Entrega No. 1

Implementación de una API REST escalable con orquestación de tareas asíncronas para el procesamiento de archivos

Objetivos

Diseñar e implementar una aplicación web **escalable**, **orientada** a la gestión de archivos y al procesamiento asíncrono de tareas, que **garantice** un desempeño eficiente, escalable, y que **soporte** de manera confiable la concurrencia de múltiples usuarios.

- Desarrollar una API RESTful escalable y segura para la gestión de usuarios y recursos, con contratos documentados en OpenAPI, autenticación y autorización basada en tokens.
- Implementar un sistema de procesamiento asíncrono que permita la ejecución de tareas en segundo plano de manera eficiente y confiable, incorporando colas de mensajes, mecanismos de reintento con backoff, y manejo de fallos mediante Dead Letter Queues.
- Administrar el almacenamiento de archivos garantizando seguridad, eficiencia y disponibilidad.
- Orquestar el despliegue de la aplicación en un entorno basado en contenedores que asegure su portabilidad, resiliencia y escalabilidad, mediante prácticas de CI/CD, pruebas automatizadas.
- Documentar la arquitectura del sistema, incluyendo los diagramas de niveles (C4), las decisiones de diseño, los contratos de la API, los diagramas de despliegue.
- Implementar el frontend de la aplicación, una interfaz web sencilla, integrada con la API.

Tiempo de dedicación

La presente entrega, correspondiente a la fase desarrollo, está programada para un periodo de **dos semanas**. Durante este tiempo, cada estudiante deberá destinar las horas asignadas en la planificación semanal. Se recuerda la importancia de conformar equipos de trabajo de acuerdo con las directrices establecidas en el curso, como condición clave para el éxito de la actividad.

Componentes de la evaluación

La distribución de la calificación de la entrega está distribuida de la siguiente manera:

1. Diseño e implementación de la API RESTful (40%)
   - Implementación de endpoints: conforme a lo establecido en la sección **especificación del API REST**.
   - Gestión de solicitudes y respuestas: aplicación rigurosa de códigos de estado HTTP y definición estructurada de las respuestas.
   - Validación y manejo de errores: establecimiento de reglas de validación y generación de reportes

de error consistentes y alineados con las mejores prácticas.

2. Autenticación y seguridad (5%)
   - Implementación de JWT: incorporación de tokens para los procesos de autenticación y autorización de usuarios.
   - Protección de endpoints: aseguramiento de rutas críticas mediante la verificación de permisos y controles de acceso.

3. Procesamiento asíncrono de tareas (15%)
   - Configuración del sistema de gestión de tareas asíncronas (por ejemplo, Asynq o Machinery): integración efectiva con el broker de mensajería seleccionado.
   - Implementación y monitoreo de tareas asíncronas, garantizando su ejecución eficiente y trazabilidad.
   - Consideración de Apache Kafka como alternativa válida: su arquitectura distribuida y orientada a eventos permite gestionar grandes volúmenes de mensajes con alta disponibilidad, tolerancia a fallos y escalabilidad horizontal, lo que la convierte en una opción robusta frente a sistemas tradicionales de message brokering.
   - Manejo de errores y reintentos: definición de estrategias para la atención de fallos en la ejecución de tareas asíncronas.

4. Gestión y almacenamiento de archivos (5%)
   - Almacenamiento seguro: gestión eficiente y confiable de los archivos cargados por los usuarios, garantizando integridad y confidencialidad.
   - Conversión y procesamiento: implementación de la lógica de transformación de archivos conforme a los requisitos funcionales y técnicos definidos.
   - Acceso y descarga: provisión de mecanismos seguros y controlados para la recuperación de archivos procesados.
   - Se recomienda la implementación de una capa de abstracción para el almacenamiento de archivos. Adoptar este patrón de diseño, definiendo una interfaz clara (como IStorageService) y utilizando la inyección de dependencias en la aplicación. Este enfoque desacopla por completo la lógica de negocio de los detalles de la infraestructura de almacenamiento. En la práctica, esto garantiza que la futura migración de un sistema de archivos local a un servicio de almacenamiento en la nube, como AWS S3, sea un proceso simple y de bajo riesgo que no requerirá modificaciones en el código de la API. Además, esta arquitectura mejora la mantenibilidad del código y simplifica drásticamente la creación de pruebas unitarias.

5. Implementación del frontend (10%)
   - Desarrollo de vistas y componentes: conforme a la especificación de UI de su preferencia (apóyese en herramientas de IA Generativa).
   - Gestión de interacción y estado: implementación de flujos de usuario, formularios y validaciones en cliente, manejo consistente de errores y notificaciones, y enrutamiento interno.

- Integración con la API y seguridad en cliente: consumo de endpoints definidos, tratamiento de sesiones y tokens.

6. Despliegue y entorno de ejecución (10%)
   - Uso de Docker y Docker Compose: configuración apropiada del entorno de despliegue, garantizando portabilidad y consistencia entre entornos.
   - Configuración de Gunicorn/Uvicorn y Nginx: implementación adecuada de servidores de aplicación y servidor proxy inverso para entornos de producción, asegurando rendimiento y estabilidad.

7. Documentación (10%)
   - Modelo de datos: inclusión del modelo de datos de la aplicación, representado mediante un diagrama Entidad-Relación (ERD) o, en su defecto, a través de una especificación detallada de las entidades, atributos y relaciones del sistema.
   - Documentación de la API: elaboración y centralización de la documentación de los endpoints, así como la ejecución de pruebas correspondientes, mediante Postman.
   - Diagrama de componentes: representación de los principales elementos de la arquitectura, considerando backend, worker, broker y base de datos.
   - Diagrama de flujo de procesos: descripción detallada de las etapas de carga, procesamiento y entrega de un archivo.
   - Despliegue y documentación: representación de la infraestructura de ejecución (máquinas virtuales, contenedores Docker y servicios activos) acompañada de una guía clara, estructurada y reproducible que facilite la réplica del entorno en diferentes contextos.
   - Reporte de análisis de SonarQube: donde se evidencien los resultados del último análisis sobre la rama principal del proyecto. Debe mostrar, al menos:
     - Métricas de **Bugs, Vulnerabilidades y Code Smells**.
     - Nivel de **Cobertura de pruebas unitarias** (%).
     - Duplicación de código (%).
     - Estado del **Quality Gate** (aprobado/rechazado).

8. Plan de pruebas de carga (5%)
   - diseño de un plan de pruebas que evalúe el comportamiento y desempeño de la aplicación bajo distintos niveles de concurrencia y volumen de solicitudes. El plan deberá incluir la definición de métricas clave (latencia, throughput, utilización de recursos y tasa de errores), así como la interpretación de resultados para identificar posibles cuellos de botella y proponer mejoras en la escalabilidad y estabilidad del sistema.

Sugerencias para los equipos

- **Planificación**: Antes de comenzar, es fundamental diseñar la arquitectura del sistema y planificar las tareas a realizar.

- **Buenas prácticas**: Adoptar estándares de codificación y seguir patrones de diseño reconocidos.
- **Pruebas continuas**: Implementar pruebas unitarias y de integración que contribuyan a la calidad del código.
- **Documentación**: Mantener una documentación actualizada facilita el mantenimiento y la escalabilidad del proyecto.

Formato de entrega

1. Respecto a la documentación, se recomienda:
   - Estructuración de la información: organizar los contenidos siguiendo las pautas definidas en la sección anterior, asegurando la inclusión de todos los elementos esenciales.
   - En el archivo principal README.md del repositorio, registre el nombre completo y el correo Uniandes de cada integrante del curso
   - Repositorio y organización de entregas: alojar toda la documentación en el repositorio de Github, dentro de un directorio dedicado (/docs/Entrega_1), y referenciarla en el archivo README.md para facilitar su acceso. Esta misma estructura deberá utilizarse en las entregas posteriores.
   - Sustentación en video: en la ruta /sustentacion/Entrega_1, incluya el enlace a la video sustentación correspondiente a la entrega, asegurando que sea accesible y funcione correctamente.
   - Colecciones de Postman y validación automatizada: crear un directorio específico (/collections) para las colecciones de Postman que contengan las solicitudes y pruebas correspondientes. Dichas colecciones deberán exportarse en formato JSON y almacenarse en el repositorio. La ejecución automatizada de estas pruebas debe validarse mediante herramientas como el CLI **newman**, manteniendo esta estructura en las entregas posteriores.

2. Incluir un conjunto de pruebas automatizadas (unitarias) que validen el correcto funcionamiento de la aplicación.

3. **Archivo .gitignore**: Incluir un archivo **.gitignore** adecuado para excluir archivos y directorios que no deban ser versionados.

4. Publicar una versión (**release**) del código fuente en el repositorio del grupo en GitHub, utilizando etiquetas (**tags**) que sigan el formato de versionado semántico (por ejemplo, v1.0.0) y proporcionando una descripción detallada de los cambios incluidos en dicha versión.

Infraestructura requerida para el despliegue

Para garantizar un despliegue simple y automatizado de la aplicación, se establecen las siguientes

directrices:

- Despliegue en contenedores: la aplicación debe ejecutarse en contenedores Docker para garantizar portabilidad y consistencia.
- Sistema operativo base: Ubuntu según los requerimientos del proyecto.
- Automatización: proveer un archivo docker-compose.yml que orqueste todos los servicios y permita el despliegue completo con docker compose up.

Estas medidas buscan optimizar el proceso de despliegue, facilitando la gestión de la aplicación.

Notas:

- Esta fase no incluye el uso de proveedores de nube pública; dicha implementación será abordada en la siguiente etapa del proyecto.
- Utilice ambientes virtuales para su desarrollo local y valide que su configuración de Docker funcione correctamente en uno de los sistemas operativos indicados; de ser necesario utilice un hipervisor gratuito u open source. Se recomienda Oracle VM VirtualBox.
- Los tutores del curso pueden solicitar una sustentación síncrona y para esto se requiere la aplicación en ejecución.

Recomendaciones y consideraciones

El backend de la aplicación web debe desarrollarse utilizando el lenguaje de programación **Go**. Para su ejecución en un entorno local, la aplicación debe contar al menos con los siguientes componentes:

- Sistema operativo: Ubuntu Server 24.04 LTS.
- Lenguaje del backend: Golang.
- Framework: Gin o Echo
- Base de datos: PostgreSQL.
- Gestión de tareas asíncronas: Asynq o Machinery con Redis o RabbitMQ como message brokers.
- Alternativa: uso de Apache Kafka en lugar de Asynq o Machinery, lo que implica una arquitectura basada en el modelo publish/subscribe.
- A nivel del frontend utilice las tecnologías de su preferencia.
- Servidor web: Nginx, configurado como proxy inverso.

Escenario de negocio y requisitos de la aplicación

Contexto

La **Asociación Nacional de Baloncesto (ANB)** es una organización reconocida por promover el desarrollo del talento emergente en el baloncesto a nivel regional y nacional. Con un enfoque en la inclusión y el impulso de nuevas generaciones de jugadores, la ANB busca identificar y proyectar a los futuros talentos del deporte a través de iniciativas innovadoras que aprovechan la tecnología para democratizar el acceso a sus programas de selección.

En respuesta al creciente interés de jóvenes atletas que sueñan con ser parte de los equipos profesionales, la ANB lanza la iniciativa **ANB Rising Stars Showcase**, un programa que se orienta a descubrir a los mejores jugadores aficionados de diferentes regiones del país, brindándoles la oportunidad de competir en un torneo de exhibición frente a los cazatalentos de la liga. Se trata, por tanto, de una competencia abierta que abarca **varias ciudades en el territorio nacional**.

Como parte del proceso de preselección, los jugadores aficionados enviarán **videos cortos demostrando sus habilidades** (entrenamientos, jugadas destacadas, lanzamientos, etc.). La ANB requiere el desarrollo de una **plataforma web** que sirva como **centro de carga, almacenamiento y evaluación de tales videos**, permitiendo que tanto el **público general** como un **jurado especializado** voten por los jugadores más destacados.

Al finalizar el proceso de votación, los jugadores con el mayor número de votos en cada ciudad serán seleccionados para integrar los equipos que participarán en el torneo **Rising Stars Showcase**, con la posibilidad de ser reclutados por equipos profesionales.

Requerimientos de la ANB

- Los **jugadores** podrán registrarse, crear un perfil y subir sus videos de prueba.
- La plataforma deberá realizar **procesamiento automático** de los videos cargados:
  - Recorte de duración a un máximo de **30 segundos**.
  - Ajuste de resolución y formato de aspecto con el fin de mantener una calidad óptima sin sobrecargar los servidores.
  - Agregar una marca de agua de ANB para autenticar el contenido.
  - Eliminar audio, puesto que no es relevante para la evaluación de los jugadores.
- El **público** podrá ver los videos y votar.
- Se generará un **ranking dinámico**, mostrando los jugadores más votados.
- Las votaciones deben ser controladas para evitar fraudes o múltiples votos por usuario.

El sistema debe habilitar la conversión de archivos de video de manera asíncrona o mediante procesos batch, con el objetivo de optimizar la experiencia del usuario y evitar tiempos de espera prolongados. Una vez culminado el procesamiento, el estado del archivo debe actualizarse

automáticamente a "procesado", tal que los usuarios sean capaces de visualizar y utilizar sus videos sin interrupciones.

Impacto esperado

La **ANB Rising Stars Showcase** permitirá democratizar el acceso al proceso de selección de nuevos talentos en el baloncesto, reduciendo barreras geográficas y económicas. La plataforma tecnológica apoyará la misión de la ANB de fomentar el deporte e identificar jóvenes promesas que puedan integrarse en el baloncesto profesional.

Alcance del proyecto

Registro de jugadores: Los jugadores aficionados pueden crear sus cuentas en la plataforma para participar en el proceso de selección y hacer seguimiento a sus postulaciones. Se requiere la siguiente información: nombre, apellidos, ciudad, país, correo electrónico.

Carga de videos: Los jugadores podrán subir videos cortos donde demuestren sus habilidades en el baloncesto, como lanzamientos, dribles, jugadas defensivas u otras destrezas. El video debe contar con una duración mínima de 20 segundos y máxima de 60 segundos, en calidad 1080p o superior.

Procesamiento de videos: Los archivos de video subidos por los usuarios serán procesados en segundo plano, evitando bloqueos en la API y mejorando la escalabilidad del sistema.

- La plataforma debe recortar cada video a una duración máxima de **30 segundos**.
- Ajustarse a una relación de aspecto **16:9** y resolución **720p**.
- Incluir una cortinilla **de apertura y cierre** con el logotipo oficial de la **Asociación Nacional de Baloncesto (ANB)**. Máximo deben agregarle 5 segundos extra al video.

Una vez completado el procesamiento, el estado del archivo se actualizará automáticamente en la base de datos, reflejando su disponibilidad para visualización y evaluación.

Votación: La plataforma permitirá al público en general votar por sus videos favoritos.

Ranking: Un ranking dinámico revelará los jugadores mejor posicionados, de acuerdo en el número de votos recibidos.

Descripción funcional de los servicios

Gestión de usuarios (autenticación y registro)

1. Registro de Jugadores

   El sistema debe permitir que los jugadores aficionados se registren en la plataforma con el fin de participar en el proceso de selección. El registro debe garantizar la validez de los datos, incluyendo la verificación de un correo electrónico único. Asimismo, se requiere la implementación de mecanismos de seguridad para la gestión de contraseñas, asegurando su cifrado y almacenamiento mediante hashing.

```
{
  "first_name": "John",
  "last_name": "Doe",
  "email": "john@example.com",
  "password1": "StrongPass123",
  "password2": "StrongPass123",
  "city": "Bogotá",
  "country": "Colombia"
}
```

   En la solicitud de registro se solicitan dos campos de contraseña (password1 y password2) únicamente con el propósito de validar que el usuario introduzca y confirme una misma contraseña, minimizando errores de tipeo y asegurando que el valor definido sea recordado.

   No obstante, en el sistema solo se almacena un único valor de contraseña (después de aplicar el correspondiente proceso de hashing y cifrado), descartándose el campo redundante tras la validación inicial.

   Códigos de respuesta:

Código	Descripción
201	Usuario creado exitosamente.
400	Error de validación (email duplicado, contraseñas no coinciden).

2. Inicio de Sesión

   El sistema debe permitir que los usuarios se autentiquen en la plataforma mediante el suministro de su correo electrónico y contraseña. Como respuesta, se debe generar y devolver un token JWT que deberá ser utilizado en todas las solicitudes autenticadas posteriores. Asimismo, es obligatorio implementar un control de sesiones basado en tokens JWT, contemplando mecanismos de expiración (es suficiente con establecer tiempos de expiración cortos para los tokens, es una solución simple).

   Ejemplo de request

```
{
  " email":" john@example.com",
  "password":"StrongPass123"
}
```


Ejemplo de respuesta exitosa


```
{
  "access_token":"eyJ0eXAiOiJKV1QiLCJhbGci...", "token_type":
  "Bearer",
  "expires_in": 3600
}
```


Códigos de respuesta

Código	Descripción
200	Autenticación exitosa, retorna token.
401	Credenciales inválidas.

Gestión de videos (carga, procesamiento y acceso)

1. Carga de video

El sistema debe permitir que los jugadores suban un video. Dicho video se almacenará en el sistema de archivos y, de manera automática, se registrará una tarea de procesamiento asíncrono encargada de recortar el contenido, ajustarlo al formato **16:9** y añadir los logos institucionales de la **ANB**.

Manejo de estados: el archivo deberá contar con un flujo de estados claramente definido. Inicialmente se marcará como *"uploaded"* y, una vez completado el procesamiento en segundo plano, pasará al estado *"processed"*.

En lugar de esperar a que un proceso externo consulte la base de datos, el endpoint debe **activamente encolar una tarea** en el broker de mensajería (Asynq o Machinery/Kafka). El API no espera a que el video se procese. Una vez que la tarea ha sido encolada, responde inmediatamente al cliente.

Este ajuste asegura que el sistema sea más eficiente y escalable, ya que las tareas se distribuyen activamente a los workers en el momento de su creación, en lugar de depender de un proceso de sondeo (polling) que consume recursos innecesariamente.

Parámetros (form-data)

Nombre	Tipo	Requerido	Descripción
video_file	archivo	Sí	Archivo de video en formato MP4, máximo 100MB.
title	string	Sí	Título descriptivo del video.

Ejemplo de respuesta exitosa


```
{
  "message": "Video subido correctamente. Procesamiento en curso.",
```

```
  "task_id": "123456"
}
```


Códigos de Respuesta

Código	Descripción
201	Video subido exitosamente, tarea creada.
400	Error en el archivo (tipo o tamaño inválido).
401	Falta de autenticación.

2. Consultar mis videos

Permite al jugador consultar el listado de sus videos subidos, junto con el estado de procesamiento y las URLs de acceso (si el procesamiento está completo).

Ejemplo de respuesta


```
[
  {
    "video_id": "123456",
    "title": "Mi mejor tiro de 3",
    "status": "processed ",
    "uploaded_at": "2025-03-10T14:30:00Z",
    "processed_at": "2025-03-10T14:35:00Z",
    "processed_url": "https://anb.com/videos/processed/123456.mp4"
  },
  {
    "video_id": "654321",
    "title": "Habilidades de dribleo",
    "status": "uploaded ",
    "uploaded_at": "2025-03-11T10:15:00Z"
  }
]
```


Códigos de respuesta

Código	Descripción
200	Lista de videos obtenida.
401	Falta de autenticación.

3. Consultar detalle de un video específico

Permite recuperar el detalle de una tarea de video específica, incluyendo la URL de descarga del video procesado, si está disponible.

Ejemplo de respuesta exitosa


```
{
  "video_id": "a1b2c3d4",
  "title": "Tiros de tres en movimiento",
  "status": "processed ",
  "uploaded_at": "2025-03-15T14:22:00Z",
  "processed_at":  "2025-03-15T15:10:00Z",
```

```
  "original_url":"https://anb.com/uploads/a1b2c3d4.mp4",
  "processed_url":"https://anb.com/processed/a1b2c3d4.mp4",
  "votes": 125
}
```


Códigos de respuesta

Código	Descripción
200 OK	Consulta exitosa. Se devuelve el detalle del video.
401 Unauthorized	El usuario no está autenticado o el token JWT es inválido o expirado.
403 Forbidden	El usuario autenticado no tiene permisos para acceder a este video (no es el propietario).
404 Not Found	El video con el video_id especificado no existe o no pertenece al usuario.

4. Eliminar video subido

Permite al jugador eliminar uno de sus videos (tanto el original como el procesado), solo si no ha sido publicado para votación o aún no ha sido procesado.

Ejemplo de respuesta exitosa


```
{
  "message": "El video ha sido eliminado exitosamente.",
  "video_id": "a1b2c3d4"
}
```


Códigos de respuesta

Código	Descripción
200 OK	El video ha sido eliminado correctamente. Se confirman los cambios en la base de datos y almacenamiento.
400 Bad Request	El video no puede ser eliminado porque no cumple las condiciones (por ejemplo, ya está habilitado para votación).
401 Unauthorized	El usuario no está autenticado o el token JWT es inválido o expirado.
403 Forbidden	El usuario autenticado no tiene permisos para eliminar este video (no es el propietario).
404 Not Found	El video con el video_id especificado no existe o no pertenece al usuario autenticado.

Sistema de votación pública

1. Listar videos disponibles para votar

Lista todos los videos públicos habilitados para votación.

2. Emitir voto por un video

Permite a un usuario registrado emitir un voto por un video específico. Un usuario solo puede

votar una vez por video. Un usuario puede votar por varios videos, pero solo puede votar una vez por video.

Ejemplo de respuesta

```
{
   "message": "Voto registrado exitosamente."
}
```

Códigos de respuesta

Código	Descripción
200	Voto exitoso.
400	Ya has votado por este video.
404	Video no encontrado.
401	Falta de autenticación.

Ranking de jugadores

1. Consultar tabla de clasificación

   Provee un ranking actualizado, en el que los competidores son organizados respecto al número de votos obtenidos. Puede incluir filtros para mostrar diferentes rangos de posiciones, por ejemplo, filtrar por ciudad.

   Si el número de videos y votos es alto, calcular este ranking en tiempo real con cada llamada a la API puede generar una carga excesiva en la base de datos y aumentar la latencia.

   Recomendación: Implementar una estrategia de caching (ej. en Redis) para los resultados del ranking, con un tiempo de vida (TTL) corto (p. ej., 1 a 5 minutos). Alternativamente, se puede utilizar una vista materializada en PostgreSQL que se actualice periódicamente.

   Ejemplo de respuesta

```
[
   {
      "position": 1,
      "username":"superplayer", "city": "Bogotá",
      "votes": 1530
   },
   {
      "position": 2, "username":"nextstar", "city": "Bogotá", "votes": 1495
   }
]
```

   Códigos de respuesta

Código	Descripción
200	Lista de rankings obtenida.
400	Parámetro inválido en la consulta.

Especificación del API REST

En este proyecto deberá crear los siguientes endpoints para su API REST. Esta definición deberá ser respetada a lo largo del desarrollo del proyecto.

	Endpoint	Método	Descripción	Autenticación	Notas
1	/api/auth/signup	POST	Registro de nuevos jugadores en la plataforma.	No	Valida email único y confirmación de contraseña.
2	/api/auth/login	POST	Autenticación de usuarios y generación de token JWT.	No	Devuelve token JWT válido para autenticación posterior.
3	/api/videos/upload	POST	Permite a un jugador subir un video de habilidades.	Sí (JWT)	Inicia proceso asíncrono de procesamiento del video.
4	/api/videos	GET	Lista todos los videos subidos por el usuario autenticado.	Sí (JWT)	Muestra estado: "uploaded" o "processed".
5	/api/videos/{video_id}	GET	Obtiene el detalle de un video específico del usuario.	Sí (JWT)	Incluye URLs para ver/descargar los videos (si está listo).
6	/api/videos/{video_id}	DELETE	Elimina un video propio, si aún es permitido.	Sí (JWT)	Solo si el video no ha sido publicado para votación.
7	/api/public/videos	GET	Lista los videos públicos disponibles para votación.	Opcional	
8	/api/public/videos/{video_id}/ vote	POST	Emite un voto por un video público.	Sí (JWT)	Limita un voto por usuario por video.
9	/api/public/rankings	GET	Muestra el ranking actual de los jugadores por votos acumulados.	No	Soporta paginación y filtros.

Nota: En los endpoints que requieran autenticación, como **consultar mis videos**, el **email** no debe ser enviado por el cliente, ya sea en el cuerpo de la solicitud o como parámetro en la URL.

La identidad del usuario autenticado debe obtenerse exclusivamente a partir del token JWT incluido en el encabezado **Authorization**. El backend es responsable de validar dicho token y asociar la operación al usuario correspondiente, garantizando así la seguridad y la integridad de la información gestionada por la aplicación.

Para el procesamiento de conversión de archivos, la aplicación debe ejecutar un proceso asíncrono que garantice una experiencia de usuario fluida.

Con el fin de evitar que el usuario permanezca esperando mientras sus videos son procesados para cumplir con las características técnicas definidas, la edición de los archivos se realiza mediante tareas en segundo plano, gestionadas de forma asíncrona o en procesos batch. Una vez finalizado el procesamiento, el estado del archivo se actualiza a "processed" en la aplicación.

Por lo tanto, la aplicación deberá contar con un proceso asíncrono y distribuido, que se ejecute en segundo plano. Este proceso se encargará de consultar de manera periódica la base de datos para identificar archivos en estado "uploaded", y proceder a realizar las siguientes acciones:

- Editar el video para ajustarlo a las especificaciones establecidas, como la duración máxima, la relación de aspecto y la inclusión de los elementos gráficos requeridos (por ejemplo, el logo de la ANB).
- Guardar el video procesado en el sistema de archivos, conservando también el archivo original.
- Actualizar el estado del video a "processed" en la base de datos.

Estas funcionalidades deberán ofrecerse dentro de una única aplicación web, ejecutándose de manera asíncrona mediante un sistema de tareas.

Por otro lado, aunque se desarrollará una interfaz gráfica de usuario, la validación de los servicios deberá realizarse mediante un conjunto de escenarios de prueba automatizados utilizando Postman. Esta herramienta permitirá documentar y probar los endpoints del API REST.

El equipo de trabajo deberá crear un workspace en Postman, en el cual colaborarán para consolidar la colección de endpoints de la aplicación.

Dicha colección deberá incluir:

- Parámetros requeridos para cada solicitud.
- Escenarios de prueba de ejemplo.
- Documentación de los mensajes de error y excepciones.

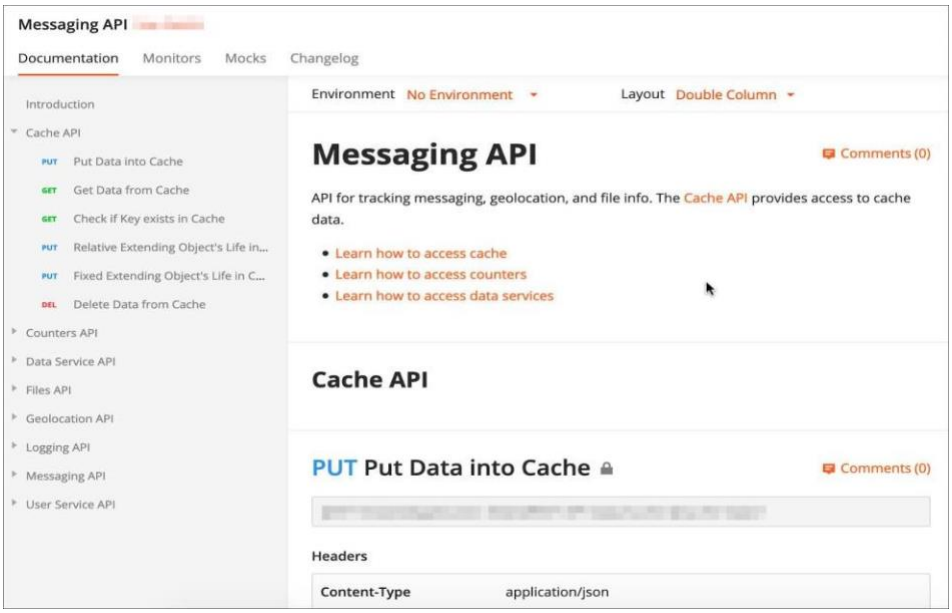

Ilustración 1. Ejemplo, documentación de un API REST en POSTMAN

Para facilitar la automatización de la validación de los endpoints de la API, se debe crear un directorio específico en el repositorio del proyecto. Las colecciones de Postman, que contienen las solicitudes y las pruebas correspondientes, deben exportarse en formato JSON y almacenarse en la ruta ***/collections***.

Se recomienda validar la ejecución automatizada de dichas pruebas utilizando herramientas como el **CLI newman**. Esto asegurará la correcta validación de los endpoints de manera reproducible y consistente.

Esta estructura deberá mantenerse para las entregas posteriores, estandarizando el manejo de las pruebas y la documentación de la API REST en el proyecto.

Además de la **colección de pruebas**, es necesario crear un archivo de entorno llamado ***postman_environment.json***. Este archivo debe incluir todas las **variables necesarias para la ejecución automatizada de las pruebas**, como, por ejemplo:

base_url	URL base de la API desplegada (ej. http://localhost:8000/api).
deploy_url	URL base de la API desplegada (ej. http://ip_de_su_proyecto:8000/api).

Nota: Para la primera entrega no se hará uso de la variable **deploy_url**. Este variable debe existir para que pueda ser validado su despliegue en entregas posteriores.

Pruebas unitarias

Como parte fundamental del desarrollo del proyecto, cada equipo debe implementar y ejecutar pruebas unitarias que validen el correcto funcionamiento de los componentes principales de la aplicación. Las pruebas deben cubrir, al menos, los casos de uso más relevantes y los servicios expuestos a través de la API REST.

Estas pruebas deben garantizar la calidad del código, facilitar la detección temprana de errores y permitir validar automáticamente el comportamiento de la aplicación antes de su despliegue.

GitHub CI/CD

La Integración Continua (CI) en GitHub constituye una práctica esencial que permite automatizar la construcción, validación y aseguramiento de la calidad del software. A través de GitHub Actions, cada cambio enviado al repositorio (por ejemplo, mediante *pull requests* o *push* a una rama principal) activa un pipeline que ejecuta de manera sistemática las validaciones definidas.

En este caso, el pipeline se limitará a dos etapas fundamentales:

- **Ejecución de pruebas unitarias**: se valida la funcionalidad del código asegurando que cada componente cumpla con el comportamiento esperado.
- **Construcción automática de la aplicación**: se genera el artefacto de la aplicación en un entorno reproducible, garantizando consistencia en futuros despliegues.

Adicionalmente, el pipeline incorpora la validación de la calidad del código con SonarQube, lo que permite detectar vulnerabilidades, errores, code smells y problemas de mantenibilidad antes de integrar los cambios en la rama principal.

Análisis de capacidad

Lea el documento anexo "**Entrega 1 - Análisis de Capacidad**" para conocer toda la especificación del análisis de capacidad que debe realizar a la aplicación.

El plan de análisis de capacidad debe ser organizado y entregado dentro del repositorio del proyecto. Para ello, se debe crear una carpeta específica llamada ***/capacity-planning***, donde se almacenará el documento correspondiente.

El plan debe estar documentado en un archivo llamado ***plan_de_pruebas.md***, el cual debe incluir el plan análisis detallado de capacidad de la aplicación, los escenarios de carga planteados, las métricas seleccionadas, los resultados esperados y las recomendaciones para escalar la solución. Esta estructura debe mantenerse de forma consistente en las futuras entregas del proyecto.