Universidad de los Andes
Facultad de Ingeniería

Desarrollo de Soluciones Cloud – ISIS 4426,

Proyecto 1

DOCUMENTACIÓN TÉCNICA

Grupo 5

María Catalina Ibáñez, Jairo Céspedes, David Saavedra y José Manuel Moreno

Contenido

Descripción.

Enlace del repositorio: https://github.com/Cloud-G05/project1.git
Enlace del video sustentación: https://youtu.be/Cq38mbTSix0
Enlace para acceder a la documentación del API: http://localhost:8000
Enlace para acceder a la aplicación: http://localhost:3000

Para poder acceder a la documentación del API y de la aplicación es necesario que esté corriendo el proyecto. Las instrucciones de cómo correr el proyecto las encuentra en el README.md del repositorio en Github.

Este documento describe la arquitectura técnica del Proyecto 1. La aplicación web desarrollada proporciona una funcionalidad gratuita para la conversión de diversos formatos de archivos (.odt, .docx, .pptx., .xlsx) a PDF, utilizando una arquitectura asíncrona para la gestión de tareas en batch, permitiendo a los usuarios no tener que esperar durante la conversión, sino recibir una notificación una vez que el proceso ha finalizado.

La solución se ha diseñado con una arquitectura de backend basada en Python, utilizando FastAPI para la construcción de la API REST, PostgreSQL como sistema de gestión de bases de datos, Celery para la orquestación de eventos en batch usando Redis como bróker de mensajería y React para el desarrollo del frontend.

Requerimientos.

El proyecto se construyó utilizando el siguiente stack tecnológico, con el propósito de cumplir con los objetivos de escalabilidad, eficiencia y facilidad de uso:

1. **Framework Rest API:** FastAPI (versión 0.96.0), proporciona una plataforma rápida y eficiente para construir APIs REST con Python.
2. **Servidor HTTP Python:** Uvicorn (versión 0.22.0), un servidor ASGI para alojar y servir la aplicación FastAPI.
3. **Librería de validación de datos:** Pydantic (versión 1.10.9), utilizada para la validación y gestión de datos de entrada y salida en la API.
4. **Gestión de base de datos:** SQLAlchemy (versión 2.0.15), ORM que permite interactuar con la base de datos de manera eficiente.
5. **Manejo de tiempo y zonas horarias:** PyTZ (versión 2024.1), proporciona soporte para zonas horarias.
6. **Autenticación y seguridad:** fastapi_jwt_auth (versión 0.5.0), permite el manejo de JWT para la autenticación segura en la aplicación.

7. **Trabajo asíncrono en Batch:** Celery (versión 5.3.6), junto con Redis (versión 5.0.1) como bróker de mensajes, para manejar tareas asíncronas y procesamiento en batch.
8. **Interacción con bases de datos PostgreSQL:** psycopg2 (versión 2.9.9) y psycopg2-binary (versión 2.9.9).
9. **Desarrollo de Frontend:** React, elegido por su flexibilidad y eficiencia en la construcción de interfaces de usuario interactivas.

Archivos:

Este conjunto de diagramas detalla la arquitectura de la aplicación, destacando tanto la estructura de contenedores Docker como la interacción de los componentes de software.

Diagrama de Arquitectura

El sistema está compuesto por contenedores Docker que encapsulan diferentes partes de la aplicación, lo que facilita la escalabilidad, el aislamiento de dependencias y la consistencia entre entornos de desarrollo y producción.

- **Frontend Docker**: Este contenedor aloja la interfaz de usuario desarrollada con React.js. Se comunica con el backend para enviar solicitudes de conversión de archivos y recibir actualizaciones de estado.

- **Backend Docker**: Este contenedor ejecuta el servidor de aplicaciones FastAPI que gestiona las solicitudes HTTP, procesa la lógica de negocio, interactúa con la base de datos PostgreSQL y comunica las tareas a ejecutar al worker de Celery.

- **Celery Docker**: Este contenedor gestiona las tareas asíncronas, como la conversión de archivos. Opera en conjunto con el backend y actualiza el estado de las tareas en la base de datos.

- **PostgreSQL Docker**: Este contenedor ejecuta el sistema de gestión de bases de datos PostgreSQL, donde se almacenan los datos de los usuarios, los archivos y el estado de las tareas de conversión.

- **Redis Docker:** Este contenedor aloja la base de datos clave-valor Redis, que actúa como un bróker y backend para Celery. Facilita la comunicación asíncrona entre el backend y el worker de Celery, almacenando las tareas en cola y los resultados de las mismas. Aunque no se declara un Dockerfile específico para Redis, esta imagen se levanta dentro del entorno Docker a través de Docker Compose, como parte de la arquitectura de Celery.

Estos contenedores están diseñados para trabajar juntos, asegurando una separación clara de responsabilidades y facilitando la escalabilidad y el mantenimiento del sistema.

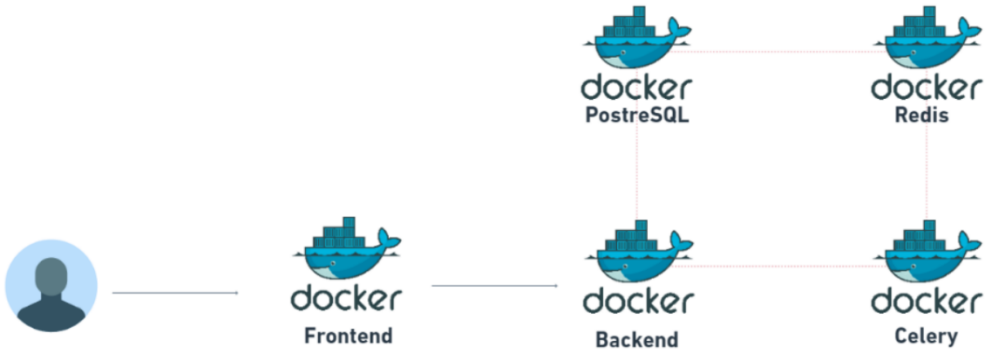

Ilustración 1. Diagrama de Arquitectura

Base de datos

La base de datos del proyecto ha sido implementada utilizando PostgreSQL, seleccionada por su robustez, escalabilidad y compatibilidad con la gestión de grandes conjuntos de datos. A continuación, se detalla la estructura de la base de datos reflejada en el diagrama de relaciones proporcionado:

Entidades Principales:

1. **User (Usuario):**
    - **username: str** - El nombre de usuario, el cual es único para cada usuario.
    - **email: str** - La dirección de correo electrónico del usuario, utilizado como identificador único para cada usuario.
    - **password: str** - La contraseña del usuario para autenticación.

Cada usuario puede tener asociadas múltiples tareas de conversión de archivos, lo que refleja una relación uno a muchos con la entidad Task.

2. **Task (Tarea):**
    - **id: str** - Un identificador único para cada tarea de conversión.

- **name: str** - El nombre o título asignado a la tarea de conversión.
- **original_file_ext: str** - La extensión del archivo original antes de la conversión.
- **file_conversion_ext: str** - La extensión deseada del archivo después de la conversión.
- **available: bool** - Un valor booleano que indica si el archivo convertido está disponible para descarga.
- **status: enum** - Un enumerado que representa el estado de la tarea (por ejemplo, 'UPLOADED' para tareas cargadas y 'PROCESSED' para tareas cuya conversión ha finalizado).
- **time_stamp: datetime** - La fecha y hora en que se creó la tarea.
- **input_file_path: str** - La ruta de acceso al archivo original almacenado.
- **output_file_path: str** - La ruta de acceso al archivo convertido almacenado.

La entidad Task está vinculada a la entidad User, indicando que cada tarea está asociada a un usuario específico.

Relaciones:

- **Relación Usuario-Tarea:** Cada usuario puede tener varias tareas asociadas, pero cada tarea está vinculada a un solo usuario. Esto establece una relación uno a muchos entre User y Task, representada en el diagrama.

Enumeraciones:

- **status: enum** - Define los posibles estados de una tarea con los valores 'UPLOADED' (indicando que el archivo ha sido cargado, pero no procesado aún) y 'PROCESSED' (indicando que el archivo ha sido procesado y está listo para ser descargado).

Diagrama de relaciones

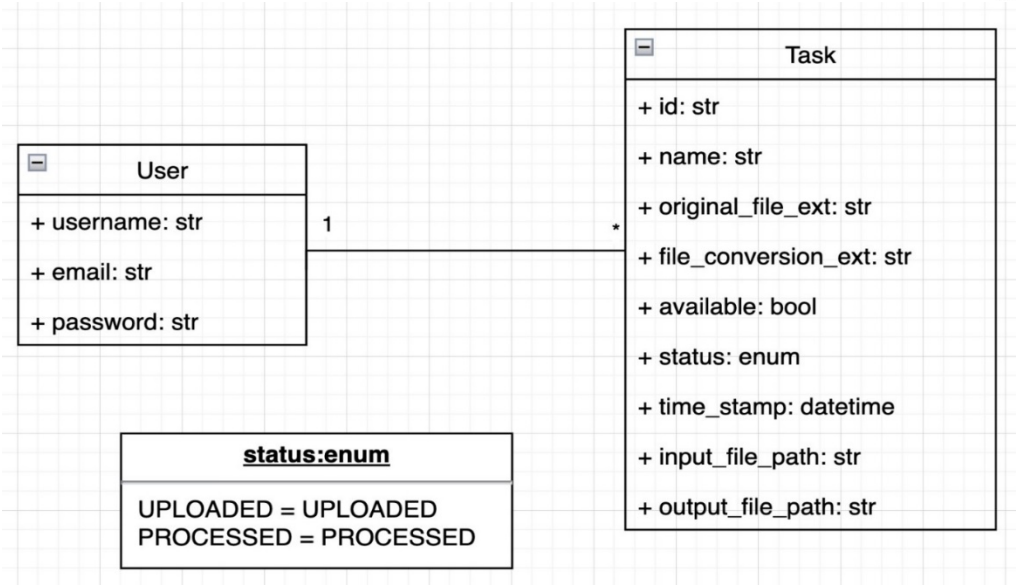

Ilustración 2. Diagrama de relaciones base de datos

Este diagrama de relaciones ofrece una visión clara de cómo se estructura la base de datos para soportar las funcionalidades de la aplicación web, permitiendo la gestión eficiente de usuarios y tareas de conversión de archivos. Debe tenerse en cuenta que la actualización de status depende de una lectura del proceso ejecutado en celery mediante su bróker de comunicación Redis.

Diagrama de componentes

El diagrama de componentes proporciona una visión clara de la arquitectura de la solución implementada para el proyecto. Este esquema refleja la interacción entre los distintos componentes de la aplicación, asegurando una operación eficiente y alineada con los estándares actuales de la industria para soluciones web escalables.

Componentes Principales:

1. **Usuario (User)**:
    - Autenticación mediante **JSON Web Tokens (JWT)**: Asegura una comunicación segura y verifica la identidad del usuario al interactuar con la plataforma.
2. **Frontend (React.js)**:
    - Proporciona una interfaz de usuario interactiva y dinámica, desarrollada con el framework moderno de JavaScript, React.js.

- Envía solicitudes HTTP al backend, autenticadas mediante JWT, y maneja las respuestas para reflejar los cambios en la interfaz de usuario.

3. **Backend (FastAPI)**:
   - Gestiona las solicitudes HTTP provenientes del frontend, procesando la lógica de negocio y las operaciones de la aplicación.
   - Interactúa con la base de datos PostgreSQL para recuperar o almacenar información y con la cola de tareas para la gestión de procesos asíncronos.
   - Envía y recibe datos en formato JSON, asegurando un intercambio de datos eficiente y estandarizado.
4. **Base de Datos (PostgreSQL)**:
   - Almacena y gestiona los datos de la aplicación, incluyendo información de usuarios y detalles de las tareas de conversión.
   - Recibe consultas del backend y actualiza el estado de las tareas según los cambios en la cola de tareas (TaskQueue).
5. **Cola de Tareas (Celery)**:
   - Maneja procesos asíncronos, como las tareas de conversión de archivos, permitiendo que el backend responda a solicitudes de usuario sin retrasos.
   - Se comunica con la base de datos para actualizar el estado de las tareas, asegurando que los usuarios puedan rastrear el progreso de sus conversiones.
6. **Bróker de Mensajes (Redis)**:
   - Actúa como intermediario entre el backend y Celery, facilitando la distribución y gestión de tareas.
   - Permite una comunicación eficaz y la sincronización del estado entre la base de datos y la cola de tareas.

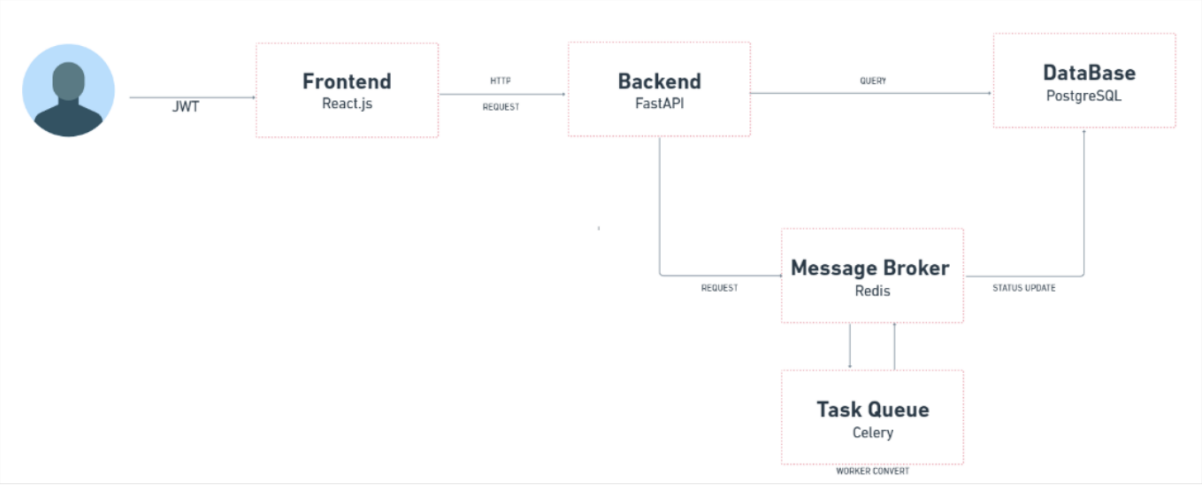

Ilustración 3. Diagrama de componentes

- La utilización de **JWT** para la autenticación entre el usuario y el sistema garantiza la seguridad de la comunicación y protege los datos sensibles.
- La elección de **React.js** para el frontend y **FastAPI** para el backend refleja un compromiso con la eficiencia, la escalabilidad y las mejores prácticas de desarrollo moderno.
- La integración de **Redis** como bróker de mensajes para **Celery** optimiza la orquestación de tareas en segundo plano, permitiendo actualizaciones en tiempo real del estado de las tareas en la base de datos.

Este diagrama de componentes subraya la estructura cohesiva y bien organizada de la aplicación, destacando la interacción entre sus distintos elementos y garantizando una solución robusta, flexible y escalable.

Diagrama de despliegue

En consideración a que la presente solución no emplea elementos de la nube, el diagrama de despliegue representa los contenedores agrupados en el archivo Docker compose.

Las relaciones de dichos contenedores se encuentran previamente detalladas en el diagrama de arquitectura, así como en el diagrama de componentes.

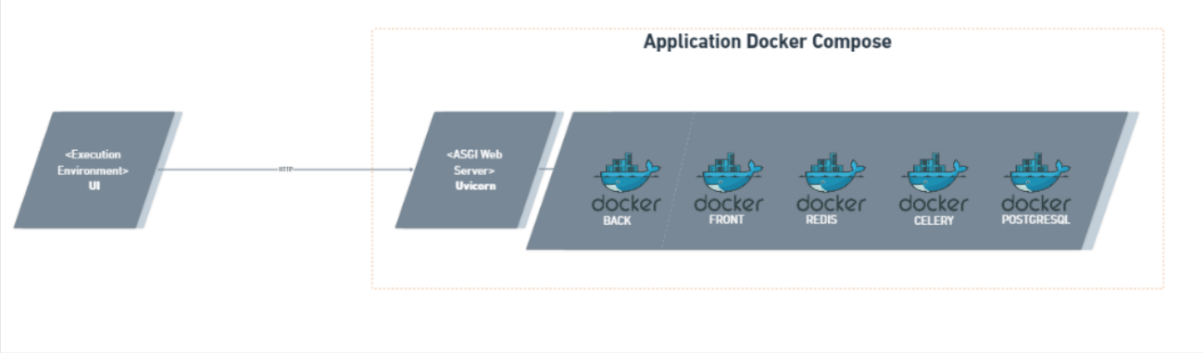

Ilustración 4. Diagrama de despliegue

Adicionalmente se incluyó el servicio de Uvicorn para la creación de un servidor web ASGI con Python, toda vez que aún cuando dicho servicio se encuentra incluido directamente en el backend, resulta pertinente su visualización para comprender los diversos servicios utilizados en el despliegue final de la aplicación.

Conclusiones

La solución propuesta para la conversión de archivos a PDF ha demostrado ser eficiente tanto en términos de procesamiento de cómputo como en la gestión de la concurrencia de tareas.

Esto se debe principalmente a la integración de Celery, un servicio de procesamiento por batch asíncrono, que optimiza la ejecución de las conversiones sin comprometer el rendimiento del sistema en tiempo real.

El uso de tecnologías modernas y frameworks como FastAPI, React.js, y la orquestación con Celery y Redis, no solo permite una respuesta rápida y eficiente a las solicitudes de los usuarios, sino que también facilita una experiencia de usuario fluida y segura, evidenciada por la implementación de autenticación mediante JSON Web Tokens (JWT).

Además, se ha identificado la oportunidad de ampliar la robustez y escalabilidad de la solución mediante la migración o integración de servicios en la nube. Esto no solo permitiría manejar un volumen mayor de usuarios y tareas de manera más eficiente, sino que también mejoraría la sostenibilidad del sistema al aprovechar la elasticidad y los recursos que ofrecen las plataformas en la nube.

En conclusión, mientras que la arquitectura actual satisface los requerimientos iniciales y demuestra una operación eficaz para la conversión de archivos, la adopción de infraestructuras en la nube representa un paso adelante natural. Esto no solo aseguraría una mayor escalabilidad y adaptabilidad frente a la demanda fluctuante, sino que también proporcionaría una base sólida para la introducción de nuevas características y la expansión de la aplicación en el futuro.