PARTE 5 – RESOLUCIÓN DE PROBLEMAS & DECISION MAKING

Objetivo:
Evaluar capacidad de liderazgo técnico, resolución de ambigüedad, análisis de tradeoffs, y diseño de sistemas escalables.

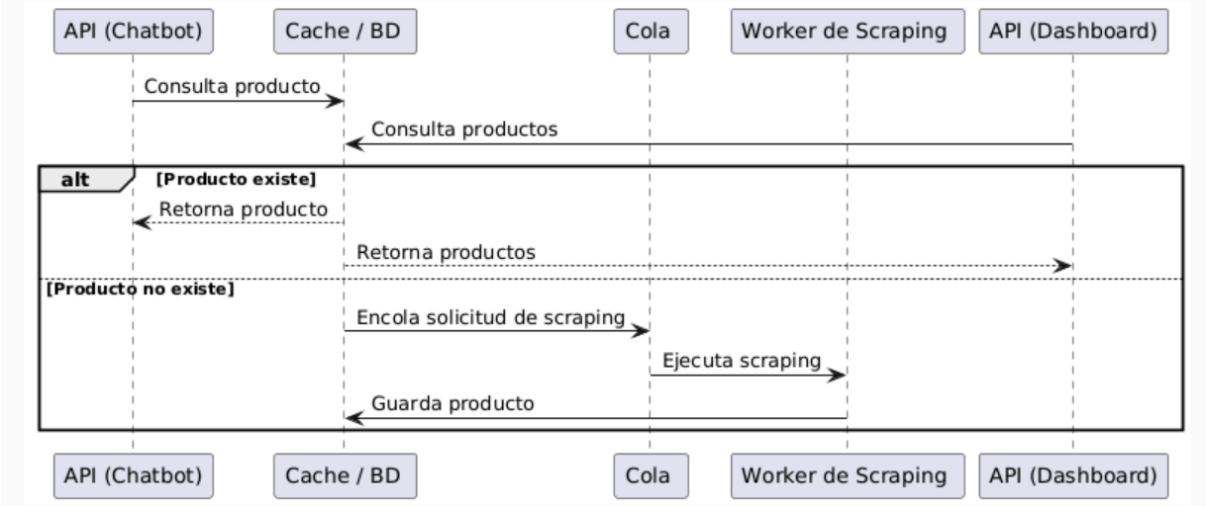
Ramiro Bastar

Caso 1: Integración multiplataforma

Tu equipo necesita integrar un sistema de scraping de productos que alimente tanto un chatbot (para consultas de clientes) como un dashboard (para analistas de negocio). El cliente quiere respuestas en tiempo real, pero el scraping toma entre 5 y 10 segundos por producto nuevo. ¿Cómo diseñarías esta arquitectura?
Entrega esperada:

- Diagrama del sistema propuesto.
- Estrategia de asincronía, colas o almacenamiento intermedio.
- Justificación técnica

Diagrama del sistema propuesto



Puntualmente describiremos el diagrama, en el siguiente apartado se hará la explicación a detalle.
1.- El Chatbot consulta el producto en el cache/bd, así como el Dashboard también lo puede hacer.
2.- En caso de que no se encuentre el producto, este se envía a una cola, para que un worker en paralelo realice el websraping
3.- Finalmente, se guarda nuevamente el producto en el Caché/DB para ser consultado

Estrategia de asincronía, colas o almacenamiento intermedio.

Primero haremos que el chatbot consulte una base intermedia Redis o SQL, y si el producto ya fue scrapeado, respondemos inmediatamente.

En caso de no existir el producto, la api coloca una tarea en una cola en paralelo, usando alguna tecnología que lo permita, por ejemplo RabbitMQ, AWS SQS o alguna otra.

Se realiza el scrapping de este producto de la cola, y se le informa al usuario que se está cargando su información.

Una vez finalizada esta actividad, no hay que olvidar guardar en la base de datos el resultado obtenido, además, es importante tenerla en caché por si es un producto de frecuente consumo en este periodo de tiempo.

Justificación técnica

Con esta arquitectura logramos una solución práctica y eficiente. Los productos que ya están en la base de datos se pueden responder casi al instante, y para los que aún no existen, los workers procesan las tareas en segundo plano sin bloquear al sistema. Esto evita cuellos de botella y mantiene el flujo ligero. Además, al guardar en cache los productos más consultados, tanto el chatbot como el dashboard pueden acceder rápidamente a la información, lo que asegura una buena experiencia tanto para los usuarios como para los analistas.

Caso 2: Fallo del OCR en producción

Después de lanzar el OCR en producción, se reportan errores en más del 35% de los recibos escaneados (mal reconocimiento de fechas y totales). ¿Cómo abordarías esta situación como líder del área?
Entrega esperada:

- Diagnóstico probable (modelo, input, UX).
- Solución técnica propuesta.
- Priorización de tareas y comunicación con stakeholders.

Diagnóstico probable

Hay varios puntos que pueden suceder, enumeraré algunos:

Problemas en los datos de entrada:

- Imágenes borrosas, mal escaneadas o inclinadas.
- Formatos de recibos muy variados: (estructura, tipografía, posición de los campos).
- Fotos tomadas con baja luz, sombras o ilegíbles.

Problemas en el procesamiento OCR

- Tesseract sin ajustes personalizados (psm, oem, idioma, etc.).
- El modelo OCR intenta procesar todo el recibo como un bloque, sin detectar primero las regiones clave (como bounding boxes para fecha y total).
- No se están extrayendo correctamente los datos

Problemas en la capa UX o de lógica

- El sistema interpreta mal lo que sí fue bien leído, por ejemplo, convertir una fecha "17/09/1991" en un formato incorrecto.
- No hay validación de resultados antes de mostrarlos.
- No se muestra una advertencia al usuario cuando el OCR no es confiable.

Solución técnica propuesta

Primero, agregaría validaciones simples, como revisar que la fecha sea válida y no futura, y que el total sea numérico y mayor a cero. Si no cumplen, se marcan como “requiere revisión”.

Después, mejoraría el preprocesamiento con OpenCV y aplicaría OCR solo en zonas específicas como fecha y total, en lugar de procesar todo el recibo.

Priorización de tareas y comunicación con stakeholders

Como encargado, lo más importante es la transparencia en el área, informando a los involucrados lo que está sucediendo, y a la par, ya estar resolviendo los problemas.

A continuación muestro una propuesta de priorización de actividades

Prioridad	Acción
Urgente	Informar a los involucrados internos en el uso del sistema.
Alta	Verificar los logs del sistema, posiblemente algo esté sucediendo
Alta	Verificar capa de entrada, posiblemente los usuarios están cargando la información de manera erronea, imágenes borrosas o sin información, etc
Alta	Verificar la UX, posiblemente existe ambigüedad o errores de validación
Alta	Validación de datos extraídos
Alta	Verificación de la lógica de los datos
Alta	Entregar una solución rápida si es posible
Media	En caso de que los datos extraídos estén de manera correcta, se debe evaluar el reentrenamiento del modelo
Media	Reentrenamiento del modelo con la misma tecnología
Media	Revisar métricas del nuevo modelo y determinar si sube a productivo
Baja/Opcional	Evaluar otras técnicas de entrenamiento de modelo

Caso 3: Chatbot con datos erróneos

El chatbot devuelve información desactualizada o incorrecta sobre compradores frecuentes. El modelo NLP funciona, pero las respuestas están mal. ¿Cómo organizarías el diagnóstico y corrección?

Entrega esperada:

- Hipótesis de errores (fuente de datos, cacheo, entrenamiento).
- Estrategia para testing, logging y validación.
- Propuesta para monitoreo y mejora continua

Hipótesis de errores

Aunque el modelo NLP interpreta bien la intención del usuario, la respuesta final contiene datos erróneos o desactualizados. Esto sugiere que el problema ocurre después de la detección de intención, posiblemente en la capa de datos, cacheo o integración lógica. Estas son las hipótesis más probables:

Problemas en los datos

Datos desactualizados: Es posible que la base de datos no se esté actualizando correctamente. Por ejemplo, si los compradores frecuentes cambian día a día, pero el sistema solo se actualiza una vez por semana, la información entregada estará atrasada.

Lógica incorrecta para calcular “frecuente”: Tal vez lo que el backend considera como "frecuente" no coincide con lo que se espera. ¿Se define por número de compras, por monto total o por actividad en los últimos meses? Esta ambigüedad puede llevar a resultados incorrectos.

Errores en la consulta: La consulta que devuelve a los compradores frecuentes podría estar mal filtrada o agrupada. Por ejemplo, podría estar mostrando resultados de todos los años en lugar de enfocarse en el período más reciente.

Problemas con el cache

Respuestas guardadas demasiado tiempo: Si se utiliza Redis u otro sistema de cache para acelerar respuestas, puede que se estén entregando datos que ya no coinciden con la base actualizada.

Caché mal configurado: Es posible que todas las consultas estén compartiendo la misma clave de caché, sin tener en cuenta variables como el usuario, la sucursal o el período consultado.

Falta de invalidación: Si no se elimina el cache cuando cambian los datos de origen, el sistema seguirá sirviendo información antigua.

Entrenamiento

Posiblemente existan ejemplos mal etiquetados, o confusos, donde a pesar de estar bien clasificadas las intenciones, las respuestas no son las correctas o ya se encuentran desactualizadas.
Otra posible situación sería que el modelo se haya entrenado con un dato coincidente con la pregunta requerida. Debemos tomar en cuenta que el reentrenamiento debe hacerse periódicamente a mediano o largo plazo, según lo determine el encargado de área.

Estrategia para testing, logging y validación

Logging: entender qué sucede internamente

Antes de hacer pruebas, es fundamental entender el comportamiento del sistema. Para ello, se deben registrar logs detallados que incluyan:

- Pregunta original del usuario.
- Intención detectada por el modelo NLP.
- Función o endpoint llamado en el backend.
- Parámetros utilizados (fechas, filtros, IDs).
- Fuente de los datos (cache vs. base de datos).
- Respuesta final entregada.

Con esto, es más fácil aislar errores y entender cómo fluye la información desde que se recibe la pregunta hasta que se responde.

Testing: comprobar que la lógica hace lo que debe

Tests unitarios

Se prueban funciones específicas del backend, como la que calcula compradores frecuentes, usando datos de prueba controlados.

Tests de integración

Se simulan preguntas reales al chatbot y se compara la respuesta con los resultados esperados directamente desde la base de datos.
También se valida que los datos obtenidos estén correctamente formateados en lenguaje natural y ordenados según el criterio esperado.

Tests automatizados diarios

Se programan scripts que ejecutan preguntas clave cada día.
Las respuestas del chatbot se comparan con los datos en vivo o una fuente validada.
Todo se registra en logs para análisis posterior.

Validación: asegurar que la respuesta sea coherente

Comparar las respuestas del chatbot con una fuente validada (dashboard o consulta directa a la base de datos). Esto incluye validar que los datos estén actualizados, bien formateados y en el orden esperado.

Propuesta para monitoreo y mejora continua

Monitoreo automatizado

Como se mencionó anteriormente, se propone una comparación automática de respuestas: preguntas clave se comparan con consultas reales a la base de datos.
Algo adicional, serían alertas por comportamientos atípicos

Feedback y aprendizaje del sistema

Una revisión manual, de parte de el departamento de QA quien realizará análisis internos y pueden marcar respuestas incorrectas para análisis posterior.
Verificar constantemente la lógica de negocio si sigue funcionando de la misma forma.
Finalmente, y nunca como primera instancia, un nuevo entrenamiento del modelo, con base en el feedback

Pruebas programadas y control de calidad

Ejecución de jobs con preguntas clave diariamente,validación tanto de precisión de datos como de claridad en la respuesta, haciendo comparativa histórica.