o *payer*.
  - `photo_url`: URL o base64 de la foto del actor (opcional).
  - `iban`: Número de cuenta bancaria internacional (IBAN).
  - `balance`: Saldo disponible del actor.
  - `psp_id`: Referencia al actor que actúa como PSP (clave foránea a `Actor`).
- **RTP** (`id`, `iban`, `amount`, `status`, `timestamp`, `beneficiary_id`, `psp_beneficiary_id`, `psp_payer_id`, `payer_id`, `concept`) — Registra las solicitudes de pago (*Request To Pay*). Sus atributos son:
  - `id`: Identificador único de la solicitud (clave primaria).
  - `iban`: IBAN del pagador.
  - `amount`: Monto de la solicitud de pago.
  - `status`: Estado actual de la solicitud (e.g., çreado").
  - `timestamp`: Fecha y hora de creación de la solicitud.
  - `beneficiary_id`: Referencia al actor beneficiario.
  - `psp_beneficiary_id`: Referencia al PSP del beneficiario.
  - `psp_payer_id`: Referencia al PSP del pagador.
  - `payer_id`: Referencia al actor pagador.
  - `concept`: Descripción o motivo de la solicitud de pago.
- **Log** (`id`, `rtp_id`, `old_status`, `new_status`, `timestamp`, `hash_value`) — Almacena el historial de cambios de estado de las solicitudes RTP. Sus atributos son:

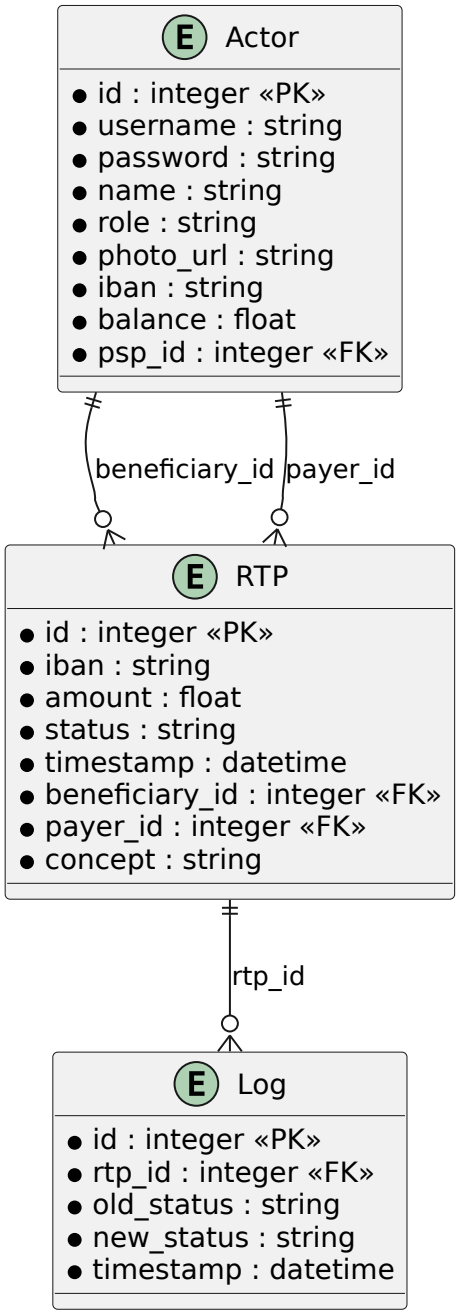

Figura 7: Modelo entidad-relación del prototipo.

- `id`: Identificador único del registro (clave primaria).
- `rtp_id`: Referencia a la solicitud RTP asociada.
- `old_status`: Estado anterior de la solicitud.
- `new_status`: Nuevo estado de la solicitud.
- `timestamp`: Marca de tiempo del cambio de estado.
- `hash_value`: Hash para verificar la integridad del registro.

El atributo `hash_value` de la tabla `Log` se calcula utilizando el algoritmo SHA-256 sobre una concatenación de campos clave, como `rtp_id`, `old_status`, `new_status` y `timestamp`. Este hash asegura la integridad de cada entrada, permitiendo detectar cualquier modificación no autorizada en el historial de estados.

Las relaciones entre las tablas están definidas mediante claves foráneas: - En `RTP`, los campos `beneficiary_id`, `psp_beneficiary_id`, `psp_payer_id` y `payer_id` referencian a `Actor.id`. - En `Log`, `rtp_id` referencia a `RTP.id`. - En `Actor`, `psp_id` referencia a otro `Actor.id`, estableciendo una relación recursiva para indicar el PSP asociado.

Todas las claves foráneas están configuradas con *`ON DELETE CASCADE`*, lo que significa que la eliminación de un registro en una tabla padre (como `Actor` o `RTP`) provocará la eliminación automática de los registros dependientes en las tablas hijas (como `RTP` o `Log`), facilitando las pruebas y el mantenimiento del prototipo.

3.6 Arquitectura lógica

La arquitectura lógica del prototipo, ilustrada en la Figura 8, organiza los componentes del sistema en capas para garantizar una separación clara de responsabilidades y facilitar la escalabilidad y el mantenimiento. El diseño se basa en un enfoque modular que permite simular el flujo del esquema SRTP en un entorno distribuido, utilizando tecnologías modernas como REST y WebSocket para la comunicación y SQLAlchemy para la persistencia.

3.6.1 Componentes principales

El sistema se estructura en tres componentes principales:

- **Gateway REST**: Actúa como punto de entrada para las solicitudes HTTP, gestionando la validación inicial de las peticiones y enrutándolas al componente adecuado. Este componente asegura que las solicitudes cumplan con los formatos esperados antes de procesarlas.
- **Service RTP**: Encapsula la lógica de negocio del esquema SRTP, incluyendo la creación, validación, enrutamiento y decisión de solicitudes de pago. Interactúa con la base de datos para almacenar y recuperar información.
- **Notification Hub**: Mantiene canales WebSocket abiertos con los cuatro actores del modelo de cuatro esquinas (Beneficiary, PSP Beneficiary, PSP Payer y Payer), enviando notificaciones en tiempo real sobre eventos como la creación o decisión de una solicitud RTP.

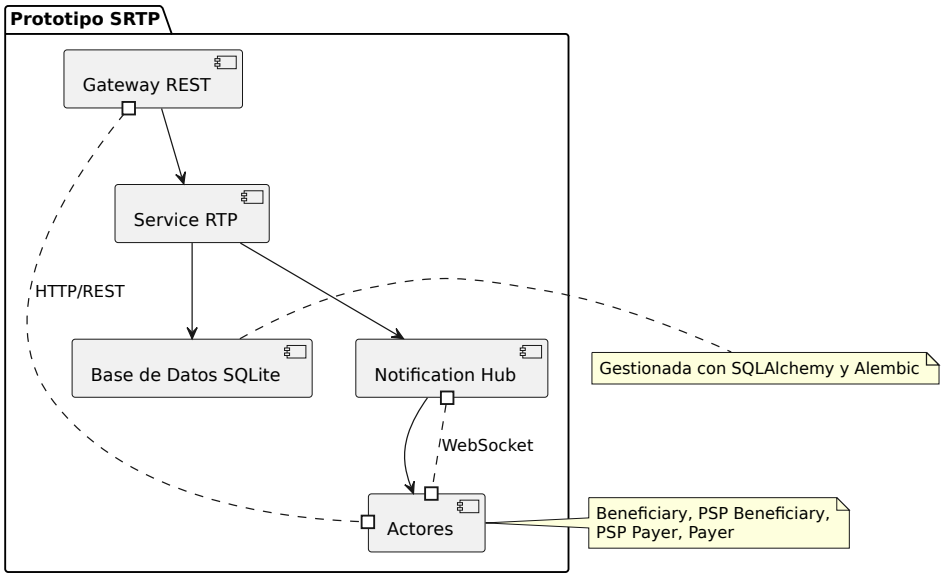

Figura 8: Arquitectura lógica y flujo de dependencias del prototipo.

3.6.2 Flujo de interacción lógico

El ciclo **SRTP** se articula en los cuatro pasos lógicos que se detallan a continuación:

1. **Recepción** – El *Gateway* recibe la petición `POST /rtp` y la delega al *Service RTP*.
2. **Procesamiento** – El *Service RTP* aplica la lógica de negocio y actualiza el estado `created` → `validated` → `routed` ....
3. **Publicación** – Tras cada transición de estado válida, el servicio emite un evento de dominio.
4. **Notificación** – El *Notification Hub* reenvía dicho evento, vía WebSocket, a las salas correspondientes (*psp_beneficiary*, *psp_payer*, *payer* o *beneficiary*).

Con este patrón se obtienen las siguientes ventajas:

- Los *clientes* no interrogan al servidor; reciben notificaciones *push*.
- La *consistencia* del estado queda centralizada en el *Service RTP*.
- El sistema puede escalar horizontalmente añadiendo réplicas del servicio sin romper el contrato público.

3.7 Emulación del flujo RTP

El objetivo de este apartado es describir cómo el prototipo recrea, de extremo a extremo, el ciclo de vida de una solicitud RTP dentro del bloque **Service RTP**, único responsable de

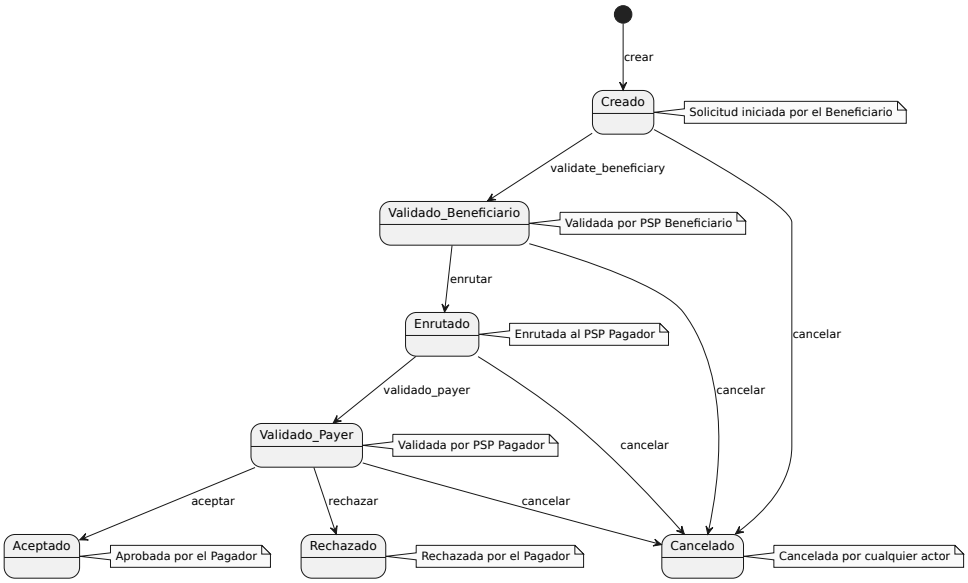

Figura 9: Máquina de estados del objeto RTP.

aplicar la lógica de negocio y salvaguardar la coherencia del estado global. A diferencia de otros componentes, este servicio actúa como *single source of truth*: valida, persiste, publica eventos de dominio y rechaza cualquier transición inválida.

3.7.1 Máquina de estados del objeto RTP

La Figura 9 sintetiza el recorrido que puede seguir una solicitud: **Creado** → **Validado Beneficiario** → **Enrutado** → **Validado Payer** → **(Aceptado** | **Rechazado** | **Cancelado)**. Cada arista representa una operación expuesta por la API (por ejemplo, `create`, `validate_beneficiary`, `route`, `validate_payer`, `accept`, `reject`, `cancel`) y sólo puede ejecutarse si el objeto se encuentra en el estado de origen indicado.

3.7.2 Transiciones y reglas de negocio

Creación (`create`) El *Beneficiary* emite un `POST /rtp`. **Service RTP** valida sintaxis y semántica (IBAN, ISO 4217, etc.), calcula un hash SHA-256 para evitar duplicados y persiste el registro como CREADO.

Validación del PSP Beneficiary (`validate_beneficiary`) El PSP del beneficiario comprueba el cumplimiento de las reglas SRTP (formato, AML/KYC básico). Si falla, genera un `reject` inmediato con motivo normalizado.

Enrutado (`route`) Una vez validada, la solicitud se envía al PSP *Payer*. El estado avanza a ENRUTADO y se incorpora un `timestamp` a la traza.

Validación del PSP Payer (`validate_payer`) El PSP del pagador confirma que el cliente puede recibir la solicitud. La aprobación conduce a VALIDADO_PAYER; de lo contrario, devuelve `reject`.

Decisión del Payer (`accept` / `reject`) El pagador dispone de un máximo de diez segundos para responder. La aceptación transita a ACEPTADO; la denegación, a RECHAZADO.

Cancelación (`cancel`) Cualquiera de los cuatro actores puede solicitar la anulación antes de la decisión final, llevando el flujo a CANCELADO.

Todas las reglas están encapsuladas en métodos de dominio homogéneos en firma y manejo de errores, favoreciendo la mantenibilidad y la trazabilidad entre diseño y código.

3.7.3 Condiciones excepcionales y *timeouts*

Si el pagador no responde antes de la **fecha/hora de expiración** definida en la solicitud, el sistema ejecuta una transición automática a CANCELADO. Del mismo modo, cualquier error técnico de un PSP desemboca en un `reject`, manteniendo la consistencia global sin intervención manual.

4 Implementación

Este apartado traduce los principios definidos en el Diseño (Secc. 3) a código ejecutable. Cada subsección muestra la correspondencia entre los artefactos lógicos y su concreción en Python / JavaScript, de modo que se pueda verificar cómo se satisfacen los requisitos **RF** y **RNF** enumerados anteriormente.

4.1 Visión general de la arquitectura y organización del código

La implementación del prototipo para simular un proveedor del esquema SRTP requiere una arquitectura robusta y una organización del código que garanticen la claridad, la mantenibilidad y la alineación con los requisitos establecidos en los capítulos anteriores. Este apartado ofrece una descripción detallada de la arquitectura general del sistema, explicando cómo se han estructurado sus componentes y cómo estos interactúan para cumplir con los objetivos del proyecto. Asimismo, se presenta la organización del código fuente, incluyendo la estructura de directorios, para ilustrar cómo se ha traducido el diseño conceptual del capítulo 3 en una implementación práctica. Esta sección no solo busca documentar las decisiones técnicas tomadas, sino también demostrar cómo el sistema aborda las limitaciones del SDD identificadas en la sección 1.1, como la necesidad de procesos más ágiles y en tiempo real.

La arquitectura adoptada se fundamenta en principios de modularidad, escalabilidad y separación de preocupaciones, lo que permite que cada parte del sistema tenga un propósito claro y que las interacciones entre componentes sean eficientes. A continuación, se describen los elementos clave de la arquitectura general, seguidos por una explicación de la organización del código y su estructura de directorios, acompañada de ejemplos concretos que refuerzan la conexión entre el diseño y la implementación.

4.1.1 Componentes del servidor

- **Capa de persistencia**: Esta capa se encarga de gestionar los datos del sistema utilizando un enfoque basado en modelos ORM (Object-Relational Mapping). Los modelos definidos aquí, como la entidad `RTP` que representa una solicitud de pago, están alineados con el diseño de la base de datos descrito en la sección 3.5. La persistencia asegura que la información se almacene de forma estructurada y sea accesible para las operaciones del SRTP.
- **Capa de servicios de negocio**: Aquí reside la lógica central del prototipo, implementando las transiciones de estado del flujo SRTP (creación, validación, enrutamiento y decisión). Cada servicio está diseñado como un módulo independiente que encapsula las reglas de negocio, garantizando que las operaciones cumplan con los requisitos funcionales y no funcionales establecidos en el capítulo 3.
- **Capa de interfaz pública**: Expone las funcionalidades del sistema mediante una API REST y un sistema de comunicación en tiempo real basado en WebSockets. La API REST ofrece endpoints bien definidos para operaciones síncronas, mientras que los WebSockets permiten notificaciones instantáneas a los actores, abordando directamente las ineficiencias de los ciclos de cobro del SDD mencionadas en la sección 1.1.1.

Cliente El cliente es una aplicación web desarrollada con Vue.js, seleccionada por su capacidad para crear interfaces reactivas y dinámicas. Esta aplicación permite a los usuarios visualizar y gestionar solicitudes RTP, desde la creación por parte del beneficiario hasta la aceptación o rechazo por el pagador. La comunicación con el servidor se realiza a través de la API REST para operaciones como la creación de solicitudes, y mediante WebSockets para actualizaciones en tiempo real, como cambios de estado.

Interacción entre componentes La interacción entre el cliente y el servidor es un aspecto crítico del sistema. Por ejemplo, cuando un beneficiario crea una solicitud RTP, el cliente envía una petición a través de un endpoint REST al servidor, que procesa la solicitud en la capa de servicios y la almacena en la capa de persistencia. Una vez procesada, el servidor notifica a los actores relevantes mediante WebSockets, actualizando la interfaz del cliente en tiempo real. Este flujo refleja las decisiones de diseño de la sección 3.3, donde se priorizó la agilidad y la inmediatez en las operaciones.

4.1.2 Estructura de directorios

La organización del código fuente ha sido diseñada para reflejar la arquitectura en capas del sistema y facilitar tanto el desarrollo como el mantenimiento del prototipo. La estructura de directorios de la figura 10 está pensada para que los desarrolladores puedan localizar rápidamente los componentes clave y comprender su propósito dentro del proyecto. A continuación, se detalla la estructura principal :

Backend. Los ocho módulos Python se bastan para ofrecer una capa de servicios coherente, auditable y sin dependencias circulares:

- `app.py` actúa como *bootstrap* del marco Flask; instancia la aplicación, registra los *blueprints*, inicializa la base de datos y arranca el *Notification Hub* basado en Socket.IO, de modo que HTTP y WebSocket comparten proceso y puerto.
- `config.py` encapsula los parámetros de despliegue y proporciona valores seguros por defecto para ejecutarse en entornos de desarrollo sin intervención manual.
- `ext_socketio.py` mantiene una única instancia global de `SocketIO`; centraliza la creación de salas y la emisión de eventos para evitar duplicidades y facilitar un eventual cambio de motor de concurrencia.
- `models.py` define las entidades *Actor*, *RTP* y *Log* con SQLAlchemy, garantizando la integridad referencial y sellando cada transición con un hash SHA-256 que permite auditar el histórico.
- `routes.py` expone los cinco endpoints REST que materializan la máquina de estados SRTP; cada manejador se limita a validar parámetros y delegar la lógica en los servicios.
- `routes_actors.py` ofrece operaciones auxiliares—registro y consulta de actores—que, aun no formando parte del flujo SRTP, resultan imprescindibles para montar un entorno de pruebas auto-contenido.

- `services.py` orquesta la lógica de dominio: comprueba pre-condiciones, transforma el estado, persiste los cambios y publica las notificaciones correspondientes; de esta forma aísla los detalles del transporte (REST o WebSocket) y favorece las pruebas unitarias.
- `utils.py` y `utils_roles.py` concentran utilidades transversales: la primera agrupa funciones de apoyo (p. ej. `cambiar_estado_rtp` o la validación de IBAN), mientras que la segunda implementa el decorador `@role_required`, responsable de aplicar el control de acceso antes de ejecutar cada endpoint.

Frontend. Los recursos estáticos se limitan a cinco archivos, suficientes para construir una SPA ligera pero reactiva:

- `index.html` sirve de punto de entrada. Además de la cabecera de navegación, contiene el contenedor <`div id=root"`> donde los módulos JavaScript inyectan las vistas solicitadas.
- `RTP.html` almacena la plantilla que representa el *workflow* SRTP; se carga bajo demanda para evitar penalizar el *time-to-first-paint*.
- `styles.css` define la paleta corporativa, el *grid* de disposición y las transiciones que dotan de fluidez a los cambios de estado.
- `app.js` gestiona la autenticación, el enrutado de vistas y la apertura de la conexión WebSocket, mostrando indicadores de estado (conectado, reconectando, ...) sin recargar la página.
- `RTP.js` implementa toda la lógica de cliente relacionada con las solicitudes: construcción de formularios, llamadas al API REST y actualización en tiempo real de la tabla de notificaciones cuando llegan los eventos *push*.

Síntesis y trazabilidad. Esta disposición jerárquica refleja en el código la arquitectura conceptual del capítulo 3: la capa de presentación (*frontend*) permanece aislada de la de negocio (*backend*); la única superficie de contacto es la API REST y el canal WebSocket, ambos documentados y versionados. Gracias a ello, cualquier sustitución—cambiar el motor de base de datos, desplegar la SPA en un CDN o servir las peticiones HTTP detrás de un WAF—puede acometerse sin modificar las capas restantes, cumpliendo así los RNF de portabilidad y mantenibilidad establecidos para el prototipo.

Esta estructura promueve la modularidad y el aislamiento de responsabilidades. Por ejemplo, la separación entre **/models** y **/services** permite modificar la lógica de negocio sin afectar la persistencia.

4.1.3 Satisfacción de requisitos

Este subapartado tiene como objetivo demostrar de manera detallada cómo el prototipo implementado cumple con cada uno de los **requisitos funcionales (RF)** y **no funcionales (RNF)** establecidos en el capítulo 3. Para cada requisito, se proporciona una explicación de su propósito dentro del flujo del esquema SRTP, se referencian los fragmentos de código que materializan la solución y se indica la ruta del archivo donde se aloja dicha

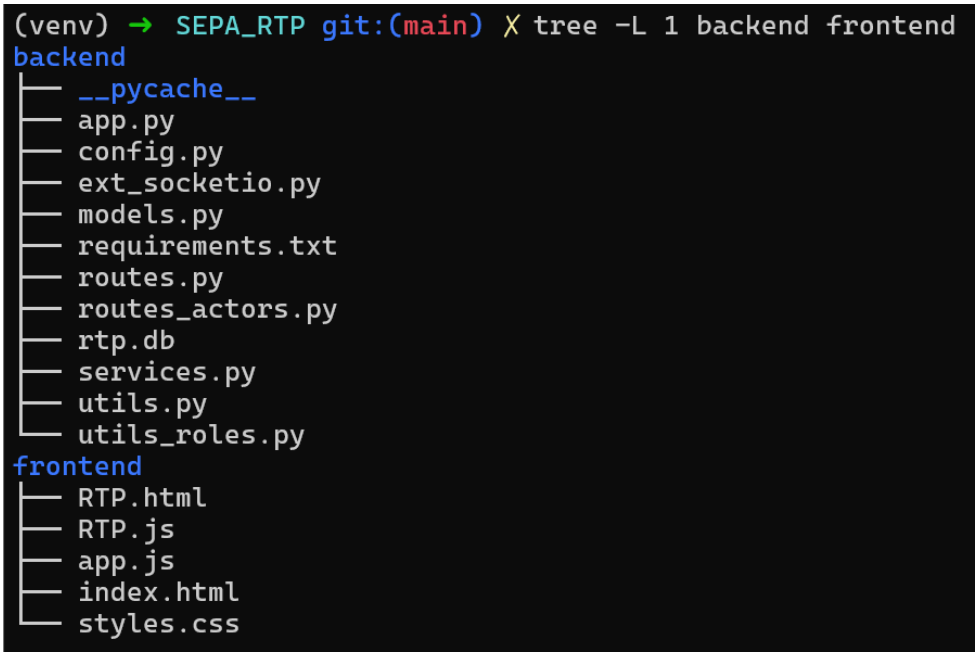

Figura 10: Estructura en árbol del directorio

lógica. Los fragmentos de código se han incluido en el *Índice de listados* al final de la memoria para facilitar su localización y consulta.

La estructura de este apartado sigue el orden de los requisitos tal como fueron definidos, comenzando por los funcionales y continuando con los no funcionales. Cada explicación busca no solo describir la implementación, sino también conectar la solución técnica con las necesidades operativas del SRTP, destacando cómo el prototipo supera las limitaciones identificadas en el SEPA Direct Debit (SDD).

RF-01 — Crear solicitud RTP El primer requisito funcional, RF-01, se refiere a la capacidad del Beneficiario para crear una solicitud de pago RTP. Este paso es crucial en el flujo SRTP, ya que inicia el proceso de cobro y establece los parámetros de la transacción, como el IBAN del deudor, el importe y el concepto del pago. En el prototipo, esta funcionalidad se implementa mediante una petición HTTP POST al endpoint `/rtp`, que es gestionada por el backend para generar y persistir la entidad RTP en la base de datos.

La lógica principal de este requisito se encuentra en el archivo `services.py`, específicamente en la función encargada de crear la solicitud RTP. El fragmento de código correspondiente, mostrado en el Listado 1, realiza varias tareas clave: valida los datos de entrada, genera un hash idempotente para garantizar la unicidad de la solicitud y persiste la entidad en la base de datos con el estado inicial *creado*. Este hash es fundamental para evitar duplicaciones y asegurar la integridad de las solicitudes, abordando así una de las ineficiencias operativas del SDD relacionadas con

la gestión manual de mandatos.

El endpoint REST que expone esta funcionalidad al frontend se define en `routes.py`, y su implementación se presenta en el Listado 2. Este handler recibe la petición del cliente, delega la lógica de negocio al servicio correspondiente y devuelve una respuesta adecuada. Desde el lado del cliente, el archivo `RTP.js` contiene la rutina JavaScript que serializa los datos del formulario y envía la petición POST al backend, como se muestra en el Listado 3. Esta interacción entre el frontend y el backend asegura que el Beneficiario pueda crear solicitudes RTP de manera intuitiva y eficiente, cumpliendo con el requisito RF-01.

RF-02 — Validar y enrutar (PSP Beneficiary) El requisito RF-02 abarca dos acciones críticas realizadas por el PSP del Beneficiario: la validación de la solicitud RTP y su posterior enrutamiento hacia el PSP del Pagador. La validación implica verificar la corrección sintáctica de los datos, como la validez de los IBAN y la coherencia del importe, mientras que el enrutamiento consiste en preparar la solicitud para su envío al siguiente actor en el flujo SRTP.

Para la validación, el prototipo utiliza un algoritmo para comprobar la validez de los IBAN, implementado en la función correspondiente del archivo `utils.py`, como se detalla en el Listado 6. Este paso es esencial para prevenir errores en las transacciones y garantizar la integridad de los datos antes de proceder con el enrutamiento.

Una vez validada la solicitud, los servicios definidos en `services.py` actualizan el estado de la entidad RTP a *validado_beneficiario* y posteriormente a *enrutado*, notificando al PSP del Pagador sobre la nueva solicitud. Estos servicios se presentan en el Listado 4. Los controladores REST que exponen estas funcionalidades se encuentran en `routes.py`, específicamente en los endpoints dedicados a la validación y enrutamiento, como se muestra en el Listado 5. En el frontend, el archivo `RTP.js` incluye la lógica para que el PSP del Beneficiario pueda disparar estas acciones a través de la interfaz de usuario, como se ilustra en el Listado 7. Esta implementación asegura que el proceso de validación y enrutamiento sea transparente y eficiente, cumpliendo con el requisito RF-02 y mejorando la agilidad operativa en comparación con los ciclos de cobro lentos del SDD.

RF-03 — Validación KYC / AML (PSP Payer) El requisito RF-03 se centra en la validación que realiza el PSP del Pagador antes de presentar la solicitud RTP al usuario final. Esta validación incluye comprobaciones de legitimidad, como la verificación de fondos suficientes y el cumplimiento de normativas KYC y AML, aunque en este prototipo se simplifica a la verificación de saldo disponible.

La lógica de este requisito se implementa en el archivo `services.py`, donde se define la función que realiza la validación del pagador. El fragmento de código correspondiente, presentado en el Listado 8, consulta el saldo del pagador y determina si la solicitud puede proceder. Si los fondos son insuficientes, la solicitud se marca como rechazada; de lo contrario, se actualiza el estado a *validado_payer*. El endpoint REST que expone esta funcionalidad se encuentra en `routes.py`, como se muestra en el Listado 9, y es consumido por el cliente a través de la lógica JavaScript en `RTP.js`, detallada en el Listado 10. Esta implementación asegura que solo las solicitudes viables financieramente lleguen al pagador, reduciendo el riesgo de transacciones fallidas y mejorando la experiencia del usuario en comparación con el SDD.

RF-04 — Decisión del pagador Una vez que la solicitud RTP ha sido validada por el PSP del Pagador, el requisito RF-04 permite al pagador tomar una decisión final: aceptar o rechazar la solicitud. Esta acción es crítica, ya que determina el resultado de la transacción y, en caso de aceptación, desencadena la transferencia de fondos.

En el prototipo, la decisión del pagador se gestiona a través de un formulario en la interfaz de usuario, implementado en `RTP.js`. El fragmento de código correspondiente, mostrado en el Listado 13, envía la decisión del usuario al backend mediante una petición POST al endpoint `/rtp/{id}/decision`. Este endpoint, definido en `routes.py` y presentado en el Listado 12, delega la lógica de negocio al servicio correspondiente en `services.py`. El servicio de decisión, detallado en el Listado 11, procesa la elección del pagador: si se acepta la solicitud y hay fondos suficientes, se descuenta el importe de la cuenta del pagador y se actualiza el estado de la solicitud a *aceptado*. En caso de rechazo o falta de fondos, se marca como *rechazado* y se genera un código de motivo adecuado. Esta implementación asegura que el pagador tenga control total sobre la transacción, cumpliendo con el requisito RF-04 y proporcionando una experiencia más segura y transparente que el SDD.

RF-05 — Notificación en tiempo real El requisito RF-05 establece la necesidad de notificar a los actores involucrados sobre los cambios de estado de las solicitudes RTP en tiempo real. Esto es fundamental para mantener a todas las partes informadas y permitir una respuesta rápida, especialmente en comparación con los procesos offline del SDD.

Para cumplir con este requisito, el prototipo utiliza WebSockets a través de la biblioteca Flask-SocketIO. El *hub* de notificaciones, configurado en `ext_socketio.py` y mostrado en el Listado 14, mantiene una *room* para cada actor, permitiendo la emisión de eventos específicos a los usuarios correctos. Por ejemplo, cuando se toma una decisión sobre una solicitud RTP, el servicio en `services.py` emite un evento `rtp_decision` al Beneficiario, como se ilustra en el Listado 15. En el frontend, el archivo `RTP.js` incluye listeners que reaccionan a estos eventos, actualizando la interfaz de usuario sin necesidad de recargar la página, como se detalla en el Listado 16. Esta implementación garantiza que las notificaciones sean instantáneas y que el sistema sea reactivo, cumpliendo con el requisito RF-05.

RF-06 — Cancelación previa a la decisión El requisito RF-06, que permite la cancelación de una solicitud RTP antes de que el pagador tome una decisión, no se ha implementado en esta versión del prototipo. Aunque está planificado para futuras iteraciones, su omisión en la versión actual se debe a que no aporta valor significativo para los objetivos de este TFG. En versiones posteriores, se añadirá un servicio `cancel_rtp_service` y un endpoint `DELETE /rtp/{id}` siguiendo la estructura de los requisitos anteriores.

RNF-01 — Documentación interna El requisito no funcional RNF-01 exige una documentación interna adecuada del código y del sistema. Este requisito se cumple mediante la presente memoria, que describe detalladamente el diseño y la implementación del prototipo, así como a través de los comentarios incluidos en todos los módulos Python del proyecto. No se requiere un fragmento de código específico para este requisito, ya que la documentación es transversal a todo el desarrollo.

RNF-02 — Seguridad y trazabilidad El requisito RNF-02 se centra en la seguridad y

la trazabilidad de las operaciones dentro del sistema. En el prototipo, la seguridad se delega parcialmente al proxy inverso (Nginx), que gestiona el cifrado TLS. Sin embargo, la trazabilidad se implementa directamente en el backend mediante el cálculo de un hash SHA-256 para cada cambio de estado de las solicitudes RTP. Este hash, generado en la función correspondiente de `utils.py` y mostrado en el Listado 17, asegura la integridad de los registros de transiciones de estado. Cada cambio se persiste en la tabla `Log` de la base de datos, junto con un sellado temporal, como se define en el modelo ORM de `models.py` presentado en el Listado 18. Esta implementación proporciona una auditoría completa de las operaciones, cumpliendo con el requisito de trazabilidad.

RNF-03 — Portabilidad (Python 3.12 + SQLite) El requisito RNF-03 exige que el sistema sea portable y fácil de desplegar en diferentes entornos. Para ello, la configuración de la base de datos y otros parámetros se centralizan en el archivo `config.py`, donde se define la URI de la base de datos SQLite, como se muestra en el Listado 19. Además, el archivo `requirements.txt`, presentado en el Listado 20, lista todas las dependencias del proyecto con versiones específicas, garantizando una instalación reproducible. La secuencia de arranque del servidor, incluyendo la creación de la base de datos y la carga de actores predefinidos, se gestiona en `app.py`, como se ilustra en el Listado 21. Esta configuración permite que el prototipo sea fácilmente portable y desplegable en cualquier sistema que soporte Python 3.12 y SQLite, cumpliendo con el requisito RNF-03.

En resumen, este subapartado ha demostrado cómo cada requisito funcional y no funcional ha sido abordado en el prototipo, ya sea mediante la implementación de lógica específica, configuraciones de despliegue o documentación adecuada. La trazabilidad entre los requisitos y el código asegura que el sistema no solo funcione correctamente, sino que también cumpla con los estándares de calidad y operatividad establecidos en el capítulo 3.

4.2 Persistencia y modelos ORM

Este apartado detalla la capa de persistencia del prototipo, abordando dos aspectos fundamentales: (i) la **elección del motor de base de datos** que soporta el sistema y (ii) la **descripción de los tres modelos de dominio** que se persisten mediante SQLAlchemy. Estas decisiones son esenciales para garantizar la fiabilidad, portabilidad y escalabilidad del sistema, alineándose con los requisitos establecidos en el Capítulo 3.

Motor de base de datos Para el desarrollo y las pruebas del prototipo, se ha seleccionado **SQLite 3** como motor de base de datos, debido a sus características que lo hacen idóneo para este contexto. Los motivos principales de esta elección se detallan a continuación:

1. **Cero configuración**: SQLite no requiere configuración previa, lo que simplifica su uso en entornos de desarrollo y pruebas. El archivo de la base de datos, `rtp.db`, se genera automáticamente al iniciar la aplicación y puede eliminarse o recrearse fácilmente, facilitando las pruebas automáticas en pipelines de integración y despliegue

continuo (CI/CD). Esto permite una iteración rápida sin la complejidad de administrar un servidor dedicado.

2. **Compatibilidad con SQLAlchemy 2**: SQLite se integra perfectamente con SQLAlchemy 2, el ORM empleado en este proyecto. Gracias a la API declarativa de SQLAlchemy, las interacciones con la base de datos quedan abstraídas, permitiendo que el código sea portable a otros motores como PostgreSQL con solo modificar la variable `SQLALCHEMY_DATABASE_URI` en el archivo `config.py` (véase Listado 19). Esta flexibilidad asegura una posible escalabilidad futura sin necesidad de refactorizar el código.
3. **Licencia y footprint**: SQLite es de dominio público y su biblioteca ocupa menos de 1 MB, lo que lo hace extremadamente ligero y portable. Esta característica cumple con el requisito no funcional RNF-03, que exige que el sistema sea fácilmente desplegable en diferentes entornos, reduciendo la complejidad y los recursos necesarios para su ejecución.

La inicialización del motor de base de datos y la creación de las tablas se realizan automáticamente durante el proceso de *bootstrap* de la aplicación, como se muestra en el Listado 21. Esto asegura un entorno consistente para el desarrollo y las pruebas.

Modelo de datos El esquema de la base de datos consta de **tres tablas** que representan los conceptos clave del dominio SRTP, definidos en el capítulo 3.5. A continuación, se describen los modelos de datos implementados:

1) **Actor**: Este modelo representa a los actores involucrados en el flujo SRTP, como pagadores, beneficiarios y proveedores de servicios de pago (PSPs). Incluye atributos como `id`, `username`, `password`, `name`, `role` (que define su función en el sistema), y opcionalmente `psp_id`, una clave foránea que referencia a otro actor que actúa como PSP. Además, cuenta con campos como `iban`, `balance` y `photo_url` para gestionar información financiera y visual. La definición completa se encuentra en el Listado 22.
2) **RTP**: El modelo `RTP` es la entidad transaccional principal, que encapsula la información de una solicitud de pago Request-to-Pay. Sus atributos incluyen `iban`, `amount`, `status` (inicialmente çreado", siguiendo la máquina de estados de la Figura 3-2), `timestamp`, y referencias a los actores involucrados (`beneficiary_id`, `psp_beneficiary_id`, `psp_payer_id`, `payer_id`). Este modelo asegura la integridad de las solicitudes y cumple con el requisito funcional RF-01. Su esquema completo está en el Listado 23.
3) **Log**: El modelo `Log` proporciona trazabilidad al registrar las transiciones de estado de las solicitudes RTP. Cada entrada incluye `rtp_id`, `old_status`, `new_status`, `timestamp` y `hash_value`. Esto satisface el requisito no funcional RNF-03 de trazabilidad. La implementación se detalla en el Listado 24.

Conclusión En conclusión, la capa de persistencia del prototipo utiliza SQLite por su simplicidad y portabilidad, mientras que SQLAlchemy abstrae las interacciones con la base de datos, ofreciendo flexibilidad para futuras migraciones. Los modelos `Actor`, `RTP` y `Log` cubren todas las necesidades del flujo SRTP, desde la gestión de actores hasta la auditoría de transacciones, sentando las bases para un sistema robusto y escalable.

4.3 Servicios de dominio

El apartado 4.3 presenta la implementación de los servicios de dominio dentro del prototipo SRTP, una capa crítica que encapsula la lógica de negocio responsable de gestionar el ciclo de vida de las solicitudes RTP. Esta capa se basa en la arquitectura lógica descrita en el capítulo 3, específicamente en la sección 3.6, donde se define el componente de servicios de aplicación como el encargado de coordinar las operaciones del sistema. Los servicios de dominio traducen los conceptos teóricos de la máquina de estados (figura 9) en funciones prácticas que aseguran la correcta ejecución de los procesos de creación, validación, enrutamiento y resolución de solicitudes RTP, cumpliendo con los requisitos funcionales establecidos.

La implementación de esta lógica se concentra en el módulo `services.py`, ubicado en el directorio `backend/` del prototipo. Este módulo actúa como el corazón operativo del sistema, proporcionando una interfaz programática que los controladores REST (definidos en `routes.py`) y los eventos WebSocket utilizan para interactuar con la base de datos y los actores del flujo SRTP. Su diseño respeta los principios de cohesión y bajo acoplamiento descritos en el capítulo 3, permitiendo que las operaciones de negocio sean independientes de los detalles de presentación o transporte.

4.3.1 Estructura de los servicios

Cada servicio de dominio sigue una estructura estandarizada que garantiza consistencia y robustez en la ejecución de las operaciones. Esta estructura incluye:

1. **Inicio transaccional**: Cada operación se encapsula en una transacción gestionada por SQLAlchemy, iniciada con `session.begin()`. Esto asegura que los cambios en la base de datos sean atómicos, evitando estados inconsistentes en caso de fallos.
2. **Validación de condiciones**: Antes de proceder, el servicio comprueba las condiciones previas, como el estado actual de la solicitud RTP en la máquina de estados y los permisos del actor que realiza la acción.
3. **Ejecución de la lógica**: Se actualizan los modelos de datos relevantes (por ejemplo, la entidad `RTP`) y se registra la operación en la tabla de auditoría `Log`, cumpliendo con el requisito no funcional de trazabilidad.
4. **Notificación**: Tras confirmar la transacción con `session.commit()`, se emite un evento a través de Socket.IO para informar a los actores involucrados, satisfaciendo el requisito funcional RF-04 de comunicación en tiempo real.

4.3.2 Descripción de los servicios implementados

A continuación, se describen los servicios clave implementados en `services.py`, cada uno asociado a una etapa del flujo SRTP:

- `crear_rtp_service`: Permite al Beneficiario iniciar una solicitud RTP. Valida los parámetros de entrada (como IBAN y cantidad) y crea un registro en la base de datos

con el estado inicial ”pendiente”. Este servicio satisface el requisito RF-01 y se muestra en el listado 1.

- `validar_beneficiario_service`: Ejecutado por el PSP del Beneficiario, este servicio verifica la solicitud en términos de formato y autenticidad, actualizando su estado a ”validada”. Cumple con RF-02 y depende de la transición ”pendiente → validada”de la máquina de estados. Se muestra en el listado 4.
- `enrutar_rtp_service`: Enruta la solicitud validada al PSP del Pagador, cambiando el estado a .enrutadaz notificando al siguiente actor. Este servicio también está vinculado a RF-02 y se detalla en el 4.
- **validatar_payer_service**: Ejecutado por el PSP del pagador. Este servicio verifica el RTP antes de enviarlo al pagador para la decisión final. Se detalla en el listado 8
- `decision_payer_service`: Permite al Pagador aceptar o rechazar la solicitud. Si se acepta, el estado pasa a çompletada”; si se rechaza, a rechazada”. Este servicio cumple con RF-03 y es el paso final del flujo básico del SRTP. Se muestra en el listado 11

4.4 API REST y WebSocket

La API REST del sistema SRTP proporciona una interfaz programática que permite la interacción con las funcionalidades principales del prototipo. Esta API sigue los principios RESTful, utilizando métodos HTTP estándar para gestionar recursos como las solicitudes Request-to-Pay (RTP) y los actores del sistema. Para organizar las rutas de manera eficiente, la implementación se estructura en dos blueprints de Flask: `rtp` y `actors`, lo que mejora la modularidad y facilita el mantenimiento y la escalabilidad del código.

4.5 Blueprints y su propósito

Los blueprints son una característica de Flask que permite dividir la lógica de las rutas en módulos independientes. En este sistema, se han definido dos blueprints:

- **Blueprint `rtp`**: Agrupa los endpoints relacionados con la gestión del ciclo de vida de las solicitudes RTP, incluyendo su creación, validación por parte de los PSPs, enrutamiento y decisión final del pagador. Además, incluye funcionalidades adicionales como la autenticación, la gestión de perfiles, la creación de actores y la obtención de logs.
- **Blueprint `actors`**: Se centra específicamente en la creación de actores del sistema, como beneficiarios, pagadores y PSPs, proporcionando una interfaz dedicada para esta funcionalidad.

Esta estructura modular permite una clara separación de responsabilidades y simplifica la extensión del sistema en el futuro.

4.6 Endpoints REST

A continuación, se presenta una tabla con todos los endpoints REST disponibles en la API, el método HTTP, la URL y una breve descripción de su funcionalidad.

Cuadro 6: Endpoints REST del sistema SRTP

Método	**URL**	**Descripción**
POST	/rtp	Crea una nueva solicitud RTP. Requiere rol `beneficiary`.
POST	/rtp/<int:rtp_id>/validate-beneficiary	Valida la RTP por el PSP del beneficiario. Requiere rol `psp_beneficiary`.
POST	/rtp/<int:rtp_id>/route	Enruta la RTP al PSP del pagador. Requiere rol `psp_beneficiary`.
POST	/rtp/<int:rtp_id>/validate-payer	Valida la RTP por el PSP del pagador. Requiere rol `psp_payer`.
POST	/rtp/<int:rtp_id>/decision	Registra la decisión final del pagador sobre la RTP. Requiere rol `payer`.
GET	/logs	Obtiene los logs del sistema para auditoría y monitoreo.
POST	/actors	Crea un nuevo actor en el sistema (beneficiario, pagador o PSP).
GET	/actors_info/<int:actor_id>	Obtiene información detallada de un actor específico.
POST	/login	Autentica a un usuario y devuelve información del actor.
GET	/profile/<int:actor_id>	Obtiene el perfil de un actor específico.
POST	/profile	Actualiza el perfil de un actor. Requiere rol `payer`.

4.7 Frontend y notificaciones

Este apartado describe la implementación del frontend del prototipo SRTP, que proporciona una interfaz de usuario intuitiva para interactuar con las funcionalidades del sistema, y el mecanismo de notificaciones en tiempo real que mantiene informados a los actores. El frontend, alineado con los requisitos funcionales del Capítulo 3, permite a los usuarios (beneficiarios, PSPs y pagadores) gestionar solicitudes RTP y acceder a sus perfiles, mientras que el sistema de notificaciones asegura una comunicación ágil, como se detalla en el requisito funcional RF-05.

4.7.1 *Frontend*

El frontend está implementado como una aplicación web de página única (SPA-like) que utiliza tecnologías modernas para ofrecer una experiencia de usuario fluida. La interfaz se basa en los siguientes componentes tecnológicos:

- **HTML5 y Bootstrap 5**: La estructura de la interfaz se define en `index.html`, utilizando Bootstrap para un diseño responsivo y componentes preestilizados como tarjetas, formularios y barras de navegación. La sección principal se divide en vistas dinámicas (dashboard, cuentas, tarjetas, perfil y RTP) que se muestran u ocultan según las acciones del usuario.
- **CSS personalizado**: El archivo `styles.css` define estilos adicionales, como gradientes de fondo, animaciones de transición (`fadeIn`) y ajustes visuales para elementos como el círculo de saldo y la foto de perfil. Las fuentes `Montserrat` y `Great Vibes` aportan una estética moderna y profesional.
- **JavaScript y Socket.IO**: La lógica del cliente se implementa en `app.js` y `RTP.js`. El primero gestiona la autenticación, el cambio de vistas y la carga de datos del perfil, mientras que el segundo se encarga de las interacciones específicas con las solicitudes RTP. Socket.IO facilita la comunicación en tiempo real con el backend.

La interfaz comienza con una pantalla de inicio de sesión (`loginSection` en `index.html`) que autentica a los actores mediante una solicitud POST al endpoint `/login`. Tras un login exitoso, se muestra la barra de navegación y el contenido principal, con un dashboard inicial que presenta información del actor (nombre, IBAN, saldo y foto). La función `cargarHomeDashboard` obtiene estos datos desde el endpoint `/profile/<actor_id>` y los formatea para su visualización, ajustando dinámicamente el tamaño de fuente del saldo con `adjustFontSize`.

La sección de Request-to-Pay, definida en `RTP.html` y cargada dinámicamente por `RTP.js`, es el núcleo de la interacción con el flujo SRTP. La función `mostrarPanelRTPporRol` ajusta la visibilidad de los paneles según el rol del usuario, mostrando formularios específicos para crear, validar, enrutar o decidir sobre solicitudes RTP. Por ejemplo, los beneficiarios pueden crear RTPs mediante `createRTPForm`, mientras que los pagadores toman decisiones con `decisionForm`, cumpliendo con los requisitos funcionales RF-01 a RF-04.

La experiencia de usuario se optimiza mediante un menú inferior (`bottomMenuSquares`) con iconos y animaciones hover, y una barra de navegación que permite alternar entre secciones. Aunque las secciones de cuentas y tarjetas son placeholders, la sección de perfil permite actualizar datos como IBAN y saldo, integrándose con el endpoint `/profile`. Esta estructura modular asegura que el frontend sea extensible para futuras funcionalidades, como se propone en el capítulo 7.

4.7.2 *Notificaciones*

El sistema de notificaciones en tiempo real, implementado con Flask-SocketIO y descrito parcialmente en los apartados 4.1 y 4.3, utiliza WebSockets para informar a los actores sobre cambios en el estado de las solicitudes RTP, cumpliendo con RF-05. En el frontend, `RTP.js`

registra listeners para eventos como `rtp_created`, `rtp_routed`, `rtp_validated_payer` y `rtp_decision`. Estos eventos actualizan una tabla de notificaciones (`notificationsTable` en `RTP.html`) que muestra el ID, monto, estado y acciones disponibles según el rol del usuario, como validar o decidir sobre una RTP. La función `renderNotificationsTable` gestiona la visualización dinámica, asegurando una interacción reactiva y eficiente.

En resumen, el frontend del prototipo SRTP ofrece una interfaz funcional y adaptable que satisface las necesidades de los actores, mientras que el sistema de notificaciones proporciona actualizaciones en tiempo real. La combinación de tecnologías modernas y un diseño basado en roles garantiza una experiencia de usuario coherente con los objetivos establecidos en el Capítulo 3.

4.8 Despliegue y configuración

Este apartado detalla el proceso de despliegue y configuración del prototipo SRTP, abarcando desde la preparación del entorno hasta la ejecución de la aplicación. El sistema está diseñado para operar en un entorno de desarrollo controlado, utilizando tecnologías ligeras y portables que cumplen con el requisito no funcional RNF-03 (portabilidad). La configuración se centraliza en archivos específicos, y el despliegue se realiza sobre un entorno virtual Python en Ubuntu 22.04.6 LTS ejecutado como WSL (Windows Subsystem for Linux) en Windows, garantizando reproducibilidad y facilidad de instalación.

4.8.1 *Preparacióndelentorno*

El prototipo se despliega en **Ubuntu 22.04.6 LTS** bajo WSL, una elección que combina la flexibilidad de un sistema Linux con la compatibilidad de Windows. Para aislar las dependencias y evitar conflictos, se utiliza un entorno virtual Python (`venv`). Los pasos iniciales para preparar el entorno son:

1. **Instalación de Python**: Se requiere Python 3.10 o superior, preinstalado en Ubuntu 22.04.6 LTS. El comando `python3 --version` verifica la versión instalada.
2. **Creación del entorno virtual**: Desde el directorio raíz del proyecto, se ejecuta `python3 -m venv venv` para crear el entorno virtual, seguido de `source venv/bin/activate` para activarlo.
3. **Instalación de dependencias**: Las dependencias del proyecto están listadas en el archivo `requirements.txt` (Listado 20), que incluye:
   - `Flask==2.2.2`: Framework web para la API REST y la gestión de rutas.
   - `Flask-SQLAlchemy==3.0.2`: ORM para interactuar con la base de datos SQLite.
   - `Flask-SocketIO`: Soporte para notificaciones en tiempo real mediante WebSockets.
   - Otras dependencias implícitas, como `sqlalchemy` y `eventlet`, necesarias para el funcionamiento del sistema.

   Las dependencias se instalan con el comando `pip install -r requirements.txt`, asegurando un entorno reproducible.

La base de datos SQLite (`rtp.db`) se genera automáticamente al iniciar la aplicación, como se describe en el apartado 4.2. No requiere configuración adicional, ya que su URI se define en `config.py`. Los sockets, gestionados por Flask-SocketIO, se inicializan en `ext_socketio.py` (Listado 14) y se configuran para permitir conexiones desde cualquier origen (`cors_allowed_origins="*"`).

4.8.2 *Configuracióndelsistema*

La configuración del prototipo se centraliza en el archivo `config.py` (Listado 19), que define parámetros clave:

- `SQLALCHEMY_DATABASE_URI`: Establece la ubicación de la base de datos SQLite (`sqlite:///rtp.db`) relativa al directorio del proyecto.
- `SQLALCHEMY_TRACK_MODIFICATIONS`: Desactivado (`False`) para optimizar el rendimiento y evitar advertencias de SQLAlchemy.

El archivo `app.py` (Listado 21) es el punto de entrada de la aplicación y configura los componentes principales:

- **Inicialización de Flask**: Se crea una instancia de `Flask` con una carpeta estática (`frontend`) para servir archivos como `index.html` y `styles.css`.
- **Configuración de SQLAlchemy y SocketIO**: Se vinculan con la aplicación mediante `db.init_app(app)` y `socketio.init_app(app)`.
- **Creación de la base de datos**: Dentro de un contexto de aplicación, se eliminan (`db.drop_all()`) y recrean (`db.create_all()`) las tablas, inicializando cuatro actores predefinidos (Mercadona, PSPMercadona, PSPalonso, Alonso) con roles y relaciones específicas, como se describe en el apartado 4.2.
- **Registro de blueprints**: El blueprint `rtp_blueprint` (definido en `routes.py`) se registra para exponer los endpoints REST.

El puerto de escucha se establece en `app.py` mediante la ejecución de `socketio.run(app, debug=True)`, que por defecto utiliza el puerto 5000. Este puerto es accesible localmente en `http://127.0.0.1:5000`. Para modificar el puerto, se puede pasar el parámetro `port` a `socketio.run`, por ejemplo, `socketio.run(app, port=8080, debug=True)`. Además, `app.py` abre automáticamente un navegador web en la URL raíz al iniciar la aplicación, facilitando las pruebas.

4.8.1 Ejecución del sistema

Para ejecutar el prototipo, se siguen estos pasos desde el directorio raíz del proyecto:

1. Activar el entorno virtual: `source venv/bin/activate`.
2. Ejecutar la aplicación: `python app.py`. Esto inicia el servidor Flask y SocketIO en el puerto 5000, crea la base de datos, inicializa los actores y sirve la interfaz web.

El modo de depuración (`debug=True`) permite recargar automáticamente el servidor ante cambios en el código, ideal para el desarrollo. Para un entorno de producción, se recomienda usar un servidor WSGI como `gunicorn` y desactivar el modo de depuración, como se sugiere en el Capítulo 6.

4.8.2 Conclusión

El despliegue del prototipo SRTP es sencillo y portable gracias al uso de un entorno virtual Python, SQLite y Flask-SocketIO. La configuración centralizada en `config.py` y la inicialización automatizada en `app.py` garantizan un arranque consistente, mientras que el uso de Ubuntu WSL proporciona un entorno de desarrollo robusto. Esta implementación cumple con los requisitos de portabilidad y trazabilidad establecidos en el Capítulo 3, sentando las bases para un sistema escalable y mantenible.

5 Validacion

El objetivo de esta fase es demostrar, mediante evidencias reproducibles, que el prototipo desarrollado cumple con los requisitos funcionales **RF-01** a **RF-06** y los no funcionales **RNF-01** a **RNF-03**, definidos en el Capítulo 3, así como con las reglas establecidas en el *SRTP Scheme Rulebook v4.0*. Para lograr esto, se diseñó una estrategia de validación escalonada que combina pruebas manuales exploratorias, una batería de pruebas automatizadas ejecutadas con Postman, y la inspección de *logs* hashados que garantizan la trazabilidad del ciclo de vida de cada solicitud RTP. En esta sección se describe en detalle la metodología empleada, el entorno de pruebas, los escenarios cubiertos y los resultados obtenidos.

5.1 Entorno de pruebas

El entorno de pruebas se configuró utilizando las siguientes herramientas y configuraciones:

- **Herramientas**: Se utilizó Postman v10 para la construcción y envío de peticiones HTTP, SQLite 3 como base de datos, y la utilidad integrada de *Flask-SocketIO* para la inspección de eventos en tiempo real.
- **Variable de entorno `baseURL`**: Antes de crear la colección de pruebas, se definió la variable `baseURL` con la URL base del servidor (véase Fig. 11). Esto permite evitar la repetición de la URL completa en cada petición y facilita el cambio entre entornos (por ejemplo, de desarrollo a integración continua) con una sola modificación.
- **Datos semilla**: Al iniciar la aplicación, el archivo `app.py` crea automáticamente cuatro actores predefinidos. Estos actores cuentan con saldos, relaciones y roles preestablecidos, lo que permite ejecutar los flujos de prueba sin necesidad de pasos adicionales de alta de usuarios.

5.2 Metodología

La metodología de validación se estructuró en tres etapas principales:

1. **Pruebas exploratorias**: En las primeras iteraciones del desarrollo, cuando el prototipo solo exponía la API REST, se empleó un ciclo manual de *definir petición* → *enviar* → *analizar respuesta* → *refactorizar código* para depurar la lógica de negocio. Durante esta etapa, se mantuvieron *endpoints* internos, como `/actor` (véase Fig. 12), que, aunque ya no están expuestos en la interfaz web, siguen disponibles para escenarios de administración.
2. **Colección Postman**: Una vez estabilizada la API, se preparó la colección *MiAPITFG* (véase Fig. 14), que orquesta las llamadas clave del flujo RTP y verifica los códigos de estado HTTP, los cuerpos de respuesta JSON y la correcta transición de estados de la entidad `RTP`. Cada vez que se modifica el código, esta colección se ejecuta localmente.

3. **Cobertura y trazabilidad**: Todas las pruebas están enlazadas con los requisitos y la función `cambiar_estado_rtp` registra un hash de los datos críticos cada vez que se produce un cambio de estado en una solicitud RTP, garantizando así la integridad y trazabilidad de los resultados de las pruebas.

5.3 Flujo *happy path*

El flujo nominal, también conocido como *happy path*, fue validado mediante la colección Postman, siguiendo los pasos descritos a continuación. Las capturas asociadas a cada paso se enumeran entre paréntesis.

1. **Creación del RTP** (`/rtp`, Figs. 16–17): El beneficiario envía una solicitud de cobro a un pagador identificado por su IBAN.
2. **Validación y enrutado en el PSP del Beneficiario** (Figs. 18–21): Se verifica la sintaxis de la solicitud (IBAN válido, importe positivo, etc.) y, si todo es correcto, se enruta la solicitud al PSP del pagador.
3. **Validación en el PSP del Pagador** (Figs. 22–23): El PSP del pagador comprueba el saldo disponible y aplica reglas simplificadas de KYC/AML.
4. **Decisión del Pagador** (Figs. 24–25): El pagador acepta o rechaza la solicitud. La decisión se propaga en menos de 10 segundos mediante WebSocket, cumpliendo con el requisito funcional **RF-05**.

En cada una de estas fases, se comprobaron los siguientes aspectos:

- El código de estado HTTP es el adecuado (por ejemplo, 201 para creación, 200 para éxito, 400 para error).
- La consistencia del estado de la solicitud RTP en la base de datos y la emisión correcta del evento `rtp_<estado>` en la sala correspondiente de WebSocket.

5.4 Escenarios de error

Para verificar la resiliencia del sistema frente a situaciones adversas, se diseñaron y probaron tres escenarios de error comunes, todos ellos incorporados a la colección Postman para garantizar la regresión continua:

IBAN inválido o no registrado : La función `validar_iban()` rechaza la solicitud con un código de estado `400 Bad Request`. Este caso se muestra en la captura Fig. 26.

Saldo insuficiente : El PSP del pagador detecta que el saldo es insuficiente y emite un **rechazo**. La colección confirma que el estado final de la solicitud RTP es `rejected`. Véase Figs. 27–28.

Actores inexistentes : Si alguno de los actores requeridos no está presente en la base de datos, el servicio responde con `404 Not Found` y el flujo se detiene inmediatamente (Fig. 29).

5.5 Conclusiones

La estrategia de validación adoptada ha demostrado que:

- Los tres escenarios de error confirman la robustez del sistema frente a datos maliciosos o inconsistentes, alineándose con el requisito no funcional **RNF-02**.
- La integración continua impide que nuevas funcionalidades degraden la experiencia del usuario o comprometan la seguridad del sistema.

En consecuencia, el prototipo satisface las metas de validación establecidas al inicio del proyecto y proporciona una base sólida para futuras extensiones, como la integración con bancos reales o la realización de pruebas de carga, tal como se propone en el capítulo 7.

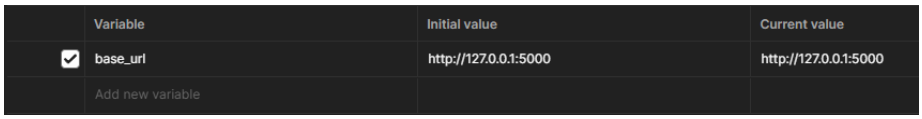

Figura 11: Variable `baseURL` en Postman

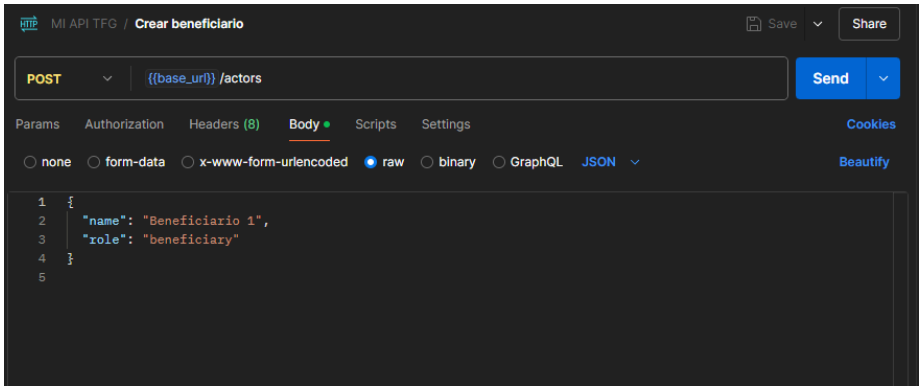

Figura 12: crearActor

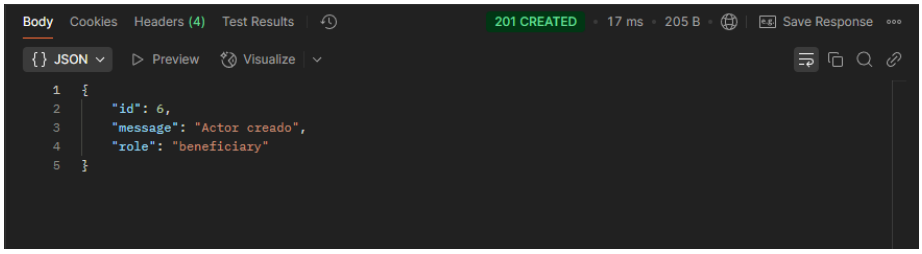

Figura 13: crearActorResponse

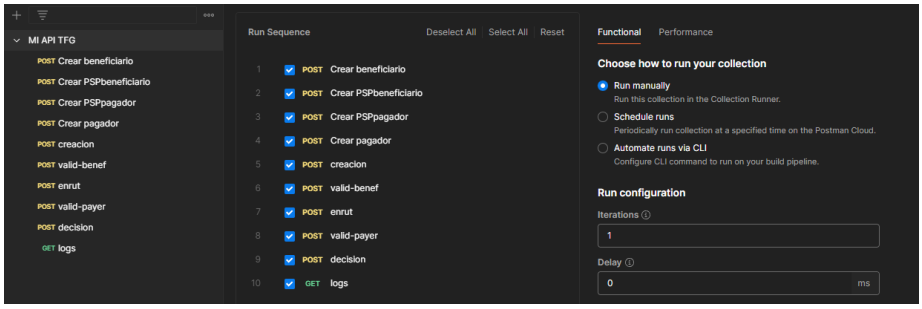

Figura 14: MiAPITFG

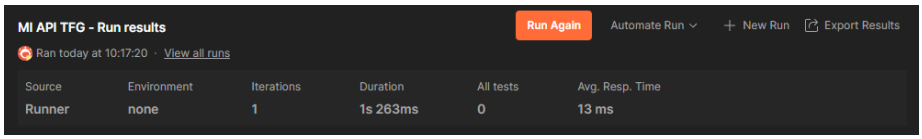

Figura 15: RunResults

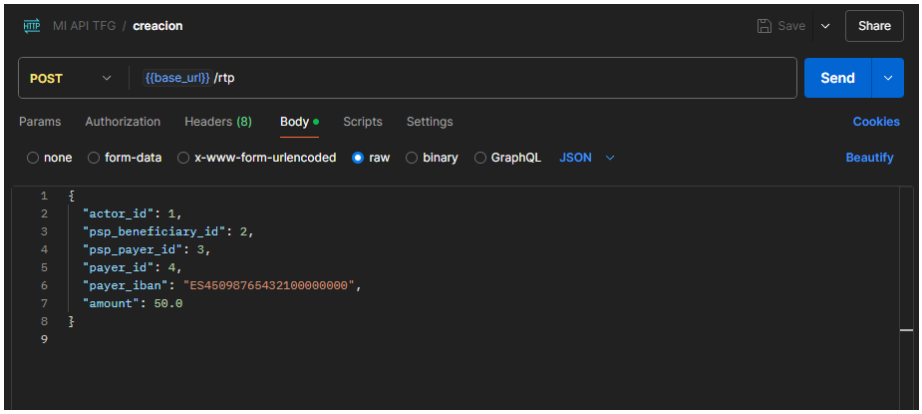

Figura 16: crearRTP

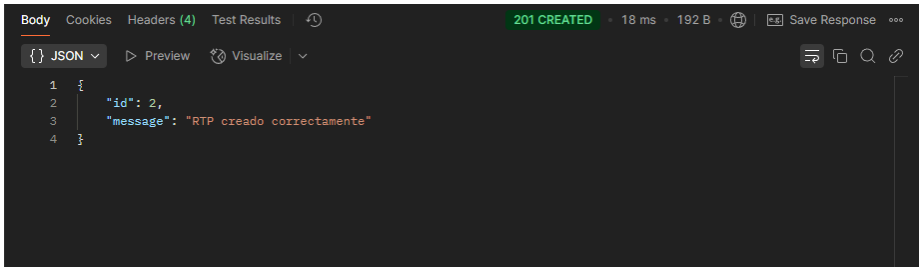

Figura 17: crearRTPResponse

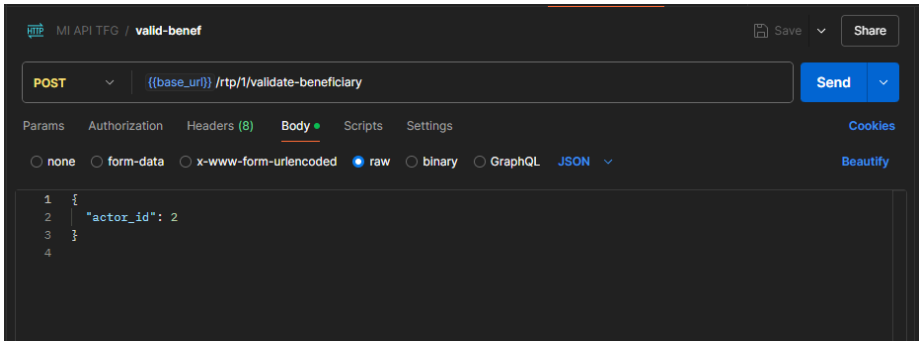

Figura 18: validarBenef

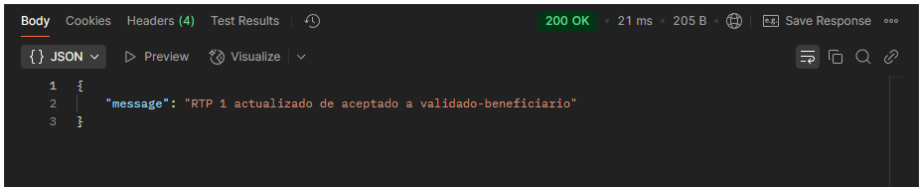

Figura 19: validarBenefResponse

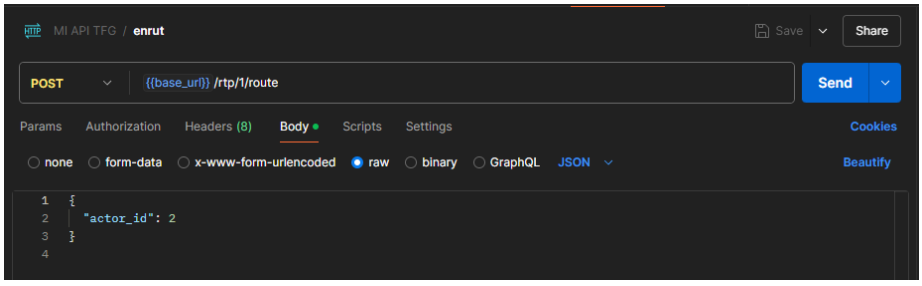

Figura 20: enrutarBenef

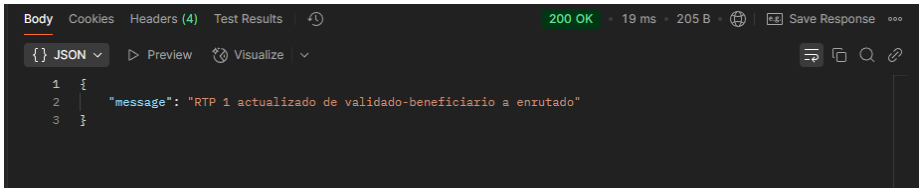

Figura 21: enrutarBenefResponse

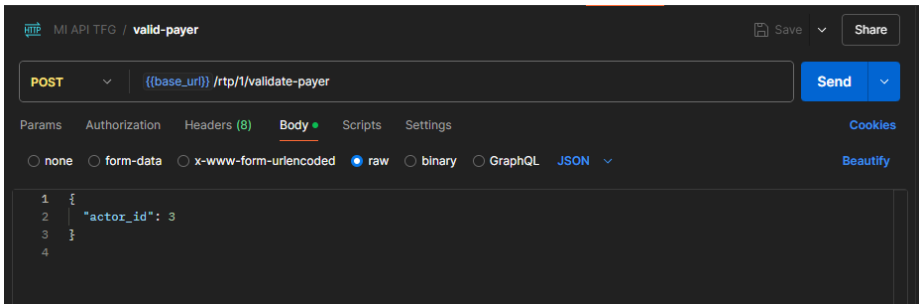

Figura 22: validarPayer

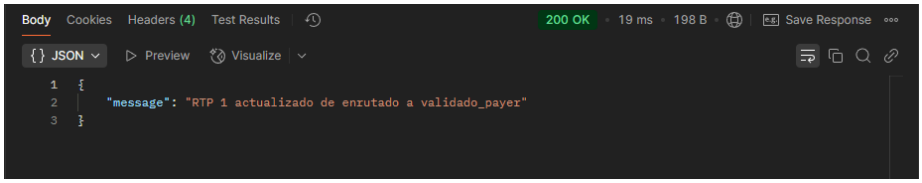

Figura 23: validarPayerResponse

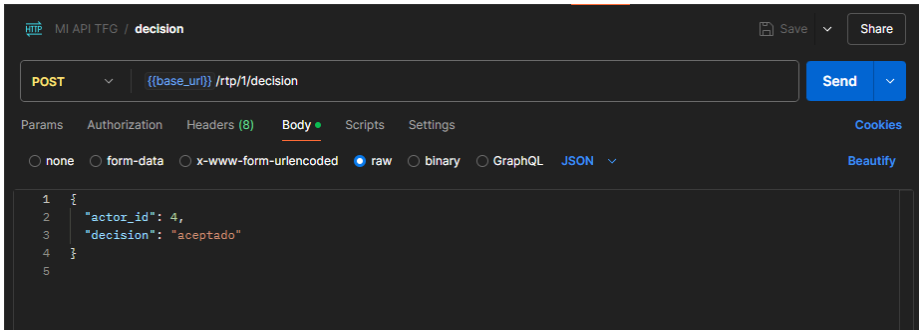

Figura 24: decision

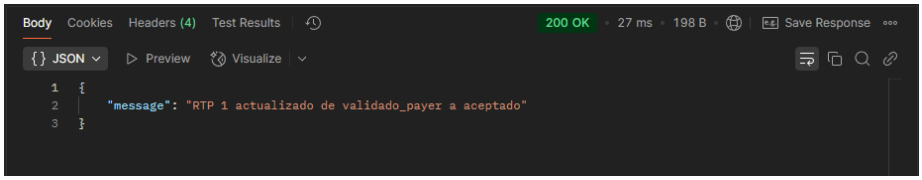

Figura 25: decisionResponse

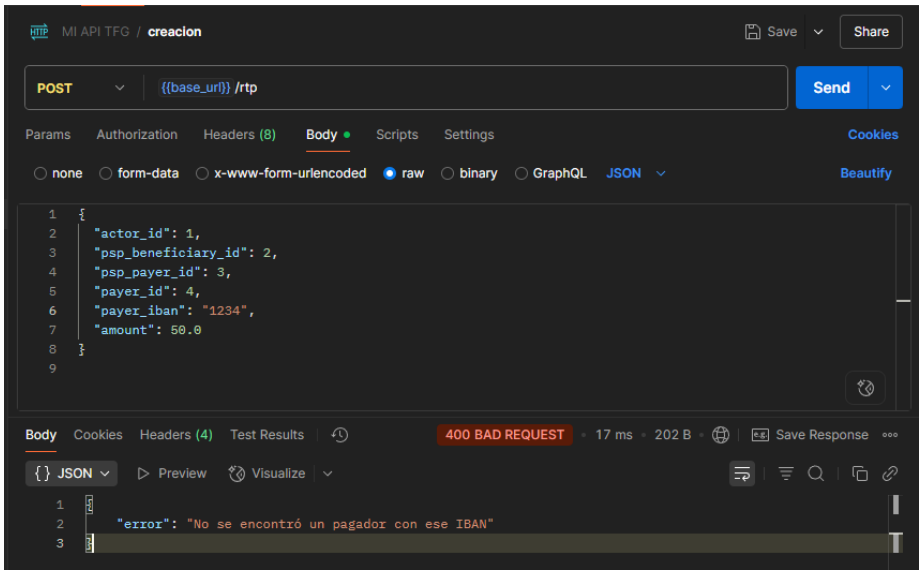

Figura 26: error1

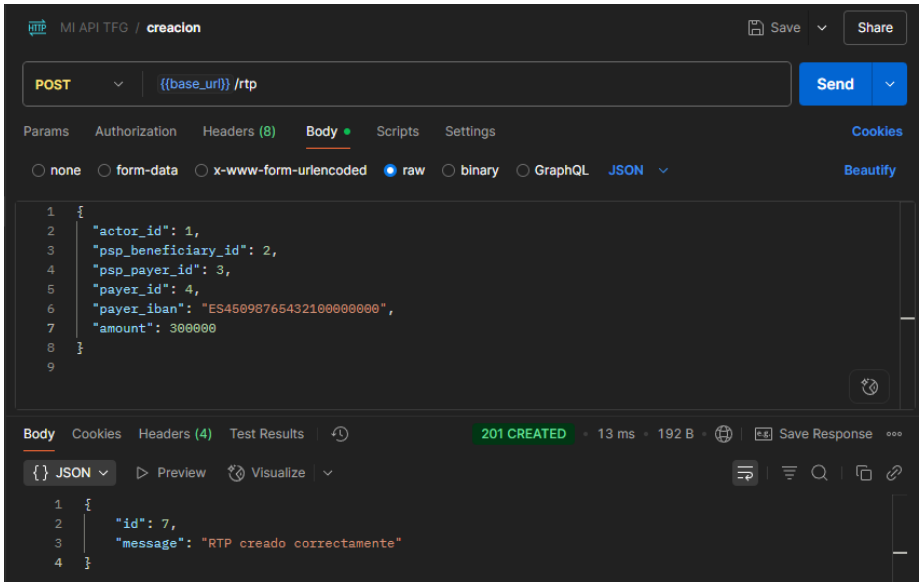

Figura 27: error2_1

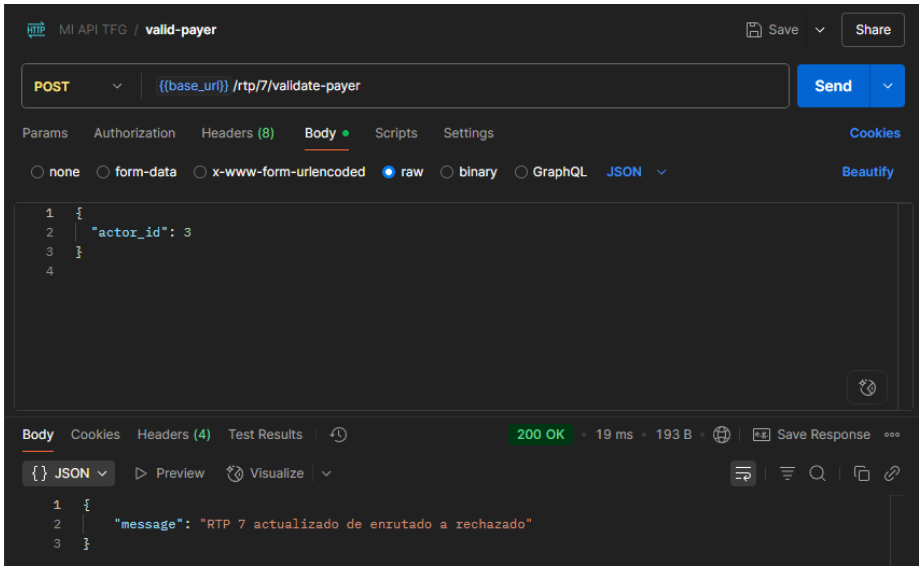

Figura 28: error2_2

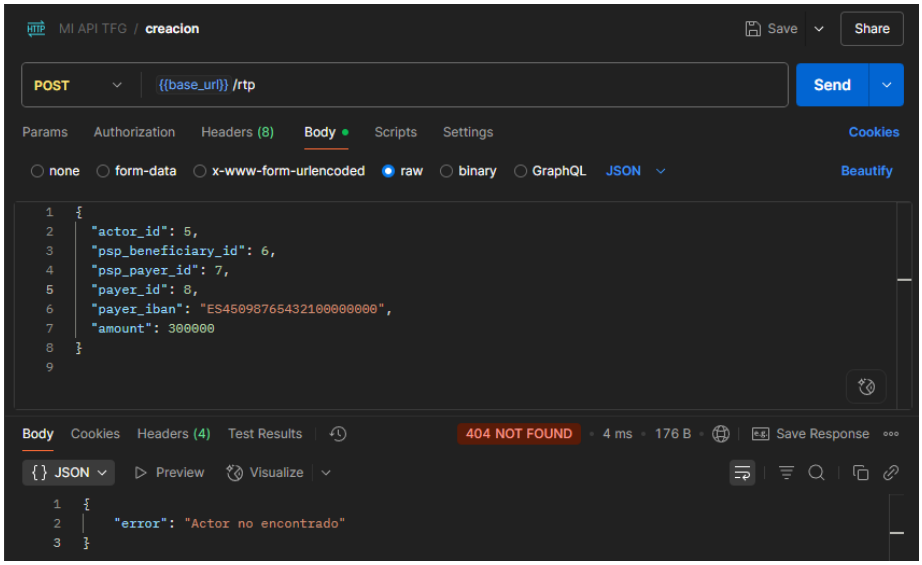

Figura 29: error3

6 Conclusiones

El desarrollo de este TFG ha representado un esfuerzo significativo que ha culminado en la creación de un prototipo funcional de un servidor HTTP basado en el esquema RTP. El objetivo principal del proyecto —diseñar e implementar un software que simulara las operaciones fundamentales del esquema SEPA RTP utilizando tecnologías actuales y accesibles— se ha alcanzado con éxito. Este trabajo no solo me ha permitido poner en práctica los conocimientos adquiridos durante mi formación en telecomunicaciones y desarrollo de software, sino también explorar un ámbito tan relevante como los sistemas de pago digitales, que desempeñan un papel crucial en la economía global de hoy.

A nivel personal, este TFG ha sido una buena oportunidad para profundizar en el ecosistema SEPA y sus diferentes esquemas de pago. En particular, he podido analizar las limitaciones del SDD y contrastarlas con las ventajas que ofrece RTP. Este aprendizaje no se ha limitado al ámbito teórico: la implementación práctica del prototipo me ha enfrentado a retos técnicos que han fortalecido mis competencias como desarrollador.

El proceso de desarrollo también ha sido una lección sobre la importancia de una metodología bien definida, lo que me permitió avanzar de manera constante, detectar errores a tiempo y ajustar los objetivos según las necesidades del proyecto. Este enfoque estructurado, combinado con una documentación detallada de cada etapa, ha sido clave para mantener el control sobre el proyecto.

El sistema RTP se posiciona como un candidato ideal paraq convertirse en un estándar en la zona SEPA, especialmente a medida que más instituciones financieras y PSP adopten el esquema y lo integren en sus operaciones. En el futuro, es probable que RTP no solo facilite los pagos entre empresas y consumidores, sino que también fomente una mayor interoprabilidad y eficiencia en el mercado financiaro europeo, contribuyendo a una economía más conectada y dinámica.

7 Líneas futuras

En cuanto al prototipo desarollado, aunque satisface plenamente los objetivos establecidos para este TFG, su potencial va mucho más allá de un simple ejercicio acádemico. Durante el proceso identifiqué varias áreas de mejora que podrían transformar este simulador en una herramienta aplicable en escenarios reales. A continuación, detallo algunas de estas oportunidades de evolución:

1. **Escalabilidad del sistema:** La base de datos SQLite, aunque suficiente para un prototipo, tiene limitaciones en térmninos de rendimiento y capacidad. Para soportar un mayor volumen de usuarios y transacciones, sería necesario migrar a un sistema más robusto como PostgreSQL o MySQL, que ofrecen mejor escalabilidad y soporte para entornos de producción.
2. **Fortalecimiento de la seguridad:** El prototipo incluye medidas básicas de autenticación, pero un sistema real requeriría estándares más altos, como la encriptación de extremo a extremo, autenticación multifactor y cumplimiento con normativas como PSD2 (Payment Services Directive 2) y GDPR (General Data Protection Regulation). Estas mejoras garantizarían la protección de los datos sensibles y la confianza de los usuarios.
3. **Interoperabilidad con sistemas bancarios:** Para que el prototipo trascienda su estado actual, sería esencial integrarlo con las APIs de bancos y PSP reales. Esto permitiría ejecutar transacciones monetarias auténticas y demostrar su utilidad en un contexto práctico, un paso crítico hacia su adopción en el mundo real.
4. **Ampliación de funcionalidades:** El sistema podría enriquecerse con características avanzadas, como soporte para pagos recurrentes (ideal para suscripciones o facturas periódicas), compatibilidad con múltiples monedas (facilitando transacciones transfronterizas), herramientas analíticas que ofrezcan estadísticas a los usuarios, y reportes detallados sobre el historial de pagos. Estas adiciones harían que el sistema fuera más versátil y atractivo para distintos tipos de usuarios.
5. **Mejora de la experiencia de usario:** Aunque el frontend actual es funcional, podría optimizarse con un diseño más moderno y accesible. Incorporar elementos como notificaciones push, un historial visual de transacciones, opciones de personalización y una interfaz adaptada a dispositivos móviles elevaría la usabilidad y la satisfacción del usuario.

Estas mejoras, aunque son algo ambiciosas, son alcanzables con el tiempo y los recursos adecuados. Implementarlas no solo incrementaría la funcionalidad del prototipo, sino que también lo alinearía con las demandas de un mercado financiero en constante camnbio, donde la innovación y la adaptabilidad son esenciales.

```python
def crear_rtp_service(data):
    beneficiary_id = data.get('actor_id')
    payer_iban = data.get('payer_iban')
    amount = data.get('amount')

    # 1) Beneficiario
    beneficiary = Actor.query.get(beneficiary_id)
    if not beneficiary or beneficiary.role != 'beneficiary':
        return {"error": "El actor no es beneficiario o no existe"}

    # 2) PSP del Beneficiario
    if not beneficiary.psp_id:
        return {"error": "El beneficiario no tiene PSP asociado"}
    psp_benef_id = beneficiary.psp_id

    # 3) Hallar el pagador por su IBAN
    payer = Actor.query.filter_by(iban=payer_iban, role='payer').first()
    if not payer:
        return {"error": "No se encontr  un pagador con ese IBAN"}

    # 4) PSP del pagador
    if not payer.psp_id:
        return {"error": "El pagador no tiene PSP asociado"}
    psp_payer_id = payer.psp_id

    # Crear el RTP
    nuevo_rtp = RTP(
        iban=payer_iban,
        amount=amount,
        beneficiary_id=beneficiary.id,
        psp_beneficiary_id=psp_benef_id,
        psp_payer_id=psp_payer_id,
        payer_id=payer.id
    )
    db.session.add(nuevo_rtp)
    db.session.commit()

    socketio.emit('rtp_created', nuevo_rtp.to_dict(),
        room=f'psp_beneficiary_{psp_benef_id}')

    return {
        "message": "RTP creado correctamente",
        "id": nuevo_rtp.id,
        #"beneficiary_id": beneficiary.id,
        #"psp_beneficiary_id": psp_benef_id,
        #"payer_id": payer.id,
        #"psp_payer_id": psp_payer_id
    }
```

Listing 1: RF-01 Crear Solicitud RTP services.py

```python

# 1. Creaci n del RTP
@rtp_blueprint.route('/rtp', methods=['POST'])
@role_required('beneficiary')
def crear_rtp():
    data = request.get_json()
```

```python
    # ID del beneficiario que est  logueado
    beneficiary_id = data.get('actor_id')
    # El IBAN del pagador
    payer_iban = data.get('payer_iban')
    # Monto
    amount = data.get('amount')

    if not payer_iban:
        return jsonify({"error": "Falta iban"}), 400

    if not amount:
        return jsonify({"error": "Falta amount"}), 400

    # Llamamos a la l gica de creaci n en services
    result = crear_rtp_service(data)
    status = 201 if "error" not in result else 400
    return jsonify(result), status
```

Listing 2: RF-01 Crear Solicitud RTP routes.py

```javascript

// Registra todos los event listeners para los formularios y botones de la
    secci n RTP
function attachRTPEventListeners() {
  // Beneficiary: Crear RTP
  const createRTPForm = document.getElementById('createRTPForm');
  if (createRTPForm) {
    createRTPForm.addEventListener('submit', function(e) {
      e.preventDefault();
      const iban = document.getElementById('ibanField').value;
      const amount =
          parseFloat(document.getElementById('amountField').value);

      const data = {
        actor_id: currentActorId,
        payer_iban: iban,
        amount: amount
      };

      fetch('http://127.0.0.1:5000/rtp', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(data)
      })
      .then(r => r.json())
      .then(result => {
        const respDiv = document.getElementById('createRTPResponse');
        respDiv.classList.remove('invisible-section');
        respDiv.innerText = JSON.stringify(result, null, 2);
      })
      .catch(err => console.error(err));
    });
  }
```

Listing 3: RF-01 Crear Solicitud RTP RTP.js

```python

def validar_beneficiario_service(rtp_id):
    rtp_obj = RTP.query.get(rtp_id)
    if not rtp_obj:
        return {"error": "RTP no encontrado"}
    result = cambiar_estado_rtp(db, rtp_obj, "validado-beneficiario")

    socketio.emit('rtp_validated_beneficiary', rtp_obj.to_dict(),
        room=f'psp_beneficiary_{rtp_obj.psp_beneficiary_id}')

    return result


def enrutar_rtp_service(rtp_id):
    rtp_obj = RTP.query.get(rtp_id)
    if not rtp_obj:
        return {"error": "RTP no encontrado"}
    result = cambiar_estado_rtp(db, rtp_obj, "enrutado")

    socketio.emit('rtp_routed', rtp_obj.to_dict(),
        room=f'psp_payer_{rtp_obj.psp_payer_id}')

    return result
```

Listing 4: RF-02 Validar y enrutar RTP services.py

```python

# 2. Validaci n Beneficiario
@rtp_blueprint.route('/rtp/<int:rtp_id>/validate-beneficiary',
    methods=['POST'])
@role_required('psp_beneficiary')
def validar_beneficiario(rtp_id):
    result = validar_beneficiario_service(rtp_id)
    status = 200 if "error" not in result else 400
    return jsonify(result), status

# 3. Enrutar al PSP del pagador
@rtp_blueprint.route('/rtp/<int:rtp_id>/route', methods=['POST'])
@role_required('psp_beneficiary')
def enrutar_rtp(rtp_id):
    result = enrutar_rtp_service(rtp_id)
    status = 200 if "error" not in result else 400
    return jsonify(result), status
```

Listing 5: RF-02 Validar y enrutar RTP routes.py

```python

def validar_iban(iban: str) -> bool:
    """
    Valida un IBAN (ISO 13616) usando el algoritmo de
       mdulo    97 = 1    .

    Par metros
    ----------
    iban : str
        IBAN en cualquier formato (con o sin espacios).

```

```
    Retorna
    -------
    bool
        True si el IBAN es sint cticamente correcto, False en caso
            contrario.
    """
    if not iban:
        return False

    # 1) Limpiar y normalizar
    iban = iban.replace(" ", "").upper()
    if len(iban) < 15 or len(iban) > 34 or not iban.isalnum():
        return False

    # 2) Reorganizar: los cuatro primeros caracteres al final
    rearranged = iban[4:] + iban[:4]

    # 3) Convertir letras a n meros (A=10, B=11,    , Z=35)
    digits = []
    for ch in rearranged:
        if ch.isdigit():
            digits.append(ch)
        elif ch.isalpha():
            digits.append(str(10 + string.ascii_uppercase.index(ch)))
        else:
            return False
    numeric_iban = int("".join(digits))

    # 4) Comprobar resto
    return numeric_iban % 97 == 1
```

Listing 6: RF-02 Validar y enrutar RTP utils.py

```

// PSP Beneficiary: Validar RTP
const validateBeneficiaryForm =
    document.getElementById('validateBeneficiaryForm');
if (validateBeneficiaryForm) {
  validateBeneficiaryForm.addEventListener('submit', function(e) {
    e.preventDefault();
    const rtpId = document.getElementById('rtpIdValidateBene').value;
    const data = { actor_id: currentActorId };

    fetch(`http://127.0.0.1:5000/rtp/${rtpId}/validate-beneficiary`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(data)
    })
    .then(r => r.json())
    .then(result => {
      const respDiv =
          document.getElementById('validateBeneficiaryResponse');
      respDiv.classList.remove('invisible-section');
      respDiv.innerText = JSON.stringify(result, null, 2);
    })
    .catch(err => console.error(err));
  });
}
```

```
  // PSP Beneficiary: Enrutar RTP
  const routeForm = document.getElementById('routeForm');
  if (routeForm) {
    routeForm.addEventListener('submit', function(e) {
      e.preventDefault();
      const rtpId = document.getElementById('rtpIdRoute').value;
      const data = { actor_id: currentActorId };

      fetch(`http://127.0.0.1:5000/rtp/${rtpId}/route`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(data)
      })
      .then(r => r.json())
      .then(result => {
        const respDiv = document.getElementById('routeResponse');
        respDiv.classList.remove('invisible-section');
        respDiv.innerText = JSON.stringify(result, null, 2);
      })
      .catch(err => console.error(err));
    });
  }
```

Listing 7: RF-02 Validar y enrutar RTP RTP.js

```
def validar_payer_service(rtp_id):
    """
    Validaci n por el PSP del pagador.
    1) Si el payer no tiene saldo suficiente, forzamos 'rechazado' (o
       'cancelado')
    2) Si s  tiene saldo, seguimos con 'validado_payer'
    """
    rtp_obj = RTP.query.get(rtp_id)
    if not rtp_obj:
        return {"error": "RTP no encontrado"}

    # Busco el Actor que sea el payer:
    from models import Actor
    payer_actor = Actor.query.get(rtp_obj.payer_id)
    if not payer_actor:
        return {"error": "El payer asignado a este RTP no existe"}

    # Verifico fondos
    if payer_actor.balance < rtp_obj.amount:
        # Forzamos el rechazo
        return rechazar_rtp(db, rtp_obj, "Saldo insuficiente (PSP forz
            cancelaci n)")

    # Si hay saldo suficiente, marcamos 'validado_payer'
    result = cambiar_estado_rtp(db, rtp_obj, "validado_payer")

    socketio.emit('rtp_validated_payer', rtp_obj.to_dict(),
        room=f'payer_{rtp_obj.payer_id}')

    return result
```

Listing 8: RF-03 Validación pagador services.py

```
# 4. Validaci n del Payer
@rtp_blueprint.route('/rtp/<int:rtp_id>/validate-payer', methods=['POST'])
@role_required('psp_payer')
def validar_payer(rtp_id):
    result = validar_payer_service(rtp_id)
    status = 200 if "error" not in result else 400
    return jsonify(result), status
```

Listing 9: RF-03 Validación pagador routes.py

```

  // PSP Payer: Validar RTP
  const validatePayerForm = document.getElementById('validatePayerForm');
  if (validatePayerForm) {
    validatePayerForm.addEventListener('submit', function(e) {
      e.preventDefault();
      const rtpId = document.getElementById('rtpIdValidatePayer').value;
      const data = { actor_id: currentActorId };

      fetch(`http://127.0.0.1:5000/rtp/${rtpId}/validate-payer`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(data)
      })
      .then(r => r.json())
      .then(result => {
        const respDiv = document.getElementById('validatePayerResponse');
        respDiv.classList.remove('invisible-section');
        respDiv.innerText = JSON.stringify(result, null, 2);
      })
      .catch(err => console.error(err));
    });
  }
```

Listing 10: RF-03 Validación pagador RTP.js

```

def decision_payer_service(rtp_id, data):
    """
    Cuando el pagador da su decisi n final ('aceptado' o 'rechazado')
    1) Si 'aceptado', restamos el dinero de su cuenta.
    """
    rtp_obj = RTP.query.get(rtp_id)
    if not rtp_obj:
        return {"error": "RTP no encontrado"}

    decision = data.get('decision')
    if decision not in ["aceptado", "rechazado"]:
        return {"error": "Decisi n no v lida"}

    # Si es 'aceptado', restamos el dinero
    if decision == "aceptado":
```

```
        from models import Actor
        payer_actor = Actor.query.get(rtp_obj.payer_id)
        if not payer_actor:
            return {"error": "El payer asignado no existe"}

        if payer_actor.balance < rtp_obj.amount:
            return rechazar_rtp(db, rtp_obj, "Saldo insuficiente en el
                ltimo  momento")

        payer_actor.balance -= rtp_obj.amount
        db.session.commit()

    result = cambiar_estado_rtp(db, rtp_obj, decision)

    # Notificar al beneficiario que ya se tom  la decisi n en el RTP
    socketio.emit('rtp_decision', rtp_obj.to_dict(),
        room=f'beneficiary_{rtp_obj.beneficiary_id}')

    return result
```

Listing 11: RF-04 Decisión final services.py

```
# 5. Decisi n final
@rtp_blueprint.route('/rtp/<int:rtp_id>/decision', methods=['POST'])
@role_required('payer')
def decision_payer(rtp_id):
    data = request.get_json()
    result = decision_payer_service(rtp_id, data)
    status = 200 if "error" not in result else 400
    return jsonify(result), status
```

Listing 12: RF-04 Decisión final routes.py

```
  // Payer: Decidir RTP
  const decisionForm = document.getElementById('decisionForm');
  if (decisionForm) {
    decisionForm.addEventListener('submit', function(e) {
      e.preventDefault();
      const rtpId = document.getElementById('rtpIdDecision').value;
      const decisionValue = document.getElementById('decision').value;
      const data = {
        actor_id: currentActorId,
        decision: decisionValue
      };

      fetch(`http://127.0.0.1:5000/rtp/${rtpId}/decision`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(data)
      })
      .then(r => r.json())
      .then(result => {
        const respDiv = document.getElementById('decisionResponse');
        respDiv.classList.remove('invisible-section');
        respDiv.innerText = JSON.stringify(result, null, 2);
      })
      .catch(err => console.error(err));
```

```
    });
  }
```

Listing 13: RF-04 Decisión final RTP.js

```
    # ext_socketio.py
    from flask_socketio import SocketIO, join_room, emit

    socketio = SocketIO()
```

Listing 14: RF-05 Notificación ext_socketio.py

```

    # Notificar al beneficiario que ya se tom  la decisi n en el RTP
    socketio.emit('rtp_decision', rtp_obj.to_dict(),
        room=f'beneficiary_{rtp_obj.beneficiary_id}')
```

Listing 15: RF-05 Notificación services.py

```

  // Para psp_beneficiary cuando se crea un RTP
  socket.on('rtp_created', function(data) {
    if (currentActorRole === 'psp_beneficiary') {
      addNotificationToList(data);
    }
  });

  // Para psp_payer cuando se enruta el RTP
  socket.on('rtp_routed', function(data) {
    if (currentActorRole === 'psp_payer') {
      addNotificationToList(data);
    }
  });

  // Para payer cuando el PSP payer valida
  socket.on('rtp_validated_payer', function(data) {
    if (currentActorRole === 'payer') {
      addNotificationToList(data);
    }
  });

  // Para beneficiary cuando el pagador decide
  socket.on('rtp_decision', function(data) {
    if (currentActorRole === 'beneficiary') {
      addNotificationToList(data);
    }
  });
}
```

Listing 16: RF-05 Notificación RTP.js

```

def cambiar_estado_rtp(db, rtp_obj, new_status):
    old_status = rtp_obj.status
    rtp_obj.status = new_status
```

```
    db.session.commit()

    # Generar un hash
    hash_input = f"{rtp_obj.id}{rtp_obj.iban}{rtp_obj.amount}{old_status}{new_status}".encode('utf-8')
    hash_value = hashlib.sha256(hash_input).hexdigest()

    nuevo_log = Log(
        rtp_id=rtp_obj.id,
        old_status=old_status,
        new_status=new_status,
        hash_value=hash_value
    )
    db.session.add(nuevo_log)
    db.session.commit()

    return {
        "message": f"RTP {rtp_obj.id} actualizado de {old_status} a {new_status}"
    }
```

Listing 17: RNF-03 Seguridad y trazabilidad utils.py

```

class Log(db.Model):
    __tablename__ = 'log'
    id = db.Column(db.Integer, primary_key=True)
    rtp_id = db.Column(db.Integer, nullable=False)
    old_status = db.Column(db.String(20))
    new_status = db.Column(db.String(20))
    timestamp = db.Column(db.DateTime, default=db.func.current_timestamp())
    hash_value = db.Column(db.String(64))

    def to_dict(self):
        return {
            "id": self.id,
            "rtp_id": self.rtp_id,
            "old_status": self.old_status,
            "new_status": self.new_status,
            "timestamp": self.timestamp.isoformat(),
            "hash_value": self.hash_value
        }
```

Listing 18: RNF-03 Seguridad y trazabilidad models.py

```
import os

basedir = os.path.abspath(os.path.dirname(__file__))

class Config:
    SQLALCHEMY_DATABASE_URI = 'sqlite:///' + os.path.join(basedir, 'rtp.db')
    SQLALCHEMY_TRACK_MODIFICATIONS = False
```

Listing 19: RNF-04 Portabilidad config.py

```
Flask==2.2.2
Flask-SQLAlchemy==3.0.2
```

Listing 20: RNF-04 Portabilidad requirements.txt

```
from flask import Flask, send_from_directory
from ext_socketio import socketio, join_room
from config import Config
from models import db, Actor
import os
import webbrowser
from routes import rtp_blueprint

STATIC_FOLDER = os.path.join(os.path.dirname(__file__), '../frontend')
app = Flask(__name__, static_folder=STATIC_FOLDER)
app.config.from_object(Config)
db.init_app(app)

# Inicializaci n de SocketIO
socketio.init_app(app, cors_allowed_origins="*")

with app.app_context():
    db.drop_all()
    db.create_all()

    # Crear los 4 usuarios predefinidos:
    # 1) "Mercadona" => beneficiary
    # 2) "PSPMercadona" => psp_beneficiary
    # 3) "PSPalonso" => psp_payer
    # 4) "alonso" => payer
    # Contrase as ficticias, p.ej. "1234"
    mercadona = Actor(
        username="e",
        password="1",
        name="Empresa",
        role="beneficiary",
        iban="ES2600123456789012345 6",
        balance=0.0
    )
    psp_merc = Actor(
        username="pm",
        password="1",
        name="PSP de empresa",
        role="psp_beneficiary"
    )
    psp_alonso = Actor(
        username="pa",
        password="1",
        name="PSP de Alonso",
        role="psp_payer"
    )
    alonso = Actor(
        username="a",
        password="1",
        name="Alonso",
        role="payer",
        iban="ES4509876543210000000 0",
        balance=1000.0
```

```
    )
```

Listing 21: RNF-04 Portabilidad app.py

```
class Actor(db.Model):
    __tablename__ = 'actor'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True)
    password = db.Column(db.String(50))
    name = db.Column(db.String(50), nullable=False)
    role = db.Column(db.String(20), nullable=False)

    photo_url = db.Column(db.String(200), nullable=True)  # URL o base64
    iban = db.Column(db.String(34), nullable=True)
    balance = db.Column(db.Float, default=0.0)

    # Campo nuevo: psp_id, que referencia a otro Actor que es PSP
    psp_id = db.Column(db.Integer, db.ForeignKey('actor.id'), nullable=True)
    # Relaci n para que se pueda acceder con actor.psp
    psp = db.relationship('Actor', remote_side=[id])

    def to_dict(self):
        return {
            "id": self.id,
            "username": self.username,
            "name": self.name,
            "role": self.role,
            "photo_url": self.photo_url,
            "iban": self.iban,
            "balance": self.balance,
            "psp_id": self.psp_id  # Podr amos incluir m s info del psp,
                etc.
        }
```

Listing 22: clase Actor

```
class RTP(db.Model):
    __tablename__ = 'rtp'
    id = db.Column(db.Integer, primary_key=True)
    iban = db.Column(db.String(34), nullable=False)
    amount = db.Column(db.Float, nullable=False)
    status = db.Column(db.String(20), default="creado")
    timestamp = db.Column(db.DateTime, default=db.func.current_timestamp())
    beneficiary_id = db.Column(db.Integer)
    psp_beneficiary_id = db.Column(db.Integer)
    psp_payer_id = db.Column(db.Integer)
    payer_id = db.Column(db.Integer)

    def to_dict(self):
        return {
            "id": self.id,
            "iban": self.iban,
            "amount": self.amount,
            "status": self.status,
            "timestamp": self.timestamp.isoformat(),
            "beneficiary_id": self.beneficiary_id,
            "psp_beneficiary_id": self.psp_beneficiary_id,
```

```
            "psp_payer_id": self.psp_payer_id,
            "payer_id": self.payer_id
        }
```

Listing 23: clase RTP

```
class Log(db.Model):
    __tablename__ = 'log'
    id = db.Column(db.Integer, primary_key=True)
    rtp_id = db.Column(db.Integer, nullable=False)
    old_status = db.Column(db.String(20))
    new_status = db.Column(db.String(20))
    timestamp = db.Column(db.DateTime, default=db.func.current_timestamp())
    hash_value = db.Column(db.String(64))

    def to_dict(self):
        return {
            "id": self.id,
            "rtp_id": self.rtp_id,
            "old_status": self.old_status,
            "new_status": self.new_status,
            "timestamp": self.timestamp.isoformat(),
            "hash_value": self.hash_value
        }
```

Listing 24: clase Log

Índice de Figuras

Índice de Tablas

Índice de Listados

Bibliografía

[Ban18] European Central Bank. *TARGET Instant Payment Settlement (TIPS)*. Online. Consultado el 23-06-2025. 2018. URL: https://www.ecb.europa.eu/paym/target/tips/html/index.en.html.

[Ban20] European Central Bank. *Euro Retail Payments Board (ERPB) – SEPA Request-to-Pay status update*. PDF. Consultado el 23-06-2025. 2020. URL: https://www.ecb.europa.eu/paym/groups/erpb/shared/pdf/14th-ERPB-meeting/Status_update_on_the_SEPA_request-to-pay_scheme.pdf.

[Ban23] European Central Bank. *Single Euro Payments Area (SEPA) – Overview*. Online. 2023. URL: https://www.ecb.europa.eu/paym/integration/retail/sepa/html/index.en.html.

[Cle17] EBA Clearing. *RT1 – Pan-European Real-Time Gross Settlement System*. Online. Consultado el 23-06-2025. 2017. URL: https://www.ebaclearing.eu/services-instant-payments/rt1/.

[Com01] European Commission. *Commission Recommendation 2001/609/EC on the use of the IBAN*. Online. Consultado el 23-06-2025. 2001. URL: https://eur-lex.europa.eu/eli/reco/2001/609/oj.

[con25] Wikipedia contributors. *Four Corners Model for Payment Security*. Online. Consultado el 23-06-2025. 2025. URL: https://en.wikipedia.org/wiki/Four_Corners_Model_for_Payment_Security.

[Cou21] European Payments Council. *EDS – EPC Directory Service Rulebook v1.0*. PDF. Consultado el 23-06-2025. 2021. URL: https://www.europeanpaymentscouncil.eu/document-library/rulebooks/epc098-21-eds-directory-service-rulebook-version-10.

[Cou23a] European Payments Council. *EPC137 - SRTP Implementation Guidelines*. 2023. URL: https://www.europeanpaymentscouncil.eu/document-library/implementation-guidelines/sepa-request-pay-implementation-guidelines-version-30.

[Cou23b] European Payments Council. *EPC164 - API Security Framework*. 2023. URL: https://www.europeanpaymentscouncil.eu/sites/default/files/kb/file/2024-10/EPC164-22%20v2.0%20API%20Security%20Framework.pdf.

[Cou23c] European Payments Council. *SEPA Credit Transfer Scheme Rulebook v1.1*. PDF. Consultado el 23-06-2025. 2023. URL: https://www.europeanpaymentscouncil.eu/sites/default/files/kb/file/2023-11/EPC125-05%202023%20SCT%20Rulebook%20version%201.1_1.pdf.

[Cou24a] European Payments Council. *2025 SEPA Direct Debit Core Rulebook v1.0*. PDF. 2024. URL: https://www.europeanpaymentscouncil.eu/document-library/rulebooks/2025-sepa-direct-debit-core-rulebook-version-10.

[Cou24b] European Payments Council. *2025 SEPA Instant Credit Transfer Rulebook v1.0*. PDF. 2024. URL: https://www.europeanpaymentscouncil.eu/document-library/rulebooks/2025-sepa-instant-credit-transfer-rulebook-version-10.

[Cou25a] European Payments Council. *European Payments Council (EPC) – Official Website*. Online. Consultado el 23-06-2025. 2025. URL: https://www.europeanpaymentscouncil.eu/.

[Cou25b] European Payments Council. *SEPA Direct Debit Core Scheme Rulebook v1.0*. PDF. 2025. URL: https://www.europeanpaymentscouncil.eu/sites/default/files/kb/file/2024-11/EPC016-06%202025%20SDD%20Core%20Rulebook%20version%201.0.pdf.

[Cou25c] European Payments Council. *SRTP Scheme Rulebook v4.0*. 2025. URL: https://www.europeanpaymentscouncil.eu/sites/default/files/kb/file/2024-11/EPC014-20%20v4.0%20SEPA%20RTP%20Scheme%20Rulebook.pdf.

[Eur18] European Commission. *Commission Delegated Regulation (EU) 2018/389 of 27 November 2017 supplementing Directive (EU) 2015/2366 with regard to regulatory technical standards for strong customer authentication and common and secure open standards of communication*. Online. 2018. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32018R0389.

[Eur12] European Parliament and Council. *Regulation (EU) No 260/2012 of 14 March 2012 on the establishment of technical and business requirements for credit transfers and direct debits in euros*. Online. Consultado el 23-06-2025. 2012. URL: https://eur-lex.europa.eu/eli/reg/2012/260/oj.

[NV24] Adyen N.V. *SEPA chargebacks – guidelines*. Online. 2024. URL: https://docs.adyen.com/risk-management/chargeback-guidelines/sepa-chargebacks/.

[PC12] European Parliament y Council. *Regulation (EU) No 260/2012 on credit transfers and direct debits*. Online. 2012. URL: https://eur-lex.europa.eu/eli/reg/2012/260/oj.

[Res10] Steinbeis University Berlin Research Center for Financial Services. *SEPA Direct Debit – a success story for the European payment market*. PDF. 2010. URL: https://www.steinbeis-research.de/pdf/Study_SEPA_Direct_Debit.pdf.

[Sta13] International Organization for Standardization. *ISO 20022 Financial services – Universal financial industry message scheme*. Online. 2013. URL: https://www.iso.org/standard/55005.html.