



Fundamentos de Ingeniería del Software para Sistemas Cloud

Proyecto de la asignatura: Microservicios de Notificaciones y Pagos Programados - BankUS

Máster Universitario en Ingeniería del Software -
Cloud, Datos y Gestión de las Tecnología de la
Información

Autores:

Berdugo Rodríguez, Elena

Costela Guijosa, Jose Luis

Tabla de contenidos

Tabla de contenidos	2
1. Introducción	3
2. Nivel de acabado	3
Nivel de acabado de la aplicación	3
Nivel de acabado de los microservicios de pareja	4
Análisis justificativo de la suscripción óptima de las APIs del proyecto	8
Notificaciones por email: Twilio SendGrid Email API	9
Sincronización temporal: NTP (pool.ntp.org)	9
Regla de decisión (estimación simple de consumo)	10
Conclusión (suscripción óptima)	10
3. Descripción de la aplicación	11
4. Descomposición y arquitectura	11
5. Customer Agreement	13
6. Plan de precios	14
7. Descripción API REST	15
Pagos programados	15
Notificaciones	16
8. Justificación de requisitos y evidencias	17
Trabajo realizado por la pareja	17
Microservicio Notificaciones	17
Microservicio Scheduled Payments	18
Aplicación basada en microservicios	18
9. Análisis de esfuerzos	19
Jose Luis Costela Guijosa	19
Elena Berdugo Rodríguez	19
10. Uso de Inteligencia Artificial	20

1. Introducción

Para la asignatura Fundamentos de Ingeniería del Software para Sistemas Cloud, del Máster Universitario en Ingeniería del Software – Cloud, Datos y Gestión de las Tecnologías de la Información, se ha desarrollado una aplicación bancaria siguiendo una arquitectura basada en microservicios. El trabajo se ha centrado en aplicar los conceptos vistos en la asignatura a un caso práctico realista, poniendo especial atención en el diseño del sistema, la separación de responsabilidades entre servicios y su correcta integración en un entorno cloud.

En este documento se describe el desarrollo de dos microservicios de dicha aplicación. Por un lado, el microservicio de Pagos Programados responsable de gestionar la creación, consulta y ejecución de pagos en fechas futuras. Por otro, el microservicio de Notificaciones que se encarga de informar a los usuarios sobre distintos eventos relevantes del sistema. Ambos servicios se han diseñado para trabajar de forma independiente pero coordinada, siguiendo buenas prácticas de ingeniería del software y utilizando APIs bien definidas para facilitar su comunicación y mantenimiento.

2. Nivel de acabado

Nivel de acabado de la aplicación

La entrega se plantea con todos los microservicios completamente funcionales integrados en la arquitectura global (API Gateway + microservicios), acompañado de un frontend común desarrollado en React que permite operar de forma íntegra todas las funcionalidades expuestas por cada microservicio, como la visualización e interacción con los dominios del sistema (Accounts, Transactions, Payments, etc.).

El proyecto se presenta con la arquitectura base y los microservicios totalmente operativos:

- API Gateway: Implementado y funcional. Incluye lógica de throttling (limitación de peticiones por subscripción), autenticación centralizada y enrutamiento al resto de microservicios.
- Microservicio Accounts: Funcionalidad completa CRUD, gestión de estados (bloqueo/desbloqueo) y endpoints dependientes de otros microservicios (Cards/Currencies).
- Microservicio Currencies: Actúa como adaptador/wrapper consumiendo la API externa para realizar conversiones reales, además de controlar el consumo de la API externa.

- **Mircroservcio Cards:** Funcionalidad completa CRUD, gestión de estados (active/frozen)
- **Mircroservcio Transfers:** Funcionalidad completa, gestiona las transferencias realizadas en la aplicación, ofreciendo recursos a otros microservicios como el de pagos programados y al mismo tiempo dependiente de otros como el microservicio de cuentas. Se permiten revertir las transferencias y se enfoca en la trazabilidad de operaciones independientemente de su estado.
- **Microservicio User Auth:** Alta/edición de usuarios, login con CAPTCHA, emisión de JWT con revocación (lista negra) y validación centralizada de tokens para el resto de microservicios.
- **Microservicio Anti-fraud:** Reglas de detección de fraude, creación/gestión de alertas y bloqueo proactivo de cuentas mediante circuit breaker hacia Accounts.
- **Microservicio Scheduled Payments:** Funcionalidad completa, realiza transferencias en momentos programados utilizando una sincronización por NTP y algunas limitaciones tipo Rate Limit para asegurar un correcto funcionamiento.
- **Microservicio Notifications:** Funcionalidad completa, se encarga de informar a los usuarios sobre distintos eventos relevantes del sistema utilizando SendGrid enviando mails a los distintos usuarios.
- **Microservicio Bank Statements:** Gestión completa de estados de cuenta bancarios con generación automática mensual mediante cron job, generación de estado de cuneta del mes actual consumiendo transacciones externas; muestra al cliente un balance de sus gastos.

Nivel de acabado de los microservicios de pareja

Acabado: Nivel hasta 9 puntos, se incluye:

- Todo lo indicado en el nivel hasta 5 puntos
- Aplicación basada en microservicios básica implementada
- Análisis justificativo de la suscripción óptima de las APIs del proyecto → Descrito en este mismo apartado de este documento.
- Un mínimo de 20 pruebas de componente implementadas incluyendo escenarios positivos y negativos → Más de 20 teniendo en cuenta los dos microservicios (notificaciones y pagos programados).
- Tener el API REST documentado con swagger (OpenAPI).
 - Pagos programados: [Enlace a JSON de Swagger](#)
 - Notificaciones: [Enlace a JSON de Swagger](#)

- 5 características del microservicio avanzado implementados → Indicados a continuación.
- 4 características de la aplicación basada en microservicios avanzada implementados → Indicados a continuación.
- **MICROSERVICIO BÁSICO QUE GESTIONE UN RECURSO (Completo)**
 - El backend debe ser una API REST tal como se ha visto en clase implementando al menos los métodos GET, POST, PUT y DELETE y devolviendo un conjunto de códigos de estado adecuado.
 - Pagos programados: [Enlace a código](#). También documentado con Swagger.
 - Notificaciones: [Enlace a código](#). También documentado con Swagger.
 - La API debe tener un mecanismo de autenticación. → Se realizan en la Api Gateway. [Enlace a código](#).
 - Debe tener un frontend que permita hacer todas las operaciones de la API.
→ Existe un frontend común donde hay varias pestañas, una común y otras individuales por microservicio. [Enlace a código](#).
 - Debe estar desplegado y ser accesible desde la nube (ya sea de forma individual o como parte de la aplicación). → [Enlace a frontend en la nube](#).
 - La API que gestione el recurso también debe ser accesible en una dirección bien versionada. → La API Gateway redirige cada solicitud a los distintos microservicios con las siguientes rutas:
 - Pagos programados:
`https://<API_GATEWAY>/v1/scheduled-payments`
 - Notificaciones: `https://<API_GATEWAY>/v1/notifications`
 - Se debe tener una documentación de todas las operaciones de la API incluyendo las posibles peticiones y las respuestas recibidas → Documentación en Swagger adjunta al proyecto.
 - Pagos programados: [Enlace a JSON de Swagger](#)
 - Notificaciones: [Enlace a JSON de Swagger](#)
 - Debe tener persistencia utilizando MongoDB u otra base de datos no SQL.
→ Uso de MongoDB para ambos microservicios
 - Deben validarse los datos antes de almacenarlos en la base de datos (por ejemplo, haciendo uso de mongoose). → Uso de Pydantic en ambos microservicios:
 - Pagos programados: [Enlace a código](#).

- Notificaciones: [Enlace a código.](#)
 - Debe haber definida una imagen Docker del proyecto.
 - Pagos programados:
 - [Enlace a compose.](#)
 - [Enlace a DockerHub.](#)
 - Notificaciones:
 - [Enlace a compose.](#)
 - [Enlace a DockerHub.](#)
 - Gestión del código fuente: El código debe estar subido a un repositorio de Github siguiendo Github Flow
 - Pagos programados: [Enlace a Network graph.](#)
 - Notificaciones: [Enlace a Network graph.](#)
 - Integración continua: El código debe compilarse, probarse y generar la imagen de Docker automáticamente usando GitHub Actions u otro sistema de integración continua en cada commit
 - Pagos programados:
 - [Enlace a Github Actions.](#)
 - [Enlace a código.](#)
 - Notificaciones:
 - [Enlace a Github Actions.](#)
 - [Enlace a código.](#)
 - Debe haber pruebas de componente implementadas en Javascript para el código del backend utilizando Jest o similar. Como norma general debe haber tests para todas las funciones del API no triviales de la aplicación. Probando tanto escenarios positivos como negativos. Las pruebas deben ser tanto in-process como out-of-process.
 - Pagos programados: [Enlace a código.](#)
 - Notificaciones: [Enlace a código.](#) [Enlace a código 2](#)
- **MICROSERVICIO AVANZADO QUE GESTIONE UN RECURSO (5 características)**
 - Implementar un frontend con rutas y navegación.
 - Frontend común: [Enlace a código.](#)
 - Frontend pagos programados: [Enlace a código.](#)
 - Frontend notificaciones: [Enlace a código.](#)
 - Consumir alguna API externa (distinta de las de los grupos de práctica) a través del backend o algún otro tipo de almacenamiento de datos en cloud como Amazon S3.
 - Pagos programados:

- Uso de api NTP “pool.ntp.org” para sincronización del microservicio en tiempo real.
- [Enlace a código.](#)
- Notificaciones:
 - Uso de SendGrid para enviar los mails y recoger el flujo de actividad.
 - [Enlace a logs exportados de SendGrid](#)
- Implementar el patrón “rate limit” al hacer uso de servicios externos
 - Pagos programados:
 - Configurado de la siguiente manera:
 - POST /v1/scheduled-payments/: Límite a 5/min
 - GET /v1/scheduled-payments/accounts/<account_id>: Límite a 60/min
 - GET /v1/scheduled-payments/accounts/<account_id>/upcoming: Límite a 30/min
 - DELETE /v1/scheduled-payments/<id>: Límite a 20/min
 - [Enlace a código.](#)
- Implementar mecanismos de gestión de la capacidad como throttling o feature toggles para rendimiento. → Mecanismo de Throttling en API Gateway
- Cualquier otra extensión al microservicio básico acordada previamente con el profesor. → Como se acordó en uno de los seguimientos con el profesor, el microservicio de pagos programados se consideraría una extensión del microservicio de Transacciones.
- **APLICACIÓN BASADA EN MICROSERVICIOS BÁSICA (Completo)**
 - Interacción completa entre todos los microservicios de la aplicación integrando información. La integración debe realizarse a través del backend. → Realizado.
 - Tener un frontend común que integre los frontends de cada uno de los microservicios. Cada pareja debe ocuparse, al menos, de la parte específica de su microservicio en el frontend común. → Realizado.
 - Permitir la suscripción del usuario a un plan de precios y adaptar automáticamente la funcionalidad de la aplicación según el plan de precios seleccionado. → Existen 3 planes de precios donde cada microservicio se adapta a cada plan:
 - Pagos programados:

- Plan Básico: Posibilidad de un solo pago programado
- Plan Premium: Hasta 10 pagos programados posibles
- Plan Pro: Pagos programados ilimitados
- Notificaciones:
 - Plan Básico: Notificaciones sobre las transacciones y los accesos en tiempo real
 - Plan Premium: Notificaciones de transacciones, accesos y posibles fraudes en tiempo real
 - Plan Pro: Notificaciones de transacciones, accesos, posibles fraudes e historial en tiempo real
- **APLICACIÓN BASADA EN MICROSERVICIOS AVANZADA (4 características)**
 - Implementar un mecanismo de autenticación basado en JWT o equivalente. → Como se acordó en el último seguimiento con el profesor, al ser realizado por todas las parejas, esta característica se considera de APLICACIÓN BASADA EN MICROSERVICIOS AVANZADA. Implementado en el Microservicio User Auth (login/validación JWT) y gestionado desde el API Gateway.
 - Incluir en el plan de precios límites de uso y aplicarlos automáticamente según la suscripción del usuario. → Comentado en la sección anterior.
 - Hacer uso de un API Gateway con funcionalidad avanzada como un mecanismo de throttling o de autenticación. → Desarrollo de una API Gateway. [Enlace a código.](#)
 - Cualquier otra extensión a la aplicación basada en microservicios básica acordada previamente con el profesor → Sistema de logs comunes con Grafana.

Análisis justificativo de la suscripción óptima de las APIs del proyecto

La suscripción óptima se define como aquella que minimiza el coste económico manteniendo un margen de seguridad suficiente para la carga prevista durante la demostración académica y un escenario realista de crecimiento, evitando tanto la infrautilización como el sobredimensionamiento de los servicios externos consumidos.

En el caso de los microservicios Notificaciones y Pagos Programados, el uso de APIs externas se basa en el envío de notificaciones por correo electrónico y en la sincronización temporal precisa necesaria para la ejecución de pagos programados.

Notificaciones por email: Twilio SendGrid Email API

El microservicio Notificaciones se ha encargado de implementar Twilio SendGrid Email API para el envío de correos electrónicos asociados a eventos relevantes del sistema, como accesos de usuario, transacciones, detección de fraude o consultas de historial, adaptando el tipo de notificación al evento relacionado y limitando el envío en función del plan de suscripción del usuario (Básico, Premium o Pro).

Durante el contexto de evaluación académica y demostración funcional, el volumen de correos enviados es limitado y está directamente ligado a eventos discretos del sistema, no a flujos continuos de alta frecuencia. En este escenario, el plan gratuito / trial de SendGrid, que permite el envío de hasta 100 correos diarios, resulta suficiente para cubrir holgadamente las necesidades del proyecto sin incurrir en costes adicionales. Adicionalmente, el microservicio de notificaciones ha sido diseñado para desacoplar la persistencia y visualización de las notificaciones del envío efectivo de correos electrónicos, de modo que, en caso de que se alcance el límite diario del proveedor externo o se produzca un fallo en el envío, las notificaciones siguen siendo almacenadas y mostradas correctamente en el frontend. Este enfoque permite mantener la funcionalidad principal del sistema incluso ante limitaciones temporales del servicio externo.

Sin embargo, en un escenario de explotación real con un mayor número de usuarios activos y una mayor frecuencia de eventos notificables, la suscripción óptima evolucionaría hacia el plan SendGrid Essentials (a partir de 19,95 €/mes), que ofrece un notable salto en el volumen de envíos mensuales (50.000 correos/día), además de garantías de continuidad y soporte. No obstante, dicho salto solo estaría justificado ante un crecimiento sostenido del sistema.

Sincronización temporal: NTP (pool.ntp.org)

El microservicio Pagos Programados requiere una sincronización temporal precisa para garantizar la correcta ejecución de pagos en fechas y horas programadas. Para ello, se utiliza el servicio público NTP (pool.ntp.org), ampliamente empleado y recomendado para este tipo de necesidades.

El consumo de la API NTP es puntual y de baja frecuencia, limitándose a sincronizaciones periódicas del reloj del microservicio y respetando las buenas prácticas de uso del pool público. Dado este patrón de consumo, no es necesario contratar un servicio NTP de pago, ya que el uso de pool.ntp.org proporciona una precisión suficiente para el contexto del proyecto sin coste económico.

En un entorno de producción a gran escala o con requisitos estrictos de alta disponibilidad, auditoría o acuerdos de nivel de servicio (SLA), podría valorarse el uso de un proveedor NTP comercial. Sin embargo, para los objetivos del proyecto, la opción pública constituye la suscripción óptima.

Regla de decisión (estimación simple de consumo)

Para justificar cuantitativamente la elección de las suscripciones:

- Si E representa el número medio de eventos diarios que generan notificaciones por correo y D el número de días de actividad mensual, el consumo aproximado de emails se puede estimar como:

$$\text{Emails/mes} \approx E \cdot D$$

Mientras $E \leq 100$, el plan gratuito de SendGrid cubre con margen suficiente el uso previsto durante la evaluación.

- En el caso de NTP, dado que la sincronización se realiza de forma periódica y no por operación individual, el volumen de consultas se mantiene muy por debajo de los límites recomendados del servicio público, justificando su uso sin necesidad de suscripción de pago.

Conclusión (suscripción óptima)

En consecuencia, para los microservicios desarrollados y en el contexto de una entrega académica, la suscripción óptima queda justificada como:

- Twilio SendGrid (Free / Trial) para el envío de notificaciones por correo electrónico desde el microservicio Notificaciones, con coste cero y volumen suficiente para la carga prevista.
- NTP público (pool.ntp.org) para la sincronización temporal del microservicio Pagos Programados, sin coste y con precisión adecuada.

En un escenario de crecimiento sostenido o explotación real del sistema, la migración natural sería hacia:

- SendGrid Essentials para garantizar continuidad del servicio de correo ante un aumento significativo de usuarios y eventos.
- Servicios NTP comerciales únicamente en caso de requerir SLA, alta disponibilidad o auditoría temporal avanzada.

3. Descripción de la aplicación

El sistema consiste en una arquitectura de microservicios para una entidad bancaria (BancUS). Permite la gestión integral de cuentas bancarias, incluyendo la creación de usuarios, consultas de saldo, creación de tarjetas, transacciones, transacciones con tarjetas, actualizaciones de estado, operaciones monetarias multidivisa y, en función del plan elegido, notificaciones, pagos programados y servicio antifraude.

El diseño separa la lógica de negocio principal (cuentas y operaciones) de servicios auxiliares (como la conversión de divisas) y del punto único de entrada (API Gateway). Esta descomposición facilita escalado independiente, tolerancia a fallos y evolución modular por equipos.

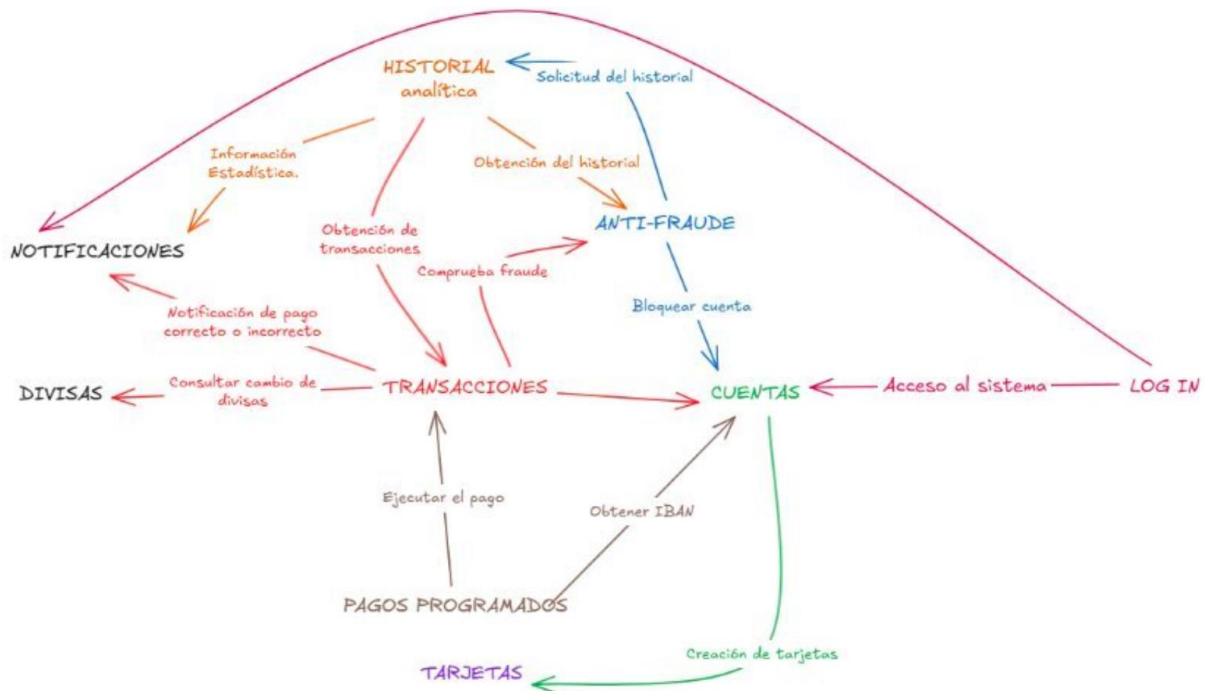
4. Descomposición y arquitectura

El sistema se compone de los siguientes elementos. Se marcan en **negrita** los desarrollados o en los que ha colaborado esta parte del equipo:

- **API Gateway:**
 - Punto único de entrada. Protege la red interna y distribuye las peticiones.
- Microservicio Accounts (Python/Quart):
 - Encargado de la persistencia y lógica de las cuentas.
 - Maneja la validación estricta de datos (Pydantic).
 - Orquesta llamadas a *Cards* y *Currencies*.
- Microservicio Currencies (Java/Spring Boot):
 - Provee servicios de conversión de moneda.
 - Integra proveedores externos (RapidAPI).
- Microservicio Cards (Node.js/Express)
 - Gestión del recurso tarjeta (CRUD), estados (*active/frozen*) y operaciones asociadas a tarjetas.
 - Recibe peticiones de *Accounts*, *Transfers* y *anti-fraud*
- Microservicio Transfers (Python/Quart):
 - Gestiona la funcionalidad de las transferencias.
 - Maneja la validación estricta de datos (Pydantic).
 - Realiza peticiones a una API Externa (TimeAPI) y al microservicio de cuentas para su funcionamiento base. (También llama a otros microservicios como el de autenticación o notificaciones).
- Microservicio User Auth (NestJS/Express):
 - Gestiona altas/bajas/edición de usuarios (MongoDB).

- Autentica con CAPTCHA, genera tokens JWT, valida/revoca tokens (lista negra) y notifica inicio de sesión.
- **Microservicio Anti-Fraud (NestJS/Express):**
 - Reglas de detección de riesgo sobre transacciones (importe, histórico, destinos).
 - Gestiona alertas de fraude (crear, listar, actualizar, eliminar) y bloquea cuentas vía Accounts con circuit breaker y timeout configurables.
- **Microservicio Scheduled Payments (Python/Quart):**
 - Realiza transferencias en momentos programados utilizando una sincronización por NTP y algunas limitaciones tipo Rate Limit para asegurar un correcto funcionamiento.
 - Servicio para crear, consultar, actualizar y eliminar pagos programados con planificación ONCE / WEEKLY / MONTHLY.
- **Microservicio Notifications (Python/Quart):**
 - Se encarga de informar a los usuarios sobre distintos eventos relevantes del sistema utilizando SendGrid enviando mails a los distintos usuarios.
- **Microservicio Bank Statements (Node.js/Express):**
 - Gestión de estados de cuenta bancarios (CRUD).
 - Generación automática mensual con cron job (día 1 de cada mes las 00:01).
 - Generación de estado decuenta del mes actual consumiendo microservicio de transacciones.
 - Visualización gráfica de balances.
- **Frontend común (React/Vite):**
 - Interfaz unificada con rutas y navegación. Incluye páginas específicas para cada microservicio

A continuación, se muestra un esquema de la conexión de los distintos microservicios:



5. Customer Agreement

Formato de respuestas

- Respuestas exitosas y de error en JSON.
- Errores consistentes y trazables (código, mensaje y, cuando proceda, detalle).

Semántica HTTP

- 200 para lecturas y actualizaciones con contenido.
- 201 para creación.
- 204 para operaciones sin contenido (por ejemplo, freeze/unfreeze si se decide sin body).
- 400 para validación y datos mal formados.
- 401 para no autenticado.
- 403 para autenticado sin permisos o por restricción de plan.
- 404 para recurso inexistente.
- 409 para conflictos de estado (por ejemplo, congelar una tarjeta ya congelada).
- 503 para dependencia no disponible cuando una operación requiere un servicio aguas abajo.

Disponibilidad y degradación

- Ante caída de dependencias, se devuelve error controlado con 503 y no se bloquea el servicio.
- Se recomiendan timeouts y reintentos limitados para evitar fallo en cascada.

6. Plan de precios

Los planes definen límites y activación de capacidades. A continuación, se muestran los 3 distintos planes configurados en la aplicación, marcando en negrita las limitaciones que afectan a los microservicios de Notificaciones y Pagos programados:

Plan	Precio	Límites y características funcionales
Básico	0 EUR/mes	<ul style="list-style-type: none"> • Cuenta de pruebas • 1 tarjeta virtual • Hasta 5 transacciones al mes • Notificaciones sobre las transacciones y los accesos en tiempo real • Posibilidad de un pago programado configurado
Premium	4,99 EUR/mes	<ul style="list-style-type: none"> • Hasta 5 tarjetas virtuales • Notificaciones de transacciones, accesos y posibles fraudes en tiempo real • Condiciones específicas para universitarios • Hasta 10 pagos programados posibles
Pro	9,99 EUR/mes	<ul style="list-style-type: none"> • Tarjetas virtuales ilimitadas • Transacciones ilimitadas • Notificaciones de transacciones, accesos, posibles fraudes e historial en tiempo real • Acceso avanzado a la API • Pagos programados ilimitados

7. Descripción API REST

Pagos programados

Desarrollado con `Quart` y `Quart-Schema` para soporte asíncrono y documentación automática.

Prefijo: /v1/scheduled-payments

Método	Endpoint	Descripción	Códigos de Respuesta
GET	/health	Healthcheck del servicio	<ul style="list-style-type: none"> 200
POST	/	Crea un pago programado (requiere Authorization; valida cuenta; aplica límites por suscripción).	<ul style="list-style-type: none"> 201 401 (sin token) 403 (límite plan) 404 (cuenta no existe) 409 (id duplicado) 503
GET	/accounts/{account_id}	Lista los pagos programados asociados a una cuenta (puede devolver lista vacía).	<ul style="list-style-type: none"> 200
GET	/accounts/{account_id}/upcoming	Devuelve los próximos pagos estimados (solo activos), ordenados por fecha más próxima. Query: limit (1..100, por defecto 10).	<ul style="list-style-type: none"> 200 400
GET	/scheduled_payment_id	Obtiene un pago programado por su id.	<ul style="list-style-type: none"> 200 404
PATCH	/scheduled_payment_id	Actualiza parcialmente un pago programado (PATCH; cambios parciales).	<ul style="list-style-type: none"> 200 404

DELETE	/scheduled_payment_id}	Elimina un pago programado.	<ul style="list-style-type: none"> • 200 • 404
--------	------------------------	-----------------------------	--

Notificaciones

Desarrollado con `Quart` y `Quart-Schema` para soporte asíncrono y documentación automática.

Prefijo: /v1/notifications

Método	Endpoint	Descripción	Códigos de Respuesta
GET	/health	Healthcheck del microservicio. Pensado para liveness/readiness en entornos Docker/Kubernetes.	<ul style="list-style-type: none"> • 200
POST	/events	Recibe eventos procedentes de otros microservicios (login, pagos, fraude, etc.), genera y persiste notificaciones y, según el tipo de evento, puede desencadenar el envío de correos electrónicos mediante SendGrid.	<ul style="list-style-type: none"> • 201
GET	/user/{userId}	Devuelve todas las notificaciones asociadas a un usuario concreto, ordenadas por fecha de creación (de la más reciente a la más antigua).	<ul style="list-style-type: none"> • 200
GET	/	Devuelve todas las notificaciones del sistema. Endpoint de administración para tareas de monitorización o debugging interno.	<ul style="list-style-type: none"> • 200
PUT	/notification_id}	Actualiza los campos de una notificación existente. Solo se modifican los campos presentes en el payload.	<ul style="list-style-type: none"> • 200 • 404 (no existe)
DELETE	/notification_id}	Elimina una notificación existente por su identificador.	<ul style="list-style-type: none"> • 200

			<ul style="list-style-type: none"> • 404 (no existe)
POST	/test-email	Endpoint de prueba para verificar la correcta integración con SendGrid mediante el envío de un correo electrónico de prueba.	<ul style="list-style-type: none"> • 200

8. Justificación de requisitos y evidencias

A continuación, aunque ya se encuentre definido en el punto 2 de esta memoria, describiremos con mayor detalle cómo los microservicios de Notificaciones y Pagos Programados cumplen los principales requisitos técnicos exigidos en la asignatura.

Trabajo realizado por la pareja

Microservicio Notificaciones

El microservicio de Notificaciones implementa una API REST completa que gestiona la creación, consulta, actualización y eliminación de notificaciones, cumpliendo con los métodos GET, POST, PUT y DELETE. La API se encuentra correctamente versionada y accesible a través del API Gateway bajo la ruta `/v1/notifications`.

La persistencia de datos se realiza mediante MongoDB, utilizando modelos definidos con Pydantic para validar los datos antes de su almacenamiento. El microservicio está documentado mediante Swagger/OpenAPI, generado automáticamente a partir del código, garantizando que la documentación se mantenga sincronizada con la implementación.

Como consumo de API externa, el microservicio integra Twilio SendGrid para el envío de correos electrónicos asociados a eventos relevantes del sistema. El envío de emails se encuentra desacoplado de la persistencia y visualización de notificaciones, permitiendo que, ante fallos o límites del proveedor externo, las notificaciones sigan mostrándose correctamente en el frontend.

El microservicio se distribuye mediante una imagen Docker, cuenta con integración continua mediante GitHub Actions y dispone de pruebas de componente que cubren escenarios positivos y negativos.

Microservicio Scheduled Payments

El microservicio de Pagos Programados implementa una API REST completa para la creación, consulta, actualización y eliminación de pagos programados, cumpliendo con los requisitos de versionado, persistencia en MongoDB y validación de datos mediante Pydantic.

Para garantizar la correcta ejecución de los pagos en el momento programado, el microservicio utiliza la API externa NTP (pool.ntp.org) para la sincronización temporal. Asimismo, se han implementado mecanismos de rate limit en determinados endpoints para asegurar un funcionamiento controlado y evitar sobrecarga.

El microservicio se encuentra documentado con Swagger/OpenAPI, desplegado mediante Docker, integrado en un flujo de CI con GitHub Actions y cubierto por pruebas de componente.

Aplicación basada en microservicios

Ambos microservicios se integran dentro de una aplicación basada en microservicios mediante un API Gateway, que centraliza la autenticación mediante JWT, el enrutamiento de peticiones y los mecanismos de throttling en función del plan de suscripción del usuario.

La aplicación dispone de un frontend común desarrollado en React, que permite interactuar con las funcionalidades expuestas por los microservicios, adaptando el comportamiento según el plan de precios contratado (Básico, Premium o Pro). En el caso de nuestros microservicios, esta adaptación se refleja en los límites de los pagos programados y en el tipo de notificaciones que se generan y muestran al usuario.

A nivel de seguridad y control de acceso, la autenticación se gestiona mediante JWT, emitidos por el microservicio de User Auth y validados de forma centralizada en el API Gateway. Asimismo, el Gateway aplica mecanismos de throttling y limitación de uso en función del plan de suscripción, asegurando una gestión controlada de la capacidad del sistema.

Para evitar la redundancia, todos los enlaces que justifican los puntos aquí expuestos se encuentran en el [apartado 2](#) de la memoria.

9. Análisis de esfuerzos

Jose Luis Costela Guijosa

Los esfuerzos se han ido calculando con la aplicación *Clockify* y se resumen en la siguiente tabla:

Fecha	Descripción actividad	Duración
13/11/25	Reunión inicial grupal	1 h y 20 min
16/11/25	Reunión inicial pareja	1 h
16/11/25	Visión de videos	1 h y 10 min
19/11/25	Revisión de aplicación	1 h y 10 min
19/11/25	Definición de api	45 min
20/11/25	Desarrollo de código	4 h y 10 min
30/11/25	Visión de videos	2 h y 20 min
30/11/25	Test aplicación en kubernetes	1 h
07/12/25	Test aplicación en kubernetes	1 h y 30 min
07/12/25	Añadir funcionalidad ejecución transferencia	2 h
14/12/25	Añadir funcionalidad ejecución transferencia	1 h
31/12/25	Arreglo errores y frontend	3 h
04/01/26	Rate limit	4 h
05/01/26	Limitación por suscripción y unificar microservicios	6 h 30 min
07/01/26	Arreglo errores y unificar microservicios	12 h 30 min
08/01/26	Arreglo errores y documentación	14 h 30 min
09/01/26	Documentación	3 h
TOTAL:		60 h y 55 min

Elena Berdugo Rodríguez

En recopilación de todos los esquemas de organización elaborados y reuniones establecidas, se pueden desglosar en la siguiente tabla los esfuerzos realizados:

Fecha	Descripción actividad	Duración
13/11/25	Reunión inicial grupal	1 h y 30 min
16/11/25	Reunión inicial pareja	1 h
18/11/25	Análisis del microservicio elegido y vídeos práctica	1 h y 30 min
23/11/25	Implementación plantilla inicial y primera subida a GitHub	2 h
26/11/25	Desarrollo microservicio inicial	4 h
27/11/25	Desarrollo microservicio y MongoDB	2 h

31/11/25	Ajustes de lógica de negocio y errores	3 h
02/12/25	Validaciones con Pydantic y pruebas	2 h
10/12/25	Adición de endpoints y pruebas	2 h
11/12/25	Organización de próximos pasos	1 h
13/12/25	Documentación Swagger	4 h
28/12/25	Integración SendGrid y pruebas de envío	4 h
01/01/26	Integración con frontend común	8h
02/01/26	Integración con API Gateway y otros microservicios	5 h
03/01/26	Tests del microservicio y ajustes	6 h
05/01/26	Integración Docker y errores	8 h
07/01/26	Integración con kubernetes y solvencia errores	12 h
08/01/26	Arreglo de errores y documentación	12 h
09/01/26	Documentación final y memoria	7 h
	TOTAL:	86 h

10. Uso de Inteligencia Artificial

Durante el desarrollo del proyecto se ha utilizado la inteligencia artificial como una herramienta de apoyo en distintas fases del trabajo. Su uso ha estado orientado principalmente a facilitar la comprensión de conceptos técnicos y a resolver dudas puntuales que han ido surgiendo a lo largo del desarrollo.

En concreto, la inteligencia artificial se ha empleado para:

- Apoyar la comprensión del funcionamiento de frameworks y tecnologías utilizadas, como Quart, la arquitectura de microservicios y la comunicación entre servicios.
- Ayudar en la resolución de problemas concretos de implementación, proponiendo posibles enfoques que posteriormente han sido analizados y adaptados.
- Asistir en la organización del proyecto y en la aplicación de buenas prácticas de desarrollo, especialmente en la estructura del código y la configuración de los entornos.
- Facilitar la redacción y revisión de la documentación, mejorando la claridad y la coherencia del contenido.

- Servir de apoyo en el diseño de pruebas automáticas y en la definición de distintos escenarios de validación.

En todo momento, la inteligencia artificial se ha utilizado de forma responsable, como una herramienta complementaria al trabajo realizado por los autores, sin sustituir el razonamiento técnico ni las decisiones de diseño adoptadas durante el desarrollo del proyecto.