

E. Guía algoritmo interfaz página web

Algoritmo utilizado en Visual Studio y archivos obtenidos.

INTRODUCCIÓN

El avance de los sistemas de comunicación digital aplicados al ámbito de la salud ha generado la necesidad de desarrollar interfaces que permitan a los usuarios interactuar con tecnologías complejas de forma sencilla, confiable y transparente. En este sentido, el presente anexo describe el proceso de diseño e implementación de una interfaz web orientada a facilitar la comunicación remota entre pacientes y profesionales de la salud, priorizando la experiencia de usuario por encima de los detalles técnicos del sistema de transmisión.

El objetivo principal de esta etapa del trabajo fue construir una capa de interacción clara e intuitiva, capaz de actuar como puente entre el usuario final y los mecanismos internos de procesamiento y transmisión de información. Para ello, se adoptó un enfoque centrado en el usuario, apoyado en el uso de un servicio web ligero y eficiente que permitiera gestionar solicitudes.

DESARROLLO

El uso de FastAPI como núcleo del sistema respondió a varios criterios fundamentales. En primer lugar, se priorizó la claridad en la definición de las rutas de interacción, de manera que cada acción realizada por el usuario, como el envío de información, la consulta de resultados o la confirmación de recepción, correspondiera a un flujo lógico y fácilmente comprensible. Cada ruta fue diseñada para cumplir una única función, evitando comportamientos ambiguos y reduciendo la posibilidad de errores en la interacción.

Los algoritmos responsables del procesamiento interno, la comunicación con el sistema de transmisión y la gestión de sesiones se ejecutan de manera completamente transparente para el usuario. Desde la perspectiva de la interfaz, el usuario únicamente interactúa con campos de entrada claramente identificados y botones que representan acciones concretas, como enviar o actualizar información.

La interfaz fue diseñada para minimizar la cantidad de decisiones que el usuario debe tomar. Este enfoque, conocido como reducción de carga cognitiva, se tradujo en una estructura visual simple, donde cada elemento tiene un propósito evidente. Se evitó deliberadamente la inclusión de opciones avanzadas o configuraciones técnicas, ya que estas no aportan valor directo al usuario final y podrían generar confusión, especialmente en contextos médicos donde el tiempo y la claridad son factores críticos.

En cuanto al flujo de interacción paciente–doctor, se diseñaron algoritmos que gestionan la información de forma ordenada y secuencial. El paciente ingresa los datos solicitados a través de la interfaz, los cuales son validados automáticamente antes de ser procesados. Esta validación temprana permite detectar inconsistencias o campos incompletos sin requerir intervención del profesional de la salud, mejorando la eficiencia del proceso y reduciendo errores humanos.

Para habilitar la comunicación entre usuarios ubicados en diferentes redes, se incorporó un mecanismo de exposición segura del servicio web mediante túneles de red. Esta solución permitió que el sistema, aun ejecutándose en un entorno local, sea accesible desde internet a través de una dirección única. Desde el punto de vista del usuario, esta complejidad queda completamente oculta: tanto el paciente como el profesional acceden al sistema mediante un enlace simple, sin necesidad de configuraciones adicionales.

El uso de estos túneles fue cuidadosamente integrado con la lógica del servicio web. Los algoritmos de gestión de conexión se encargan de mantener activa la comunicación, reenviar solicitudes y garantizar que las respuestas lleguen al usuario correspondiente. Este enfoque permitió crear una experiencia de uso continua, en la cual la distancia física entre paciente y doctor no afecta la percepción de funcionamiento del sistema.

Otro aspecto relevante fue la implementación de mecanismos de retroalimentación clara. Cada acción del usuario genera una respuesta visible e inmediata, informando sobre el estado del proceso. Mensajes de confirmación, notificaciones de espera y avisos de recepción fueron diseñados con un lenguaje simple y directo, evitando tecnicismos. Esta retroalimentación constante contribuye a generar confianza en el sistema, un factor especialmente importante en aplicaciones relacionadas con la salud.

A continuación, se explica el funcionamiento del algoritmo implementado para esta sección, se puede revisar el siguiente enlace: (Rodríguez y Díaz, 13)

1. app.py: API REST y controlador de ejecuciones

app.py expone una pequeña API web que permite a los usuarios interactuar con el sistema desde un navegador. Sus funciones principales son: recibir archivos o texto, validar y almacenar la entrada, registrar la ejecución en la base de datos y lanzar el proceso de transmisión en segundo plano. También ofrece endpoints para listar transmisiones, consultar estado y borrar todo.

¿Cómo funciona internamente?:

- Al recibir una petición de subida, primero valida que el formato y el tamaño sean aceptables. Esto evita que archivos dañados o demasiado grandes saturen el sistema.

- Genera un identificador único para la ejecución (run_id) y guarda una copia del archivo en una carpeta de uploads para auditoría.
- Copia el contenido en una ruta específica que el flujo de transmisión (GNU Radio) revisa: esa copia es la “puerta” que comunica el mundo web con el flujo de radio.
- Registra la ejecución en la base de datos con estado inicial "QUEUED", lo que permite al interfaz mostrar una lista históricamente ordenada de transmisiones.
- Usa tareas en segundo plano para ejecutar el algoritmo de transmisión. Mientras el proceso corre, la API actualiza el estado a "RUNNING" y, al terminar, a "DONE" o "FAILED", permitiendo al usuario hacer consultas periódicas y ver logs parciales del proceso.

Decisiones de diseño centradas en el usuario:

- Exponer /uploads y recursos estáticos facilita que el usuario verifique el archivo subido sin acceder al servidor por consola.
- Registrar y devolver metadatos (original_filename, user_role, preview de texto) mejora la trazabilidad y la experiencia visual en la interfaz: el usuario no queda en la incertidumbre tras subir un archivo.
- Mantener la tarea de ejecución en segundo plano evita que la interfaz se “congele” y permite que el servidor web responda rápidamente.

2. db.py: Persistencia ligera para trazabilidad

La base de datos es simple pero crítica: almacena cada ejecución con su id, tipo (imagen/texto), rol del usuario, ruta de archivo, estado y rutas de log. Esto permite auditar, listar ejecuciones y mostrar información histórica en la interfaz.

Como prioridades se buscó tener consistencia de registro (timestamp en UTC), operación transaccional para evitar registros incompletos y compatibilidad sin depender de un servidor adicional.

Comportamiento relevante para el uso:

- Las funciones `create_run`, `update_status` y `get_run` proveen la API necesaria para que `app.py` mantenga la trazabilidad sin duplicar lógica de acceso a datos.

3. main.py: Punto de arranque robusto

Facilita el arranque del servicio en diferentes entornos y asegura que la aplicación (FastAPI) se importe de forma resiliente. Incluye mensajes de diagnóstico para el desarrollador y permite ejecutar con `uvicorn backend.main:app --reload`.

¿Por qué importa?

Documenta cómo iniciar la interfaz y reduce fricción al replicar o desplegar el sistema en máquinas con diferentes estructuras de paquete.

4. services/runner.py: Puente y controlador del algoritmo GNU Radio

Este módulo es responsable de lanzar la ejecución del flujo de transmisión (GNU Radio), capturar su salida en un archivo de log y devolver un código de resultado para actualizar el estado en la base de datos.

¿Cómo se integra con la experiencia del usuario?:

- El usuario sube el archivo y recibe un `run_id`. La API encola y delega al runner.

- El runner ejecuta el flujo con los parámetros correctos y redirige stdout (salida de datos) y stderr (salida de errores) a un log que el usuario puede consultar desde la interfaz (por medio de `get_run_status`).
- El runner controla tiempos, puede aplicar un timeout, y retorna si la ejecución fue correcta o si falló.

Separar el lanzador del servidor web evita que errores del flujo GNU Radio impacten la disponibilidad de la API. Además, permite instrumentar métricas (tiempo de ejecución, errores) sin mezclar responsabilidades.

5. `core/config.py`: Contrato entre la web y el flujo de transmisión

Define constantes como las rutas compartidas (`UPLOAD_DIR`, `IMAGE_INPUT_PATH`, `TEXT_INPUT_PATH`) y parámetros de validación (`ALLOWED_MIME`, límites de tamaño). Este archivo define el *contrato* entre la interfaz web y el flujo GNU Radio: ambos confían en las mismas rutas y formatos.

Importancia:

Documentar y centralizar estas variables facilita mover el sistema a otro servidor (basta actualizar las rutas), y es la primera referencia para el evaluador que quiera entender cómo se comunican los procesos.

RESULTADOS Y HALLAZGOS

La implementación de la interfaz basada en servicios web permitió demostrar que es posible encapsular procesos complejos de comunicación y transmisión dentro de una experiencia

de usuario sencilla y accesible. Durante las pruebas realizadas, se observó que los usuarios podían interactuar con la interfaz gracias al manual de usuario, ya que enviar información y recibir respuestas del sistema es un proceso intuitivo.

Al eliminar cualquier referencia a configuraciones técnicas, se logró que la comunicación remota se percibiera como un proceso natural y confiable. Asimismo, el uso de FastAPI facilitó la escalabilidad del sistema, permitiendo que nuevos flujos de interacción puedan incorporarse sin afectar la experiencia del usuario.

CONCLUSIONES Y RECOMENDACIONES

El trabajo desarrollado evidencia que el éxito de un sistema tecnológico en el ámbito de la salud no depende únicamente de su capacidad técnica, sino de la forma en que esta capacidad es presentada al usuario final. El diseño de una interfaz intuitiva, apoyada en algoritmos bien estructurados y en principios de usabilidad, permitió crear un sistema accesible tanto para pacientes como para profesionales de la salud.

La elección de un servicio web ligero y eficiente, junto con el uso de túneles de comunicación remota, permite eliminar barreras geográficas y técnicas, fortaleciendo el potencial del sistema para aplicaciones reales de telemedicina. Como recomendación futura, se sugiere profundizar en estudios de experiencia de usuario con población clínica real, así como integrar mecanismos adicionales de autenticación y trazabilidad que refuercen la seguridad y la confiabilidad del sistema en entornos de uso continuo.