

SEP

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TIJUANA  
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA



**INFORME TÉCNICO DE RESIDENCIA PROFESIONAL**

"SISTEMA DE ENTRENAMIENTO AUTOMÁTICO PARA CLASIFICADOR DE  
AUDIO CARDÍACO"

PRESENTA:

AMANDA CAROLINA ALVARADO OCHOA  
20211954

**BAJO LA ASESORÍA:**

**INTERNA:** MC. FORTUNATO RAMÍREZ ARZATE

**EXTERNA:** DR. ANGEL HUMBERTO CORRAL DOMINGUEZ

TIJUANA, BC.

MAYO 2025

## **Agradecimientos**

Agradezco profundamente al Instituto Tecnológico de Tijuana por brindarme la formación académica y las herramientas necesarias para el desarrollo de este proyecto. Extiendo mi sincero agradecimiento al M.C. Fortunato Ramírez Arzate, asesor externo, por su guía técnica, compromiso y constante apoyo durante el desarrollo de esta residencia profesional.

De igual manera, agradezco al personal docente y administrativo del Departamento de Ingeniería Eléctrica y Electrónica por su orientación y disposición a lo largo de mi formación. A mi familia, por su apoyo incondicional y confianza en cada etapa de mi trayectoria académica. Finalmente, reconozco el valor de todas las personas que, directa o indirectamente, contribuyeron al logro de este trabajo.

# Resumen

Este documento presenta el desarrollo de una plataforma integral para la automatización del reentrenamiento de clasificadores de audio cardíaco, dirigida a profesionales de la salud sin conocimientos técnicos especializados. El proyecto fue realizado en el Instituto Tecnológico de Tijuana como parte de la Residencia Profesional para el título de Ingeniería Biomédica.

La solución propuesta facilita la carga y procesamiento de señales de audio cardíaco mediante una interfaz intuitiva y un *backend* robusto que automatiza el flujo de entrenamiento del modelo. Esto elimina la necesidad de intervención manual en la ejecución de *scripts* y la gestión de modelos, mejorando la eficiencia, reduciendo errores y disminuyendo los tiempos de espera asociados al proceso.

Además, se implementaron mecanismos de seguridad para proteger la información sensible y asegurar la integridad de los datos. La plataforma contribuye a optimizar el diagnóstico cardíaco, ampliando el acceso a herramientas avanzadas en entornos clínicos y regiones con recursos limitados.

Este trabajo sienta las bases para futuras mejoras que incluyen la integración de capacidades en la nube y la incorporación de análisis biomédicos más detallados, con el objetivo de fortalecer la precisión y usabilidad del sistema en contextos reales.

**Palabras clave:** Auscultación cardíaca; Fonocardiograma; Inteligencia artificial; Interfaz de usuario; Telemedicina.

---

# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Descripción de la Institución . . . . .	2
1.2. Antecedentes . . . . .	3
1.3. Planteamiento del problema . . . . .	5
1.4. Objetivos . . . . .	6
1.4.1. Objetivo general . . . . .	6
1.4.2. Objetivos específicos . . . . .	6
1.5. Justificación . . . . .	7
<b>2. Marco teórico</b>	<b>9</b>
2.1. Fisiología del corazón . . . . .	9
2.1.1. Ciclo cardíaco . . . . .	10
2.2. Auscultación cardíaca . . . . .	11
2.2.1. Ruidos cardíacos . . . . .	11
2.2.2. Otros sonidos cardíacos . . . . .	13
2.2.3. Soplos . . . . .	13
2.3. Inteligencia artificial . . . . .	15
2.3.1. Aprendizaje automático . . . . .	15
2.3.2. Algoritmos de aprendizaje automático . . . . .	16
2.3.3. Optimización automática con TPOT . . . . .	17
2.3.3.1. Programación Genética en TPOT . . . . .	18
2.3.3.2. TPOTClassifier . . . . .	19
2.3.3.3. TPOTRegressor . . . . .	20
2.4. Tecnologías Web para el Desarrollo del Sistema . . . . .	20
2.4.1. Visual Studio Code . . . . .	20
2.4.2. HTML . . . . .	21
2.4.3. CSS . . . . .	23

2.4.4. JavaScript . . . . .	23
2.4.5. Node.js y su ecosistema . . . . .	24
2.4.6. MySQL . . . . .	24
<b>3. Desarrollo y resultados</b>	<b>26</b>
3.1. Backend . . . . .	26
3.2. Frontend . . . . .	33
3.3. Base de Datos . . . . .	46
<b>4. Conclusiones</b>	<b>50</b>
4.1. Trabajo a futuro . . . . .	51
<b>Bibliografía</b>	<b>52</b>

---

# Índice de figuras

1.1.	Contextualización del problema . . . . .	6
2.1.	Anatomía del interior del corazón. . . . .	9
2.2.	Ciclo cardíaco . . . . .	10
2.3.	Aparición de los ruidos dentro de un período cardíaco. . . . .	12
2.4.	Partes del flujo de trabajo del aprendizaje automático automatizado por TPOT	18
2.5.	Proceso de una petición web. . . . .	22
3.1.	Estructura general de los archivos de la aplicación . . . . .	33
3.2.	Vista de la barra de navegación en línea . . . . .	34
3.3.	Inicio de sesión . . . . .	36
3.4.	Ventana para crear una cuenta . . . . .	37
3.5.	Inicio de la página principal . . . . .	37
3.6.	Vista de la sección “Sobre nosotros” de la plataforma HeartForge. . . . .	38
3.7.	Descripción del servicio y botón para cargar archivos de audio en la página principal . . . . .	38
3.8.	Formulario para subir archivos de audio y reunir datos del paciente . . . . .	43
3.9.	Opciones disponibles para seleccionar el sexo del paciente en el formulario. .	44
3.10.	Despliegue de opciones para indicar el tipo de sonido cardíaco correspondiente.	44
3.11.	Mensaje de advertencia al dejar un campo obligatorio sin completar en el formulario. . . . .	44
3.12.	Reproducción y visualización del audio cardiaco. . . . .	45
3.13.	Popup de éxito tras la subida del audio. . . . .	46

---

# Índice de tablas

2.1. Clasificación de los soplos cardíacos según su intensidad . . . . .	14
3.1. Estructura de la tabla <code>usuarios</code> . . . . .	47
3.2. Estructura de la tabla <code>audios</code> . . . . .	48

---

# Índice de Códigos

1.	Etiquetado de un campo de formulario en HTML . . . . .	22
2.	Estructura de una regla CSS . . . . .	23
3.	Instalación de dependencias desde la terminal de Visual Studio Code. . . . .	26
4.	Importación de módulos necesarios para el funcionamiento del servidor. . . . .	27
5.	Configuración de <i>middleware</i> para procesar solicitudes entrantes. . . . .	28
6.	Esquema de validación de datos de registro con <code>Joi</code> . . . . .	29
7.	Configuración de almacenamiento de archivos con Multer. . . . .	29
8.	Conexión a la base de datos MySQL . . . . .	30
9.	Ruta para subir archivos . . . . .	32
10.	Inicialización del servidor . . . . .	32
11.	Barra de navegación . . . . .	34
12.	Inicio del contenedor de formularios. . . . .	34
13.	Formulario HTML de inicio de sesión. . . . .	35
14.	Formulario HTML de registro. . . . .	36
15.	Inicialización del DOM y selección de elementos . . . . .	39
16.	Manejo de eventos para la visualización del formulario emergente . . . . .	40
17.	Validación del formulario de registro en el cliente . . . . .	41
18.	Validación del formulario de registro en el cliente . . . . .	41
19.	Gestión del estado de sesión y almacenamiento local tras inicio exitoso . . . . .	42
20.	Autocompletar correo si está guardado en <code>localStorage</code> . . . . .	42
21.	Inicialización del reproductor y llamada a la visualización. . . . .	45
22.	Comando para crear la base de datos . . . . .	47
23.	Comando SQL para la creación de la tabla <code>usuarios</code> . . . . .	47
24.	Encriptación de contraseña con <code>bcrypt</code> durante el registro. . . . .	48
25.	Comparación de la contraseña ingresada con la almacenada. . . . .	49

---

# CAPÍTULO 1

---

## Introducción

México, como país de gran extensión territorial y geografía compleja, enfrenta serias dificultades para garantizar la conectividad y la adecuada comunicación entre sus distintas regiones. Estas condiciones han favorecido una marcada centralización de los servicios de salud, concentrando la mayoría de los recursos humanos, tecnológicos y de infraestructura en las principales zonas urbanas. Extender dichos servicios hacia regiones apartadas representa un reto logístico considerable, frecuentemente costoso, riesgoso y, en muchos casos, poco viable. A pesar de los esfuerzos institucionales por lograr una cobertura universal en salud, persisten rezagos significativos en cuanto a la accesibilidad y disponibilidad de atención médica en diversas regiones del país [1].

En este contexto, el presente proyecto surge como respuesta a la creciente necesidad de herramientas tecnológicas que optimicen la calidad, eficiencia y precisión de los diagnósticos a distancia, particularmente en el ámbito de la salud cardiovascular. El enfoque principal consiste en el desarrollo de un sistema basado en telemedicina, capaz de clasificar señales de audio cardíaco para emitir un diagnóstico acorde con la información contenida en dichas señales. La problemática a resolver radica en la necesidad de mantener actualizado el sistema de clasificación sin requerir la intervención constante de personal técnico especializado.

La implementación de una plataforma automatizada permitirá un proceso de reentrenamiento eficiente, autónomo y escalable, que aborde de manera integral las limitaciones actuales. Este avance no solo mejorará la precisión y oportunidad del diagnóstico remoto, sino que también representará una alternativa más accesible y adaptable a las necesidades reales de las instituciones de salud, particularmente en entornos con recursos limitados.

## **1.1. Descripción de la Institución**

El Tecnológico Nacional de México (TecNM) campus Tijuana, comúnmente conocido como Instituto Tecnológico de Tijuana (ITT), inició actividades el 17 de septiembre de 1971. Actualmente, el ITT representa la principal oferta de educación tecnológica en el estado de Baja California.

Su **misión** es:

“Contribuir a la formación integral de profesionistas e investigadores líderes en la innovación y el desarrollo tecnológico de la región, del país y del mundo; con alto sentido de responsabilidad social, a través de un servicio educativo de calidad, con equidad y pertinencia.”

Y su **visión**:

“Ser una institución rectora de la educación superior tecnológica, la investigación científica y el desarrollo tecnológico en la Región Noroeste del país, con proyección a nivel nacional e internacional.”

El ITT establece el compromiso de implementar todos sus procesos orientados hacia la satisfacción de sus usuarios, sustentados en la calidad del proceso educativo, mediante la eficacia de un Sistema de Gestión de la Calidad y de mejora continua, conforme a la norma ISO 9001:2008 / NMX-CC-9001-IMNC-2008.

Actualmente, la institución cuenta con una oferta educativa de 20 licenciaturas, 6 maestrías y 3 doctorados, distribuidos en sus dos planteles: Tomás Aquino y Otay [2].

Este proyecto de Residencia Profesional se desarrolla en el Departamento de Ingeniería Eléctrica y Electrónica del ITT, y sus resultados incidirán directamente en la continuidad del proyecto de investigación titulado: **“Plataforma de Diagnóstico Asistido por Computadora para un Sistema de Telemedicina de Auscultación Cardíaca Remota del TecNM”**, con número de asignación 20998.24-P, cuyo responsable es el M.C. Fortunato Ramírez Arzate, asignado como asesor interno del proyecto.

## 1.2. Antecedentes

La Organización Mundial de la Salud (OMS) define la telemedicina como “el suministro de servicios de atención sanitaria en los cuales la distancia constituye un factor crítico”. Esta definición resalta la relevancia del soporte tecnológico avanzado para superar las barreras impuestas por la separación geográfica entre los pacientes y los centros de atención médica. La infraestructura tecnológica facilita el intercambio de información entre participantes remotos durante un acto médico, con el objetivo principal de ofrecer servicios multimedia en red —como la transferencia de audio, video, imágenes, datos y texto— que permitan la prestación de atención médica a distancia. Para una auscultación digital adecuada, el equipamiento básico incluye un estetoscopio electrónico, auriculares, una computadora con capacidad de procesamiento de señal y un software especializado para el registro y análisis de los sonidos cardíacos [3].

En México, la implementación de la telemedicina data de 1995, según la Secretaría de Salud Federal. Actualmente, este modelo ofrece servicios médicos virtuales a más de 3.3 millones de personas, a través de 606 centros de salud distribuidos en 21 entidades federativas, entre las que destacan el Estado de México, Querétaro, Puebla y la Ciudad de México [4].

La auscultación de señales mediante estetoscopios tradicionales y electrónicos enfrenta limitaciones técnicas importantes. No solo se registran los sonidos internos del cuerpo, sino también el ruido ambiental que puede interferir en el mismo rango de frecuencia. Además, esta técnica está condicionada por la variabilidad en los umbrales auditivos de los profesionales de la salud, así como por su experiencia para identificar patrones acústicos relevantes. En general, los resultados de una auscultación suelen expresarse de manera cualitativa, lo que los hace susceptibles a interpretaciones subjetivas, como lo señala la Revista Mexicana de Ingeniería Biomédica [5].

Un estudio publicado en la Revista de Investigaciones de la Universidad del Quindío analiza cómo los sistemas automatizados de diagnóstico han sido aplicados para la detección de soplos cardíacos. Estos sistemas suelen registrar señales fonocardiográficas segmentadas en intervalos breves, lo cual, si bien útil, puede omitir características propias de la dinámica cardíaca individual. Un caso representativo es el análisis de 1060 señales fonocardiográficas

obtenidas de los cuatro focos de auscultación en 144 pacientes, clasificadas como normales o patológicas y categorizadas según su ubicación en el ciclo cardíaco (soplos sistólicos, diastólicos o sistodiastólicos). En lugar de segmentar de forma convencional, estos registros fueron preprocesados para conservar la dinámica fisiológica individual, lo que permite una mejor precisión diagnóstica [6].

El desarrollo de estos sistemas de diagnóstico ha incorporado técnicas avanzadas de procesamiento de señales, tales como la Predicción Lineal Perceptiva y la obtención de coeficientes cepstrales mediante la Transformada Rápida de Fourier (FFT) y la Transformada de Fourier de Corta Duración (STFT). Asimismo, se han empleado clasificadores estocásticos como los Modelos Ocultos de Markov (HMM) y los Modelos de Mezcla Gaussiana (GMM), combinados con métodos de selección de características. Estas metodologías permiten capturar información significativa sobre la dinámica fisiológica del corazón, optimizando el entrenamiento del sistema y logrando tasas de clasificación efectivas para su aplicación en entornos clínicos [6].

Por su parte, la Revista Peruana de Investigación en Salud señala que los avances en inteligencia artificial (IA) han transformado múltiples sectores, incluido el de la salud. No obstante, su eficacia depende directamente de la calidad y diversidad de las bases de datos utilizadas para el entrenamiento de modelos. La existencia de datos representativos y libres de sesgos es esencial para garantizar decisiones precisas y justas por parte de los algoritmos [7].

Actualmente, uno de los mayores retos de la comunidad científica y tecnológica es obtener datos éticos, imparciales y representativos, respetando siempre la privacidad, la transparencia y los derechos de las personas. Asimismo, el crecimiento exponencial de las aplicaciones de IA requiere una colaboración constante entre investigadores, instituciones, empresas y gobiernos para facilitar el acceso y la compartición responsable de datos. Este enfoque permitirá un desarrollo más equitativo, ético y socialmente beneficioso de estas tecnologías [7].

### **1.3. Planteamiento del problema**

En México, la mayoría de las instituciones de salud —incluidos centros de salud, hospitalares y servicios especializados— se concentran en zonas urbanas, como consecuencia directa de la distribución poblacional. No obstante, las necesidades de atención médica en las regiones rurales siguen siendo apremiantes, particularmente debido a las limitaciones en infraestructura y servicios básicos. Según el último censo del Instituto Nacional de Estadística y Geografía (INEGI), la población nacional asciende a 123.5 millones de personas, de las cuales el 23.3 % reside en localidades rurales. Estas comunidades enfrentan una marcada desventaja en términos de acceso a servicios médicos oportunos y especializados [8].

En este contexto, las enfermedades cardiovasculares representan un desafío prioritario en materia de salud pública. De acuerdo con cifras preliminares del INEGI correspondientes al primer semestre de 2022, las enfermedades del corazón constituyeron la principal causa de muerte en el país, con un total de 105,864 defunciones registradas. Esta situación evidencia la necesidad urgente de contar con herramientas diagnósticas eficientes, accesibles y actualizadas, particularmente en regiones con acceso limitado a servicios especializados [9].

Actualmente, los sistemas de diagnóstico remoto mediante audio cardíaco enfrentan una limitación crítica: el proceso de reentrenamiento del clasificador requiere intervención constante y especializada. Dicha actualización se realiza de manera manual, mediante la ejecución de scripts y ajustes técnicos en servidores remotos, lo cual implica una alta dependencia de personal capacitado en programación. Este procedimiento no solo es propenso a errores, sino que también resulta lento, costoso y restrictivo en términos operativos.

La falta de automatización restringe la autonomía del personal médico, que no puede actualizar el clasificador por sí mismo debido a la complejidad técnica del proceso. Esta situación prolonga los tiempos de respuesta y afecta la capacidad del sistema para adaptarse con agilidad a nuevas condiciones clínicas, comprometiendo así la calidad del diagnóstico.

Para superar estas limitaciones, se propone el desarrollo de una plataforma que automate el proceso de reentrenamiento del clasificador. Esta solución integrará la ejecución automática de scripts, la gestión dinámica de modelos de clasificación y la actualización de datos en la nube, sin requerir intervención manual. De esta manera, los médicos podrán

actualizar el sistema de forma ágil e intuitiva, sin necesidad de conocimientos técnicos avanzados.

La implementación de esta plataforma reducirá significativamente el tiempo y el esfuerzo requeridos para actualizar los clasificadores, incrementando la eficiencia del sistema, minimizando la probabilidad de errores y promoviendo un uso más amplio y accesible de herramientas basadas en inteligencia artificial dentro del entorno clínico.

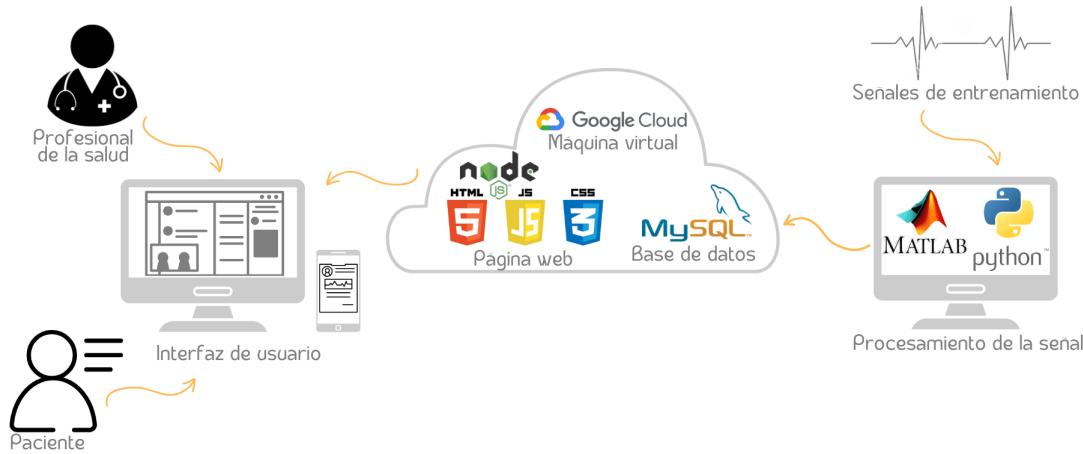


Figura 1.1. Contextualización del problema

## 1.4. Objetivos

### 1.4.1. Objetivo general

Desarrollar un sistema automatizado para el entrenamiento de un clasificador de audio cardiaco, que permita a profesionales de la salud re-entrenar el sistema sin requerir conocimientos técnicos especializados, incorporando una interfaz de usuario intuitiva para la carga de señales de entrenamiento.

### 1.4.2. Objetivos específicos

1. Establecer un flujo de trabajo secuencial y automático que le permita al sistema realizar el entrenamiento del clasificador sin intervención manual, asegurando la adecuación de cada etapa del proceso.

2. Implementar una interfaz de usuario que le permita a profesionales de la salud cargar señales de audio cardiaco y establecer los parámetros mínimos necesarios de reentrenamiento del clasificador de audio cardiaco.
3. Evaluar el desempeño del nuevo proceso de entrenamiento automático para garantizar su precisión, realizando pruebas y ajustes necesarios.

## 1.5. Justificación

El proyecto que se propone busca abordar una problemática crucial en el diagnóstico remoto de enfermedades cardiovasculares: la falta de automatización en el reentrenamiento de un clasificador de audio cardiaco. Esta deficiencia limita considerablemente la capacidad del sistema para adaptarse a nuevas condiciones clínicas, lo que incrementa los errores diagnósticos y alarga los tiempos de respuesta. La falta de actualización automática impide que el clasificador evolucione a la par con los avances en la práctica clínica, lo que podría afectar la precisión del diagnóstico y la eficacia de la atención. Así, desarrollar una solución automatizada no solo permitirá agilizar los procesos, sino también garantizar una mayor precisión diagnóstica y un servicio de salud más eficiente.

En el corto plazo, se espera que la solución automatizada reduzca el tiempo necesario para actualizar el clasificador, mejorando la capacidad de respuesta en entornos clínicos. En el largo plazo, este proyecto contribuirá al desarrollo de sistemas diagnósticos más robustos, escalables y accesibles, promoviendo la equidad en el acceso a la salud y fortaleciendo la atención primaria en comunidades rurales y marginadas.

La mejora en la precisión del diagnóstico remoto tiene un impacto directo en el bienestar de los pacientes y en la eficiencia operativa de las instituciones de salud. Reducir diagnósticos erróneos y optimizar los tiempos de atención repercutirá en una atención más ágil y de mayor calidad. Además, este proyecto garantiza que los sistemas diagnósticos se mantendrán actualizados con los datos más recientes, beneficiando a profesionales y pacientes por igual.

Los indicadores clave para medir el éxito incluyen la reducción del tiempo promedio de reentrenamiento en comparación con el sistema manual, la disminución en la tasa de errores diagnósticos y el aumento en la adopción por parte de los profesionales de la salud,

particularmente en zonas rurales.

Este proyecto también tiene el potencial de generar conocimiento único en la integración de tecnología de inteligencia artificial en el ámbito clínico. Servirá como referencia para futuros desarrollos en automatización de procesos médicos y su aplicación en entornos rurales y urbanos. Asimismo, puede inspirar innovaciones similares en otras áreas de la medicina.

---

# CAPÍTULO 2

---

## Marco teórico

En este capítulo se muestran los conceptos y herramientas necesarias para desarrollar y cumplir con los objetivos de este proyecto, se mencionan conceptos relacionados con el análisis y diagnóstico en cardiología así como aspectos fundamentales de tecnología informática para el desarrollo web.

### 2.1. Fisiología del corazón

El corazón es un órgano muscular de tamaño pequeño, que se encuentra situado en la parte inferior de la cara anterior del mediastino. Constituido por tres capas de tejido y subdividido internamente en cuatro cámaras, como se observa en la Figura 2., comunicadas entre sí por las válvulas cardíacas [10].

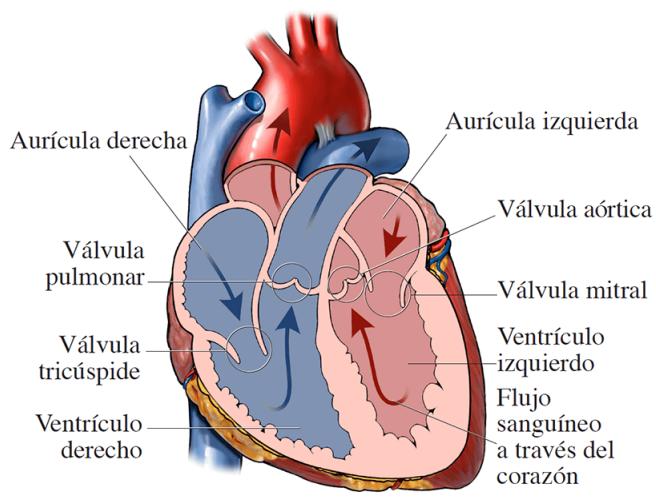


Figura 2.1. Anatomía del interior del corazón. Esta imagen muestra las cuatro cámaras del corazón y la dirección del flujo de sangre por el corazón. La sangre con bajo contenido de oxígeno, que se muestra en azul, fluye al corazón y es bombeada hacia los pulmones. Luego, la sangre con alto contenido de oxígeno, es bombeada hacia el resto del cuerpo, con la ayuda de las válvulas cardíacas.

### 2.1.1. Ciclo cardíaco

El corazón tiene una función de bomba, encargado de impulsar la sangre a todo el organismo. La circulación sanguínea es un proceso cíclico en el que la sangre oxigenada es impulsada desde el ventrículo izquierdo hacia todo el cuerpo y regresa, con menos oxígeno y desechos, al corazón por el lado derecho. Luego, el ventrículo derecho envía esta sangre hacia los pulmones para ser oxigenada nuevamente y reiniciar el ciclo.

El ciclo cardíaco se divide en dos fases principales: sístole (contracción y eyeción de sangre) y diástole (relajación y llenado ventricular). Como se observa en la Figura 3., cada fase tiene subdivisiones que implican cambios de presión y volumen en las cavidades cardíacas, así como la apertura y cierre de válvulas. Estos eventos generan ruidos cardíacos identificables mediante la auscultación (primer y segundo ruido). El texto también introduce conceptos funcionales esenciales como el gasto cardíaco, el volumen minuto y el volumen latido, fundamentales para comprender el funcionamiento cardiovascular[11].

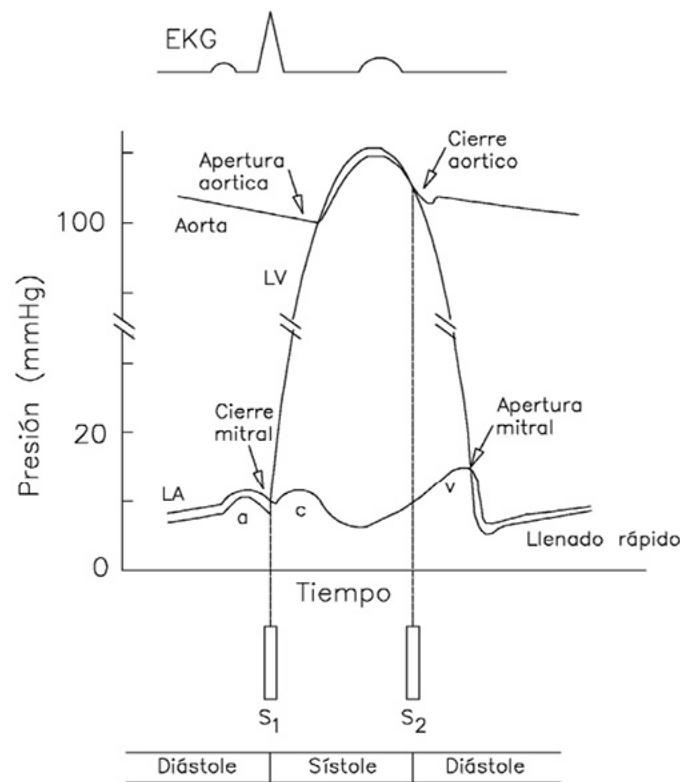


Figura 2.2. Ciclo cardíaco

La detección temprana de anomalías en el sistema cardiovascular es fundamental para establecer un diagnóstico oportuno e iniciar el tratamiento adecuado, lo cual resulta crucial para prevenir tanto la progresión de la enfermedad como el aumento de la morbimortalidad en los pacientes. Asimismo, resulta esencial no solo reconocer rápidamente un cuadro de insuficiencia cardíaca, sino también identificar su etiología y localizar con precisión el sitio del fallo cardíaco, ya que la implementación de un tratamiento específico es determinante para lograr una evolución clínica favorable[12].

## **2.2. Auscultación cardíaca**

Han pasado casi 200 años desde que Laënnec utilizó un cuaderno enrollado para auscultar el corazón de una paciente con manifestaciones de enfermedad cardíaca, con lo que creó el primer antecesor del estetoscopio, también correctamente llamado fonendoscopio [13].

La auscultación cardíaca es una técnica clínica fundamental que permite al médico escuchar los sonidos producidos por el corazón mediante el uso del estetoscopio, con el objetivo de identificar alteraciones funcionales o estructurales. A través de esta práctica, es posible detectar ruidos normales como los sonidos del ciclo cardíaco, así como sonidos anormales, entre los que se incluyen soplos, clics o variaciones en la intensidad de los tonos, los cuales pueden indicar la presencia de diversas enfermedades cardiovasculares[14].

No obstante, dominar la auscultación cardíaca representa un reto significativo para los profesionales de la salud. Se trata de una habilidad compleja que requiere tiempo, formación constante y exposición clínica directa a una amplia variedad de casos. Se estima que alcanzar un nivel adecuado de competencia puede llevar entre dos y tres años de práctica intensiva, bajo la supervisión de cardiólogos experimentados. Esta destreza, al igual que el aprendizaje de un instrumento musical, demanda no solo conocimiento teórico, sino también sensibilidad auditiva, capacidad de concentración y experiencia clínica acumulada [14].

### **2.2.1. Ruidos cardíacos**

Cuando el corazón funciona correctamente, las válvulas cardíacas abren y cierran de manera coordinada provocando dos sonidos principales conocidos como “lub-dub” o, primer

sonido cardíaco (S1), y segundo sonido cardíaco (S2) [15].

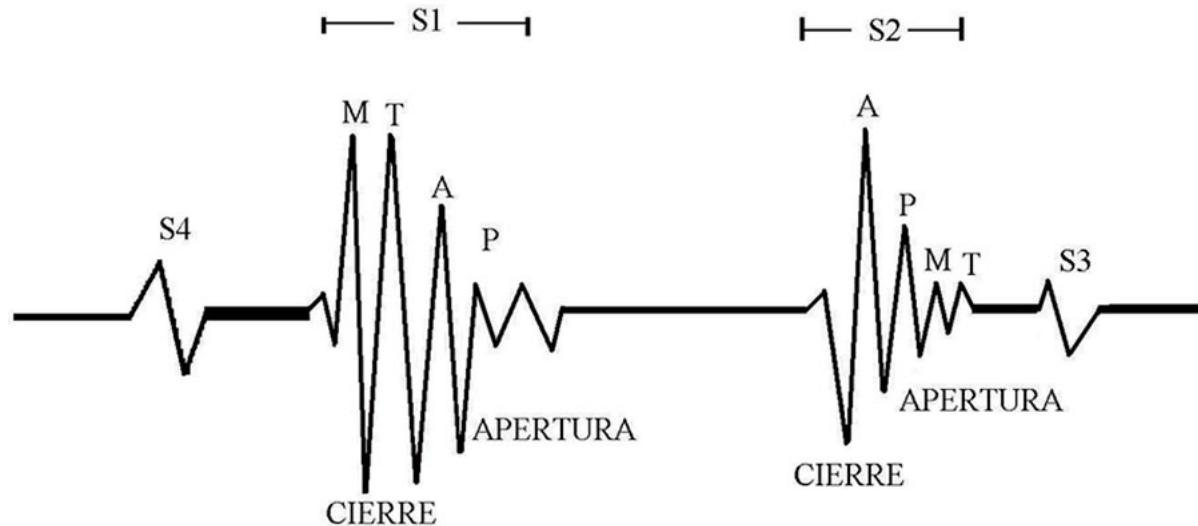


Figura 2.3. Aparición de los ruidos dentro de un período cardíaco.

Como se puede observar en la Figura 2.3, el primero lo generan principalmente el cierre de las válvulas atrio ventriculares, Mitral (M) y Tricúspide (T). Sin embargo, también ocurre la apertura de las válvulas semilunares. El segundo, lo generan principalmente el cierre de las válvulas semilunares, Pulmonar (P) y Aórtica (A); a su vez, las válvulas atrio ventriculares se abren, de tal manera que, en el simple “lub” ocurren 4 eventos: cierre de M y T, apertura de P y A. Mientras que en el “dub” ocurre lo opuesto. Todo esto sucede en un tiempo de 0.8 segundos, en un adulto joven sano, con una frecuencia cardíaca de 75 latidos por minuto [15] .

El tercer sonido cardíaco (S3) se produce inmediatamente después del S2, durante llenado ventricular rápido. Se puede oír en niños y adultos jóvenes sanos. También en pacientes con ventrículos dilatados y situaciones en las que el llenado ventricular rápido aumenta, como: cortocircuitos grandes, insuficiencia cardíaca congestiva, insuficiencia tricúspide o miocardiopatías [16].

El cuarto sonido cardíaco (S4): aparece en la diástole ventricular, justo antes del S1, coincidiendo con la sistole auricular. Es siempre patológico [16].

## 2.2.2. Otros sonidos cardíacos

Por otro lado, los sonidos cardíacos incluyen no solo estos ruidos fisiológicos, sino también una variedad de sonidos anormales o accesorios que pueden aparecer en presencia de enfermedades cardiovasculares. Ejemplos de estos sonidos patológicos incluyen el **ritmo de galope**, caracterizado por la audición de S3 y S4 en el contexto de insuficiencia cardíaca; el **chasquido de apertura**, indicativo de estenosis mitral o tricuspídea; el **clic sistólico** de eyección, asociado a estenosis valvular o dilatación de grandes vasos; el **chasquido mesosistólico**, típico del prolapsio mitral; y el **roce pericárdico**, que sugiere pericarditis y tiene una cualidad áspera que se percibe mejor en ciertas posiciones del paciente [16].

## 2.2.3. Soplos

Cuando las válvulas cardíacas presentan alteraciones estructurales o funcionales que impiden su apertura o cierre adecuado, se generan sonidos adicionales debido al flujo sanguíneo turbulento. Esta turbulencia, causada por el incremento de la presión y la obstrucción parcial del paso sanguíneo, produce vibraciones audibles conocidas como soplos cardíacos. La fisiopatología de este fenómeno puede compararse al efecto que se produce al comprimir parcialmente una manguera con agua en flujo continuo, generando ruidos irregulares por el paso forzado del líquido. Los soplos pueden manifestarse durante distintas fases del ciclo cardíaco y su intensidad, tonalidad y localización varían en función de la naturaleza y severidad del compromiso valvular [15].

La descripción de los soplos se realiza utilizando los siguientes criterios, que se explican a continuación: ubicación, calidad y momento.

- La localización se determina por el punto de máxima intensidad, generalmente asociado a los focos clásicos de auscultación: foco aórtico (segundo espacio intercostal derecho, junto al esternón), foco pulmonar (segundo espacio intercostal izquierdo), foco tricuspídeo (cuarto espacio intercostal izquierdo) y foco mitral (quinto espacio intercostal en la línea medioclavicular izquierda). Además, es importante considerar la irradiación del sopro: los de origen pulmonar pueden percibirse hacia la espalda, mientras que los aórticos tienden a irradiarse hacia el cuello [17].

- Los soplos se gradúan según una escala numérica descrita por Levine en 1933, que se describe en la Tabla 2.1. Cuando se refiere por escrito, se suele consignar en forma de fracción (1/6, 2/6, 3/6...). Recordemos que la calidad de la intensidad del soplo no se correlaciona con la gravedad de la lesión [16].

Tabla 2.1. Clasificación de los soplos cardíacos según su intensidad

Grado	Descripción
1	Audible solo con gran concentración y en circunstancias favorables.
2	Débil, pero audible con facilidad.
3	Fácil de oír, de intensidad intermedia.
4	Fácilmente audible y acompañado de un thrill o frémito (vibración palpable en la pared torácica).
5	Muy intenso, acompañado de frémito y audible con solo el borde del estetoscopio sobre la pared torácica.
6	Audible sin necesidad de apoyar el estetoscopio.

- El momento del soplo se refiere al punto del ciclo cardíaco en el que ocurre. Los soplos sistólicos se presentan entre el primer (S1) y el segundo (S2) ruido cardíaco, mientras que los diastólicos ocurren entre S2 y el siguiente S1. Estos pueden describirse con mayor precisión como tempranos, medios o tardíos, dependiendo del momento específico dentro de la sístole o diástole. Algunos soplos sistólicos abarcan toda la sístole y se denominan holosistólicos, mientras que los soplos continuos comienzan durante la sístole y se extienden más allá de S2, cubriendo parte de la diástole [17].
- El timbre se refiere a la calidad del sonido, determinada por la presencia de armónicos o sobretonos. Esta característica permite diferenciar los soplos no solo por su frecuencia, sino también por su tonalidad y textura. En términos de frecuencia, los soplos pueden ser de alta o baja frecuencia. Además, se pueden emplear otros términos descriptivos como soplante, áspero, musical, retumbante o chirriante para caracterizar mejor el tipo de sonido que emiten [17].

## 2.3. Inteligencia artificial

Entender qué es la inteligencia artificial y cómo se puede emular fue un gran acontecimiento que se debe en gran parte a la contribución de Alan Turing. Gracias a la Prueba de Turing, se logra tener una visión general de qué es la inteligencia. Turing definió el comportamiento inteligente como la capacidad de lograr un rendimiento a nivel humano en todas las tareas cognitivas, suficiente para engañar a un interrogador. El concepto formal de la IA fue posteriormente acuñado en 1956 durante el *Dartmouth Summer Research Project On Artificial Intelligence*, en el cual investigadores como John McCarthy, Marvin L. Minsky, Nathaniel Rochester y Claude E. Shannon, sentaron las bases para el desarrollo de máquinas capaces de simular características de la inteligencia humana [18].

Más recientemente, la evolución de la IA ha sido marcada por nuevas propuestas y avances. Russell y Norvig destacaron la importancia de desarrollar sistemas capaces de comportarse de manera inteligente, clasificándolos en categorías tales como aquellos que simulan el pensamiento y la acción humana, y los que piensan y actúan de manera puramente racional. Esta distinción ha sido clave en la expansión de la IA, que ahora abarca áreas como el procesamiento de lenguaje natural, la representación del conocimiento, el razonamiento automático, el aprendizaje automático, la visión computacional y la robótica. Estas áreas continúan siendo fundamentales en el avance de la IA moderna, que sigue evolucionando a medida que las tecnologías mejoran y los algoritmos se optimizan [18].

### 2.3.1. Aprendizaje automático

El paso del tiempo, dió como resultado especializaciones y un tipo particular de IA llamado Aprendizaje Automático (*machine learning* en inglés), el cual fue definido en 1959 por Arthur Samuel. Este enfoque se centra en dotar a los sistemas informáticos de la capacidad de aprender a partir de los datos, sin necesidad de ser programados explícitamente para cada tarea específica. Desde una perspectiva práctica, el aprendizaje automático puede definirse como la disciplina que permite a un programa mejorar su rendimiento en una tarea determinada conforme adquiere experiencia [19].

Esta rama ha experimentado un crecimiento acelerado impulsado por el aumento exponencial de datos disponibles —un fenómeno conocido como *Big Data*— y por el acceso global a través de Internet. Esta abundancia de información ha sido clave para entrenar modelos más precisos, complejos y adaptables. Como resultado, el aprendizaje automático se ha diversificado en distintas categorías, entre las que destacan el aprendizaje supervisado, no supervisado y por refuerzo, cada uno con sus propios métodos y aplicaciones según el tipo de problema a resolver [18].

### 2.3.2. Algoritmos de aprendizaje automático

Árboles de decisión (*Decision Trees*, DT) Introducidos por Ross Quinlan en 1986, los árboles de decisión permiten clasificar objetos mediante una estructura jerárquica de reglas. El modelo se construye a partir de un nodo raíz, que se ramifica en nodos internos y termina en nodos hoja, donde se asignan las clases. Cada decisión se basa en características (por ejemplo, propiedades espectrales), lo que permite una clasificación interpretable y directa [20].

Bosques aleatorios (*Random Forest*, RF) Propuesto por Leo Breiman en 2001, Random Forest es un método de ensamble que genera múltiples árboles de decisión sobre submuestras aleatorias del conjunto de entrenamiento y combina sus resultados para mejorar precisión y estabilidad. Es robusto frente a ruido, datos insuficientes o desbalanceados y puede integrar múltiples fuentes de información, como imágenes multiespectrales, hiperespectrales o datos geográficos. Requiere pocos parámetros: generalmente el número de árboles y el número de variables seleccionadas por división [21].

K vecinos más cercanos (*k-Nearest Neighbours*, K-NN) Introducido por Thomas Cover en 1967, K-NN es un clasificador basado en instancias que no requiere entrenamiento previo. Clasifica un dato desconocido en función de la similitud con sus vecinos más cercanos en el espacio de características. Es sencillo de implementar y eficaz con clases multimodales, aunque su rendimiento puede verse afectado por la dimensión del espacio [22].

Máquinas de soporte vectorial (*Support Vector Machines*, SVM) Desarrollado por Vapnik y Cortes en 1993, SVM es un algoritmo robusto y flexible que busca el hiperplano óptimo que maximiza la separación entre clases. Se basa en los vectores de soporte, los datos más

cercanos al margen de decisión. Su eficacia depende del tipo de kernel utilizado (lineal, radial o polinomial) y de la estrategia multiclas seleccionada (por ejemplo, “uno contra todos”). Ha demostrado gran rendimiento en clasificación de imágenes, especialmente en contextos con pocas muestras [20].

Redes Neuronales Artificiales (*Artificial Neural Networks*, ANN) Inspiradas en la estructura y funcionamiento del cerebro humano, las redes neuronales artificiales fueron propuestas inicialmente por Warren McCulloch y Walter Pitts en 1943. Son modelos no paramétricos, lo que significa que su desempeño depende directamente de la calidad del entrenamiento y de la distribución de los datos [23].

En el campo de la clasificación de imágenes satelitales o de sensores remotos, se utilizan arquitecturas como el perceptrón multicapa con retropropagación del error, mapas auto-organizados, redes de contrarretropropagación, redes de Hopfield y modelos basados en teoría de resonancia adaptativa. Estas redes se componen de unidades de procesamiento llamadas “neuronas”, conectadas entre sí mediante enlaces ponderados (pesos) que modifican la información que fluye entre capas.

Cada nodo realiza operaciones matemáticas básicas, y la combinación de estas permite obtener salidas complejas a partir de entradas estructuradas. El usuario puede definir distintos parámetros, como el número y tamaño de capas ocultas, la tasa de aprendizaje y valores de regularización. Aunque existen lineamientos generales, la parametrización óptima debe ajustarse según las características específicas de cada problema o conjunto de datos.

### 2.3.3. Optimización automática con TPOT

Durante el último año se ha desarrollado una herramienta denominada *Tree-based Pipeline Optimization Tool* (TPOT), la cual es una biblioteca de código abierto desarrollada en Python que permite diseñar y optimizar automáticamente flujos de trabajo (*pipelines*) de aprendizaje automático para un dominio de problema específico. Su principal objetivo es automatizar el proceso de diseño, selección y optimización de flujos de trabajo de *machine learning*, buscando las mejores combinaciones de transformaciones de datos, selección de características y algoritmos de clasificación o regresión [24].

### 2.3.3.1. Programación Genética en TPOT

TPOT utiliza programación genética (GP) como estrategia de búsqueda para encontrar los mejores *pipelines* de aprendizaje automático de forma automatizada. Este enfoque está inspirado en el principio de la selección natural de Charles Darwin, donde las soluciones evolucionan generación tras generación para adaptarse mejor al problema planteado.

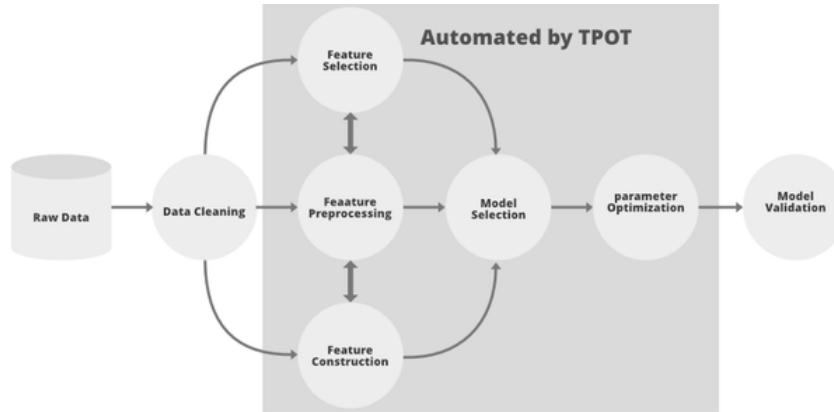


Figura 2.4. Partes del flujo de trabajo del aprendizaje automático automatizado por TPOT

El proceso evolutivo aplicado por TPOT incluye los siguientes mecanismos fundamentales:

- Selección:** Cada individuo (es decir, cada *pipeline* generado) es evaluado usando una función de aptitud que mide su rendimiento. Luego, los valores se normalizan entre 0 y 1. Se genera un número aleatorio  $R \in [0, 1]$  y se seleccionan los individuos cuya aptitud es mayor o igual a  $R$ .
- Crossover (Cruce):** Se seleccionan los individuos más aptos y se combinan entre sí mediante *crossover*, intercambiando subestructuras de los *pipelines* para generar nuevas combinaciones.
- Mutación:** A los individuos generados por *crossover* se les aplican modificaciones aleatorias, como cambiar un operador, insertar uno nuevo o eliminar parte del *pipeline*. Este paso introduce variabilidad en la población y puede mejorar la solución.

Al finalizar el proceso, TPOT devuelve el pipeline con mayor precisión y menor complejidad, garantizando así una solución equilibrada entre rendimiento y eficiencia computacional [25].

### 2.3.3.2. TPOTClassifier

El **TPOTClassifier** es un módulo para realizar aprendizaje automático automatizado en tareas de clasificación supervisada. Acepta los siguientes argumentos:

- **generations**: Número de iteraciones para el proceso de optimización del pipeline (valor predeterminado: 100).
- **population\_size**: Número de individuos a mantener en la población genética por cada generación (valor predeterminado: 100).
- **offspring\_size**: Número de descendientes generados por cada iteración de programación genética (valor predeterminado: 100).
- **mutation\_rate**: Tasa de mutación entre 0 y 1 (valor predeterminado: 0.9).
- **crossover\_rate**: Tasa de cruce entre 0 y 1, con la restricción de que la suma de `mutation_rate + crossover_rate <= 1` (valor predeterminado: 0.1).
- **scoring**: Métrica utilizada para evaluar la calidad del pipeline, como Accuracy, F1 score, etc.
- **cv**: Método de validación cruzada, si el valor es un entero, se utiliza como K en K-Fold cross-validation.
- **n\_jobs**: Número de procesos que se pueden ejecutar en paralelo (valor predeterminado: 1).
- **max\_time\_mins**: Tiempo máximo que TPOT está permitido optimizar el pipeline (valor predeterminado: None).
- **max\_eval\_time\_mins**: Tiempo máximo que TPOT tiene para evaluar un pipeline (valor predeterminado: None).
- **verbosity**: Cuánta información TPOT debe mostrar mientras se ejecuta (valores: 0-3).

### 2.3.3.3. TPOTRegressor

El **TPOTRegressor** funciona de manera similar, pero está diseñado para tareas de regresión. Acepta los mismos argumentos que el **TPOTClassifier**, con la diferencia principal en la métrica de evaluación, usando parámetros como:

- **neg\_median\_absolute\_error**, **neg\_mean\_absolute\_error**, **neg\_mean\_squared\_error**, **r2**.

Ambos módulos proporcionan las siguientes funciones para ajustar y evaluar el modelo:

- **fit(features, target)**: Ejecuta el proceso de optimización del pipeline sobre el conjunto de datos.
- **predict(features)**: Utiliza el pipeline optimizado para predecir valores objetivo.
- **score(test\_features, test\_target)**: Evalúa el modelo sobre los datos de prueba y devuelve la mejor puntuación.
- **export(output\_file\_name)**: Exporta el pipeline optimizado como código Python.

## 2.4. Tecnologías Web para el Desarrollo del Sistema

En esta sección se detallan las herramientas y tecnologías necesarias para construir la plataforma automatizada de entrenamiento.

### 2.4.1. Visual Studio Code

Un entorno de programación o entorno de desarrollo integrado (IDE) es un programa o aplicación de software que proporciona soluciones integrales a los programadores para desarrollar software, sea del tipo que sea. Todos los IDE normalmente poseen un editor de código fuente, herramientas varias para la automatización de tareas, validadores de código como linters, opciones para la compilación y un depurador, entre otras características. Uno de los IDE más extendidos para múltiples arquitecturas como puedan ser PHP, Angular, React, JavaScript, HTML y CSS es Visual Studio Code. Visual Code es un IDE (Entorno

de Desarrollo Integrado) desarrollado por Microsoft que permite editar los archivos en modo texto y presenta multitud de extensiones entre las que se incluyen soporte para la depuración, control integrado de Git y el resaltado de sintaxis.

#### 2.4.2. HTML

HTML (*Hypertext Markup Language*) es el lenguaje estándar utilizado para la estructura de contenido en la web. Cuando un usuario se conecta a Internet con un dispositivo cualquiera, se le asigna un identificador único mediante los protocolos TCP/IP (Protocolo de Control de Transmisión / Protocolo de Internet). El protocolo TCP proporciona el medio para crear las conexiones y el protocolo IP proporciona el mejor “camino” para alcanzar su destino. Este identificador único, más conocido como dirección IP, suele estar compuesto por cuatro códigos de 8 bits y vinculado a un nombre, también único, el cual utilizamos para acceder a un sitio web. Internet se mueve a través de direcciones IP, por lo que, para conseguir la dirección IP asignada a ese nombre que hemos introducido, primero se debe acceder a un sistema intermedio que almacena dicha relación. Ese sistema intermedio se conoce como DNS (Sistema de Nombres de Dominio) y, fundamentalmente, lo que hace es recopilar un catálogo de correspondencias de nombres e IPs y devolver un valor concreto como, por ejemplo, 216.58.211.35. Una vez que se tiene el objetivo al que dirigirse, el navegador, también llamado Cliente en términos de comunicaciones, abre una instancia de comunicación con el servidor mediante el protocolo HTTP (Protocolo de Transferencia de Hipertexto). Este protocolo es quién dicta las normas para que el Cliente se comunique con el Servidor Web asignado a la IP anteriormente adquirida y es, además, quién define la sintaxis y semántica que se debe utilizar en cada conexión.

En este tipo de comunicación, el servidor establece un cifrado basado en la seguridad de textos mediante los protocolos criptográficos SSL/TLS, los cuales, permiten crear una capa codificada intermedia entre los protocolos HTTP y TCP/IP por el que envía el código HTML que el navegador muestra al usuario [26].

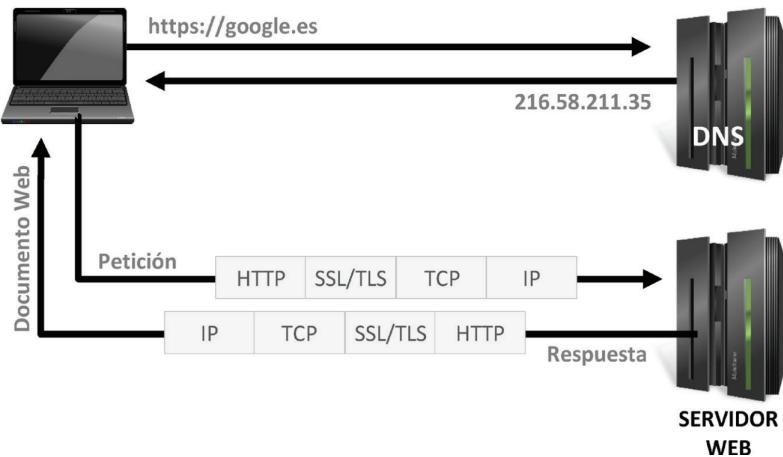


Figura 2.5. Proceso de una petición web.

Si bien HTML es un lenguaje formado por entidades que ayudan a estructurar y proporcionar significado a las diferentes partes del documento, cada una de estas entidades, usualmente denominadas elementos o etiquetas, están formadas por un contenido y cero, uno o varios atributos. Cada uno de los atributos tiene una función y puede estar o no asociado a un comportamiento o definición específica. Por ejemplo, el atributo ID habitualmente es utilizado para poder manipular el elemento a través de un nombre corto, sin embargo, también puede ser declarado para vincularse con otro elemento generando una entidad mayor, como es el caso del siguiente código.

```

1 <label for="nombre">Nombre</label>
2 <input id="nombre" placeholder="Inserte el nombre completo" />
```

Código 1. Etiquetado de un campo de formulario en HTML

El atributo FOR, utiliza el atributo ID para vincular el LABEL con el INPUT y generar un elemento combinado o pequeño componente. Cabe destacar que, aunque puede haber etiquetas sin cierre, como es el caso del elemento INPUT, lo normal es que todas las etiquetas o marcas tengan un principio y un final, como es el caso de la etiqueta LABEL.

### 2.4.3. CSS

Las hojas de estilo, o archivos CSS (*Cascading Style Sheets* por sus siglas en inglés), son los que aplican los estilos a un sitio web en forma de cascada, leyendo el código de arriba hacia abajo. Una regla CSS es la unidad fundamental en la que se basa la creación de estilos en una página web. Es una instrucción que define cómo se debe visualizar un elemento HTML, incluyendo su color, tamaño, posición y otros atributos.

```
1  h1 {                      /* selector */
2    color: blue;            /* propiedad: color, valor: blue */
3    font-size: 24px;         /* propiedad: font-size, valor: 24px */
4  }                          /* fin de la declaración */
```

Código 2. Estructura de una regla CSS

El selector indicará a qué componente del HTML se le aplicarán las propiedades escritas. Dentro de la declaración tendremos que indicar una propiedad y un valor a modificar. Las propiedades que podemos alterar son varias, algunas dependerán de que otras propiedades tengan ciertos valores específicos [27].

### 2.4.4. JavaScript

JavaScript es un lenguaje de programación ampliamente utilizado para desarrollar funcionalidades interactivas en el navegador. Con JavaScript, se pueden manipular elementos HTML, gestionar eventos, interactuar con el usuario en tiempo real, y actualizar la interfaz sin necesidad de recargar la página (a través de técnicas como AJAX). En el desarrollo de nuestra plataforma, JavaScript será clave para implementar la lógica interactiva, manejar formularios de usuarios, validar entradas y realizar peticiones asincrónicas al servidor para obtener o enviar datos sin interrumpir la experiencia del usuario [28].

#### **2.4.5. Node.js y su ecosistema**

Node.js es un entorno de ejecución para JavaScript del lado del servidor, construido sobre el motor V8 de Google Chrome. Este entorno permite ejecutar código JavaScript fuera del navegador, lo que lo convierte en una herramienta poderosa para el desarrollo de aplicaciones web modernas. Su diseño se basa en un modelo de entrada/salida asincrónico, no bloqueante y orientado a eventos, lo cual facilita la creación de aplicaciones altamente escalables, con un uso eficiente de los recursos del sistema. Una de sus características clave es su arquitectura de un solo hilo que, a través del bucle de eventos, permite atender múltiples solicitudes concurrentes sin necesidad de recurrir al procesamiento en paralelo mediante múltiples hilos, como ocurre en otros entornos más tradicionales [29].

En el contexto del desarrollo de la plataforma propuesta, Node.js será la tecnología central empleada para la construcción del servidor backend. Desde este entorno se gestionarán las peticiones HTTP que provienen del cliente, se establecerá la comunicación con la base de datos y se expondrán las interfaces de programación de aplicaciones (APIs) necesarias para que el frontend pueda interactuar de manera fluida con el sistema. Esta arquitectura facilita el desarrollo de aplicaciones en tiempo real y con alta capacidad de respuesta, cualidades esenciales para el tipo de solución que se busca implementar [29].

Además, Node.js cuenta con un ecosistema robusto de módulos y paquetes a través del gestor npm (Node Package Manager). Este ecosistema permitirá integrar fácilmente funcionalidades avanzadas como el manejo de archivos, la autenticación y autorización de usuarios, la validación de datos, la conexión con bases de datos relacionales o no relacionales, y la creación de APIs RESTful seguras y eficientes. Gracias a esta flexibilidad, se podrá acelerar el proceso de desarrollo, asegurar una buena mantenibilidad del código y escalar la plataforma en función de las necesidades futuras [29].

#### **2.4.6. MySQL**

MySQL es un sistema gestor de bases de datos (SGBD, DBMS por sus siglas en inglés) muy conocido y ampliamente usado por su simplicidad y notable rendimiento. Una de las ventajas clave de MySQL es su portabilidad y compatibilidad con múltiples lenguajes de

programación, lo que permite desarrollar aplicaciones basadas en bases de datos utilizando lenguajes como Java, Python, PHP, C++, entre otros. Además, puede ejecutarse en una gran variedad de sistemas operativos, incluyendo Linux, Windows, MacOS y otros menos comunes [30].

En el desarrollo de la plataforma, MySQL se utilizará para almacenar y gestionar datos relacionados con los usuarios, los registros de entrenamiento, las predicciones generadas por el sistema y otros datos relevantes para el funcionamiento del sistema. Su capacidad para realizar consultas rápidas y complejas lo convierte en la opción ideal para manejar los datos dinámicos y las relaciones entre diferentes conjuntos de información en la plataforma.

---

# CAPÍTULO 3

---

## Desarrollo y resultados

En esta sección del documento se redacta el desarrollo para la creación del sistema capaz de recibir archivos de audio cardíaco útiles para el reentrenamiento de un sistema de detección de enfermedades cardíacas.

### 3.1. Backend

Para el desarrollo del *backend*, fue necesario instalar y configurar diversas herramientas y bibliotecas, que permitieron gestionar las cargas de archivos y almacenar los metadatos en la base de datos MySQL. Entre ellas destacan:

- Node.js: Entorno de ejecución de JavaScript en el servidor.
- MySQL: Sistema de gestión de bases de datos utilizado para almacenar los registros de los audios cardíacos y sus metadatos.
- Express.js: *Framework* para la creación de rutas y *middleware* en el servidor.
- Multer: *Middleware* para manejar formularios `multipart/form-data`, utilizado para la subida de archivos.
- bcrypt: Para el cifrado de contraseñas de los usuarios.
- dotenv: Para la gestión de variables de entorno.

Se instalaron las siguientes dependencias mediante `npm` (*Node Package Manager*), ejecutando los siguientes comandos en la terminal del proyecto:

```
1 npm install express multer mysql2 bcrypt dotenv
```

Código 3. Instalación de dependencias desde la terminal de Visual Studio Code.

El archivo principal para el *backend* se denomina `index.js`. Comienza con la importación de las librerías necesarias:

```
1 const express = require('express');
2 const app = express();
3 const mysql = require('mysql2/promise');
4 const multer = require('multer');
5 const cors = require('cors');
6 const bodyParser = require('body-parser');
7 const path = require('path');
8 const port = 3000;
9 const Joi = require('joi');
10 const bcrypt = require('bcrypt');
```

Código 4. Importación de módulos necesarios para el funcionamiento del servidor.

En esta sección del código se importan los módulos necesarios para la implementación del servidor en Node.js. Se utiliza `express` como *framework* principal para gestionar las rutas y peticiones HTTP. La conexión a la base de datos se realiza mediante `mysql2/promise`, que permite usar consultas asíncronas. `multer` se encarga de procesar las cargas de archivos desde el *frontend*. `cors` habilita el acceso entre dominios. `body-parser` y los métodos de `express` permiten interpretar los cuerpos de las peticiones. `path` se emplea para manipular rutas de archivos. Además, `Joi` se utiliza para validar los datos del usuario y `bcrypt` para encriptar contraseñas. Finalmente, se define el puerto en el que se ejecutará el servidor.

En el Código 5 se presenta la configuración del *middleware* en el servidor para procesar las solicitudes entrantes, estas interceptan y procesan las solicitudes entrantes antes de que lleguen a las rutas:

- `express.static('public')`: permite servir archivos estáticos como HTML, CSS y JS desde la carpeta `public`.
- `bodyParser.urlencoded( extended: true )` y `bodyParser.json()`: habilitan la in-

interpretación de datos enviados desde formularios y en formato JSON, respectivamente.

- `cors()`: permite solicitudes entre distintos orígenes, algo común cuando el *frontend* y el *backend* no están en el mismo servidor.
- `express.json()` y `express.urlencoded(...)`: hacen lo mismo que `bodyParser`, pero usando la API moderna integrada de Express (se incluyen ambas por compatibilidad).

Esto asegura que los datos enviados desde el *frontend* puedan ser correctamente interpretados y utilizados por las rutas del servidor.

```
1 app.use(express.static('public')); // Para servir archivos estáticos
2 app.use(bodyParser.urlencoded({ extended: true }));
3 app.use(bodyParser.json());
4 app.use(cors());
5 app.use(express.json());
6 app.use(express.urlencoded({ extended: true }));
```

Código 5. Configuración de *middleware* para procesar solicitudes entrantes.

En el siguiente paso, se define un esquema de validación usando la librería `Joi`, el cual define las reglas que deben cumplir los datos enviados por el usuario al momento de registrarse.

- `nombre`: debe ser una cadena de texto y es obligatorio.
- `email`: debe tener formato válido de correo electrónico.
- `password`: también es obligatorio y se espera como texto plano (aunque luego se cifrará).
- `aceptarTerminos`: debe ser un valor booleano (verdadero/falso) y es obligatorio, indicando que el usuario ha aceptado los términos y condiciones del sistema.

Esta validación, como se puede observar en el Código 6 se aplica antes de almacenar los datos en la base de datos para garantizar la integridad y seguridad del sistema.

```
1 const registerSchema = Joi.object({
2   nombre: Joi.string().required(),
3   email: Joi.string().email().required(),
4   password: Joi.string().required(),
5   aceptarTerminos: Joi.boolean().required(),
6 });

```

Código 6. Esquema de validación de datos de registro con Joi.

Posteriormente se configura `multer` para guardar los archivos subidos en la carpeta `uploads`:

```
1 const storage = multer.diskStorage({
2   destination: (req, file, cb) => cb(null, 'uploads'),
3   filename: (req, file, cb) => {
4     const uniqueName = `${Date.now()}-${Math.round(Math.random() * 1E9)}${
5       path.extname(file.originalname)}`;
6     cb(null, uniqueName); \code{
7   }
8 });
9 const upload = multer({ storage });

```

Código 7. Configuración de almacenamiento de archivos con Multer.

Se usa la librería `multer` para procesar formularios de tipo `multipart/form-data`, el formato usado en formularios que contienen archivos. Como se puede observar en el Código 7, `diskStorage` permite personalizar dos aspectos:

- `destination`: indica que los archivos se guardarán en la carpeta `uploads`.

- **filename**: define un nombre único para cada archivo, compuesto por la fecha actual, un número aleatorio y la extensión original del archivo.

Finalmente, `const upload = multer( storage )` crea una instancia del *middleware* de `multer` utilizando esta configuración, que luego será utilizada en la ruta `/subir`.

Acto seguido, se establece la conexión a la base de datos MySQL mediante un `pool` de conexiones, lo cual permite manejar múltiples solicitudes concurrentes de forma eficiente. Se utiliza `createPool()` desde `mysql2/promise` para establecer los parámetros básicos: `host`, `user`, `password` y `database`.

```

1  const db = mysql.createConnection({
2
3     host: 'localhost',
4
5     user: 'root',
6
7     password: 'mi contraseña',
8
9     database: 'heartforge'
10
11 });
12
13 (async () => {
14
15     try {
16
17         const [rows] = await pool.query('SELECT NOW() AS now');
18
19         console.log(' Conexión a MySQL exitosa:', rows[0]);
20
21     } catch (err) {
22
23         console.error(' Error de conexión inicial:', err);
24
25     }
26
27 })();

```

Código 8. Conexión a la base de datos MySQL

Como se puede observar en el Código 8, una función asíncrona realiza una consulta simple (`SELECT NOW()`) para probar si la conexión se ha realizado correctamente. Si es exitosa, se imprime la hora actual y un mensaje de éxito en consola. Si falla, se muestra un mensaje de error con los detalles.

En la plataforma se implementan dos rutas clave para la gestión de usuarios. La ruta del registro permite a nuevos usuarios crear una cuenta, se valida que todos los campos estén completos, verifica que el usuario no exista ya en la base de datos y se cifra la contraseña antes de almacenarla. En caso de éxito, devuelve una confirmación al cliente.

Para el inicio de sesión se autentica al usuario comparando la contraseña ingresada con la contraseña cifrada almacenada. Si coinciden, el inicio de sesión es exitoso. Si no, se notifica al usuario que sus credenciales son incorrectas o que no está registrado.

El *endpoint* `/subir` recibe los datos del formulario y el archivo de audio:

```

1 app.post('/subir', upload.single('audio'), async (req, res) => {
2   try {
3     if (!req.file) {
4       return res.status(400).send("No se recibió el archivo");}
5     console.log("Archivo recibido:", req.file);
6     console.log("Datos recibidos:", req.body);
7     const { nombre_paciente, sexo, edad, estatura, peso, categoria
8   } = req.body;
9     const nombre_archivo = req.file.filename;
10    const sql = '
11      INSERT INTO audios (usuario_id, nombre_archivo, fecha_subida,
12      categoria, nombre_paciente, edad, estatura, peso, sexo)
13      VALUES (?, ?, NOW(), ?, ?, ?, ?, ?, ?)';
14    await pool.query(sql, [usuario_id, nombre_archivo, categoria,
15      nombre_paciente, edad, estatura, peso, sexo]);
16    res.send(" Archivo subido y datos registrados con éxito");
17  } catch (error) {
18    console.error("Error al subir el archivo:", error);
19    res.status(500).send("Error en el servidor");}
20 });

```

Código 9. Ruta para subir archivos

Por último, se define el puerto en el que se ejecuta el servidor:

```

1 app.listen(port, () => {
2   console.log('Servidor corriendo en http://localhost:${port}');
3 });

```

Código 10. Inicialización del servidor

## 3.2. Frontend

La aplicación fue nombrada **HeartForge**, para darle un toque y estructura único. El *frontend* de HeartForge está diseñado para proporcionar una interfaz web intuitiva, amigable y responsive que permite a los usuarios interactuar con las funcionalidades principales del sistema. Se desarrolló utilizando HTML, CSS y JavaScript, priorizando la simplicidad, la claridad visual y la facilidad de uso.

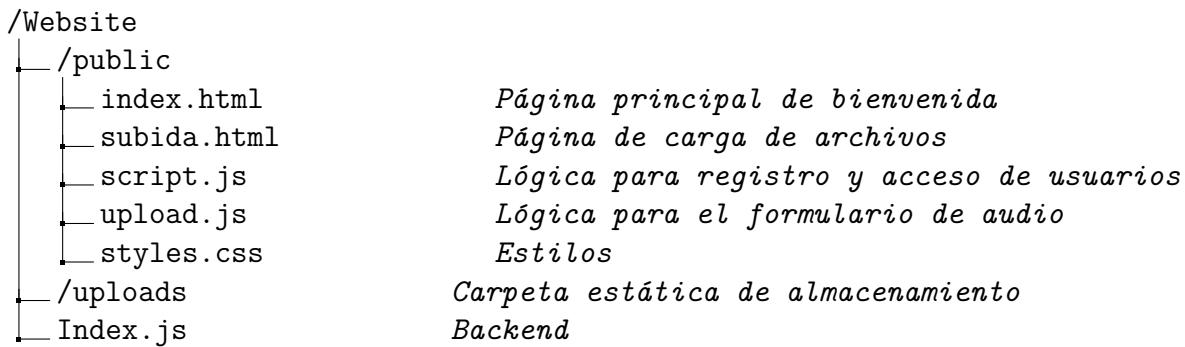


Figura 3.1. Estructura general de los archivos de la aplicación

En el entorno de desarrollo de Visual Studio Code, se inicio estructurando la página principal. El documento inicia con la declaración `<!DOCTYPE html>` para indicar que se trata de un archivo HTML. Se establece el idioma principal como español (`lang=".es"`), el conjunto de caracteres como UTF-8, y las configuraciones de compatibilidad y visualización para dispositivos móviles con las metaetiquetas correspondientes.

Se enlaza un archivo externo de estilos CSS denominado `styles.css` para definir la apariencia visual de la página.

Dentro de la etiqueta `<header>`, se incluye el nombre del sistema como título, junto con una barra de navegación `<nav>` que contiene enlaces de anclaje para navegar dentro de la misma página:

```

1 <body>
2   <header>
3     <h2 class="HeartForge">HeartForge</h2>
4     <nav class="navigation">
5       <a href="#inicio">Inicio</a>
6       <a href="#sobre-nosotros">Sobre nosotros</a>
7       <a href="#servicios">Servicios</a>
8       <a href="#contacto">Contacto</a>
9       <button class="btnAcceder-popup">Acceder</button>
10      </nav>
11    </header>

```

Código 11. Barra de navegación

Este menú permite al usuario desplazarse rápidamente a las diferentes secciones de la página.



Figura 3.2. Vista de la barra de navegación en línea

Dentro de la clase `.wrapper`, se define una estructura emergente que contiene los formularios de Acceder y Registro. Esta ventana se muestra como un *popup* (modal) cuando el usuario hace clic en el botón Acceder. Además, contiene un ícono de cierre (`icon-close`) que permite ocultar la interfaz flotante.

```

1 <div class="wrapper">
2   <span class="icon-close">
3     <ion-icon name="close"></ion-icon>
4   </span>

```

Código 12. Inicio del contenedor de formularios.

La sección con clase `.form-box acceder` contiene el formulario HTML para que los usuarios puedan iniciar sesión. El formulario utiliza el método `POST` y se conecta a la ruta `/login` del servidor Node.js. Incluye dos campos: correo electrónico y contraseña. Cada campo está envuelto en un contenedor `.input-box`, acompañado de un ícono representativo (correo y candado, respectivamente).

Además, se proporciona una casilla para recordar sesión y un enlace de recuperación de contraseña. Al final del formulario se incluye un botón para enviar los datos y un mensaje con un enlace para registrarse si el usuario no tiene cuenta.

```
1 <div class="form-box acceder">
2   <h2>Iniciar sesión</h2>
3   <form id="login-form" method="POST" action="http://localhost:3000/login">
4     <div class="input-box">
5       <span class="icon"><ion-icon name="mail"></ion-icon></span>
6       <input type="email" name="email" placeholder="Correo electrónico" required>
7       <label>Correo</label>
8     </div>
9     <!-- Otro campo -->
10    </form>
11  </div>
```

Código 13. Formulario HTML de inicio de sesión.

Como se puede observar en la Figura 3.3, el formulario de Acceder, permite a los usuarios iniciar sesión mediante su correo electrónico y contraseña, incluye un *checkbox* “Recordarme” para guardar la sesión y un enlace para recuperar la contraseña.

El segundo formulario, contenido dentro de la clase `.form-box register`, representado en el Código 14 permite a los nuevos usuarios crear una cuenta. Se conecta al *endpoint* `/register` del *backend* mediante el método `POST`. Este formulario solicita el nombre del usuario, correo electrónico y una contraseña, con campos estructurados en bloques `input-box`, cada uno acompañado de su respectivo ícono.

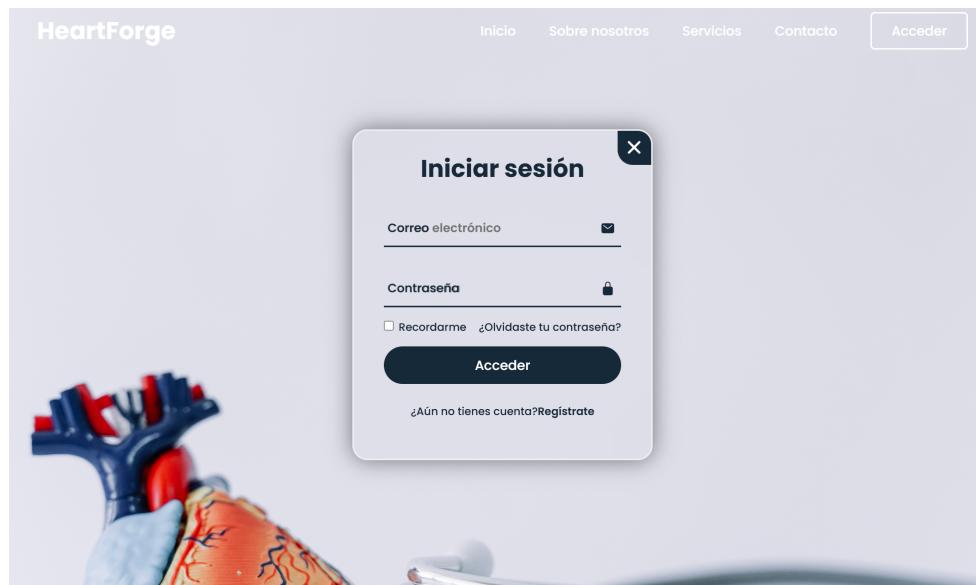


Figura 3.3. Inicio de sesión

Ambos formularios usan íconos de la librería Ionicons para mejorar la presentación.

```
1 <div class="form-box register">
2   <h2>Crea una cuenta</h2>
3   <form id="signup-form" method="POST" action="http://localhost:3000/register">
4     <div class="input-box">
5       <span class="icon"><ion-icon name="person"></ion-icon></span>
6       <input type="text" name="nombre" required>
7       <label>Nombre de usuario</label>
8     </div>
9     <!-- Otros campos -->
10    </form>
11  </div>
```

Código 14. Formulario HTML de registro.

A continuación se observa como al crear una cuenta se incluye una casilla para aceptar términos y condiciones, un botón de envío y un enlace para redirigir a la sección de acceso si el usuario ya tiene cuenta.

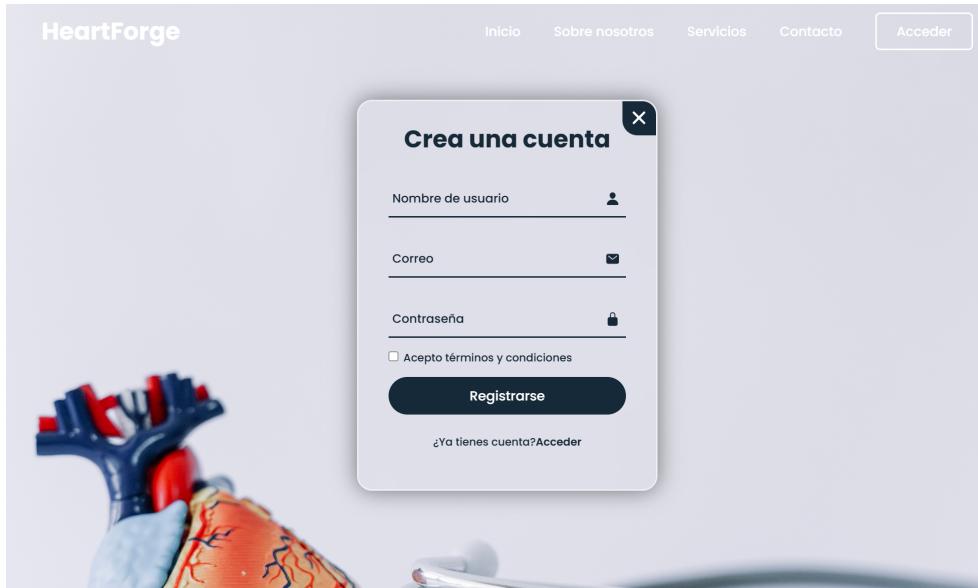


Figura 3.4. Ventana para crear una cuenta

Esta interfaz está diseñada para integrarse con validaciones desde el servidor y ofrecer una experiencia clara y responsiva.

La estructura de la página principal del sistema HeartForge está organizada en diversas secciones que permiten al usuario navegar intuitivamente por el sitio y conocer sus funcionalidades clave.



Figura 3.5. Inicio de la página principal

La interfaz web inicia con un mensaje de bienvenida que presenta al usuario la plataforma **HeartForge**. La vista del Inicio esta disponible en la Figura 3.5, esta plataforma fue desarrollada con el objetivo de facilitar el diagnóstico oportuno de enfermedades cardíacas,

mediante un sistema que permite el entrenamiento automático de clasificadores de audio cardíaco. El enfoque principal está orientado a profesionales de la salud, quienes podrán actualizar sus herramientas diagnósticas sin requerir conocimientos técnicos especializados.

En la sección “Sobre Nosotros”, se introduce al equipo de trabajo detrás de HeartForge, destacando su enfoque interdisciplinario y su compromiso con el bienestar social. Puede apreciarse en la Figura 3.6, esta sección busca generar confianza en los usuarios, mostrando la visión y los valores que motivan el desarrollo del proyecto.



Figura 3.6. Vista de la sección “Sobre nosotros” de la plataforma HeartForge.

La sección “Servicios” describe de forma clara y concisa el funcionamiento del sistema a través de cuatro pasos: la carga del archivo de audio cardíaco, el proceso de entrenamiento automático del modelo, la actualización del sistema con los nuevos datos y la presentación de los resultados listos para ser utilizados por el personal médico. Además, como se puede ver en la Figura 3.7 esta sección incluye un botón que redirige al usuario a la página `subida.html`, donde es posible subir nuevos archivos de audio para integrarse al clasificador.



Figura 3.7. Descripción del servicio y botón para cargar archivos de audio en la página principal

Por último, la sección “Contacto” proporciona la información necesaria para establecer comunicación directa con el equipo de HeartForge. Incluye una dirección de correo electrónico para consultas o soporte, así como una invitación a seguir las redes sociales del proyecto, fomentando así la interacción con los usuarios y la comunidad interesada en soluciones tecnológicas para la salud.

Tras definir la estructura HTML, se implementa la lógica dinámica en `script.js`, encargado de gestionar la interacción con la interfaz: manipulación del DOM, validación de formularios, autenticación y control de sesión.

Se utiliza el evento `DOMContentLoaded` para asegurar que el contenido de la página esté completamente cargado antes de ejecutar el `script`. Esto garantiza que los elementos HTML puedan ser referenciados y manipulados correctamente.

```
1 document.addEventListener('DOMContentLoaded', () => {  
2     const wrapper = document.querySelector('.wrapper');  
3     const accederLink = document.querySelector('.acceder-link');  
4     const registerLink = document.querySelector('.register-link');  
5     const btnPopup = document.querySelector('.btnAcceder-popup');  
6     const iconClose = document.querySelector('.icon-close');  
7     const loginForm = document.querySelector('#login-form');  
8     const registerForm = document.querySelector('#signup-form');
```

Código 15. Inicialización del DOM y selección de elementos

Mediante `querySelector`, se obtienen referencias a los elementos HTML clave, tales como: Contenedor principal (`wrapper`), que gestiona el cambio de clases para mostrar o ocultar los formularios y el *popup*, enlaces para alternar entre el formulario de registro y el de acceso, botón de acceso (`btnPopup`) y el ícono para cerrar el *popup* (`iconClose`), así como formularios de registro (`registerForm`) e inicio de sesión (`loginForm`).

```

1 registerLink.addEventListener('click', () => {
2   wrapper.classList.add('active');
3 });
4 accederLink.addEventListener('click', () => {
5   wrapper.classList.remove('active');
6 });
7 btnPopup.addEventListener('click', () => {
8   wrapper.classList.add('active-popup');
9 });
10 iconClose.addEventListener('click', () => {
11   wrapper.classList.remove('active-popup');
12 });

```

Código 16. Manejo de eventos para la visualización del formulario emergente

Como se puede observar en el Código 16, al hacer clic en el enlace de registro, se añade la clase `.active` al contenedor, mostrando el formulario de registro. Al hacer clic en el enlace de acceso, se elimina la clase `.active`, mostrando el formulario de inicio de sesión. El botón de acceso (Acceder) muestra el *popup* añadiendo la clase `.active-popup`, mientras que el ícono de cierre la elimina, ocultando el formulario.

El siguiente fragmento de código se encarga de interceptar el evento `submit` del formulario de registro. Se previene el comportamiento por defecto con `preventDefault()` para manejar la lógica de validación en el cliente. Posteriormente, se obtienen los valores de los campos: nombre, correo electrónico, contraseña y la aceptación de los términos y condiciones. Si el usuario no ha marcado la casilla correspondiente, se interrumpe el proceso y se muestra un mensaje de alerta (véase Código 17).

```

1 registerForm.addEventListener('submit', async (e) => {
2   e.preventDefault();
3
4   const nombre = registerForm.querySelector('input[type="text"]').value;
5   const email = registerForm.querySelector('input[type="email"]').value;
6   const password = registerForm.querySelector('input[type="password"]').value;
7   const aceptarTerminos = registerForm.querySelector('input[type="checkbox"]').checked;
8
9   if (!aceptarTerminos) {
10     alert('Debes aceptar los términos y condiciones');
11     return;
12   }

```

Código 17. Validación del formulario de registro en el cliente

Tras la validación inicial, se realiza una petición POST al servidor mediante la función asíncrona `fetch()`, enviando los datos del formulario en formato JSON.

```

1 try {
2
3   const res = await fetch('http://localhost:3000/register', {
4     method: 'POST',
5     headers: { 'Content-Type': 'application/json' },
6     body: JSON.stringify({ nombre, email, password, aceptarTerminos }));
7
8   const data = await res.json();
9
10  if (data.success) {
11    alert('Registro exitoso');
12    wrapper.classList.remove('active');
13  } else {
14    alert('Hubo un error al registrar el usuario'); } } );

```

Código 18. Validación del formulario de registro en el cliente

Si la respuesta del servidor es exitosa, se muestra una notificación de confirmación y se

oculta el formulario de registro. En caso contrario, se captura el error y se notifica al usuario (véase Código 18).

De forma similar, al enviar el formulario de inicio de sesión, se obtiene el correo y la contraseña proporcionados. Se realiza una solicitud POST al *backend* (/login) para autenticar al usuario. Si las credenciales son válidas, el sistema cambia el texto del botón de acceso a “Cuenta” y guarda el estado de la sesión en `localStorage` mediante la clave `sesionIniciada`.

```
1 if (data.success) {  
2     alert('Inicio de sesión exitoso');  
3     btnPopup.textContent = 'Cuenta';  
4     localStorage.setItem('sesionIniciada', 'true');  
5     loginForm.reset();  
6     if (rememberMe) {  
7         localStorage.setItem('rememberedEmail', email);  
8     }  
9 }
```

Código 19. Gestión del estado de sesión y almacenamiento local tras inicio exitoso

Si el usuario seleccionó la opción “Recordarme”, se guarda también el correo electrónico para autocompletarlo en futuras visitas.

```
1 window.addEventListener('DOMContentLoaded', () => {  
2     const rememberedEmail = localStorage.getItem('rememberedEmail');  
3     if (rememberedEmail) {  
4         document.querySelector('#login-form input[type="email"]').value = rememberedEmail;  
5     }  
6 });
```

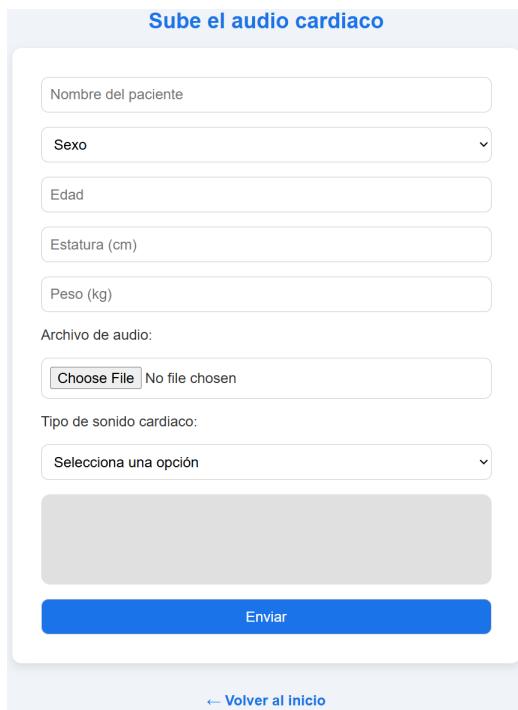
Código 20. Autocompletar correo si está guardado en `localStorage`

Al cargar la página, si existe un correo guardado en `localStorage`, este se asigna au-

tomáticamente al campo de correo del formulario. Asimismo, si el indicador de sesión activa (`sesionIniciada`) está presente, el botón de acceso muestra el texto “Cuenta”, reflejando que la sesión sigue activa.

Además de la estructura de la página principal y la lógica de autenticación, el sistema incorpora una funcionalidad para la carga de archivos de audio. En el apartado de Servicios, se encuentra el botón para poder subir el archivo de audio como se observa en la Figura 3.7.

Para la funcionalidad de subida de archivos de audio cardíaco, se diseñó un formulario web que permite a los usuarios ingresar datos del paciente y cargar el audio correspondiente. El formulario incluye campos para nombre, sexo, edad, estatura, peso, y tipo de sonido cardíaco. A continuación, se observa el formulario para subir el archivo y datos relevantes sobre el paciente.



The screenshot shows a web-based form titled "Sube el audio cardiaco". The form consists of several input fields and a file upload section. At the top, there are five text input fields: "Nombre del paciente", "Sexo" (with a dropdown arrow), "Edad", "Estatura (cm)", and "Peso (kg)". Below these is a section for "Archivo de audio:" which includes a "Choose File" button and a message "No file chosen". Underneath is a dropdown menu labeled "Tipo de sonido cardiaco:" with the option "Selecciona una opción". In the center of the form is a large, light-gray rectangular area for previewing the uploaded audio file. At the bottom is a prominent blue "Enviar" button. Below the button is a small link "← Volver al inicio".

Figura 3.8. Formulario para subir archivos de audio y reunir datos del paciente

Para mejorar la precisión en la captura de metadatos, el formulario permite seleccionar el sexo del paciente mediante un menú desplegable (Figura 3.9).

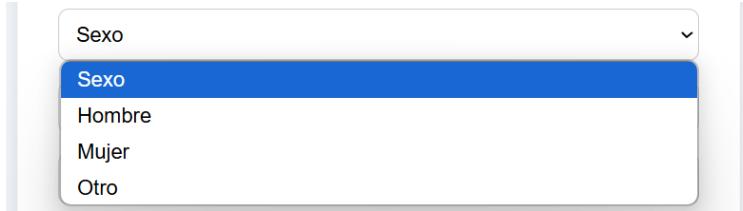


Figura 3.9. Opciones disponibles para seleccionar el sexo del paciente en el formulario.

Asimismo, el usuario puede indicar el tipo de sonido cardíaco correspondiente (Figura 3.10).

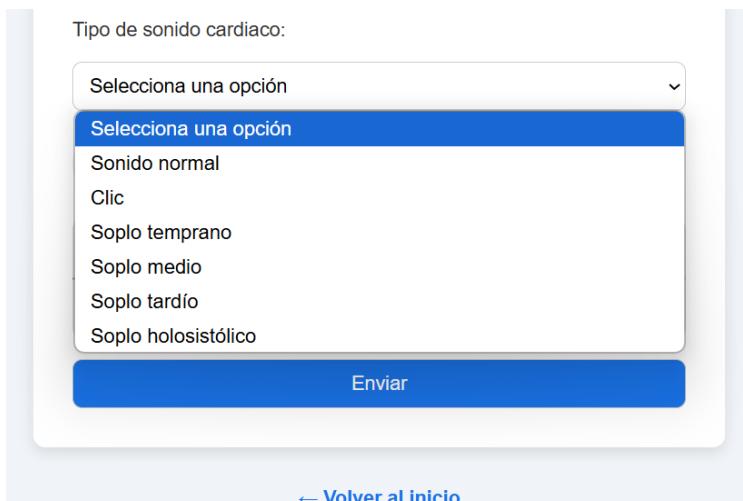


Figura 3.10. Despliegue de opciones para indicar el tipo de sonido cardíaco correspondiente.

En caso de omitir campos obligatorios, el sistema muestra una alerta visual que advierte sobre la necesidad de completar todos los datos requeridos antes de continuar (Figura 3.11).

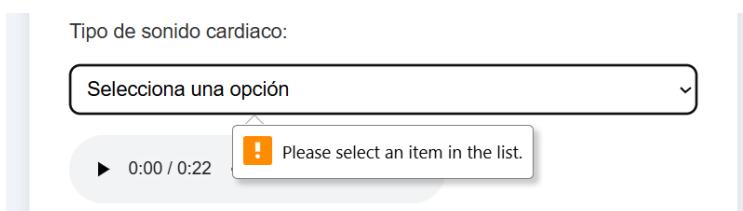


Figura 3.11. Mensaje de advertencia al dejar un campo obligatorio sin completar en el formulario.

El formulario permite la previsualización y reproducción del audio, facilitando la validación del archivo antes de la carga definitiva.

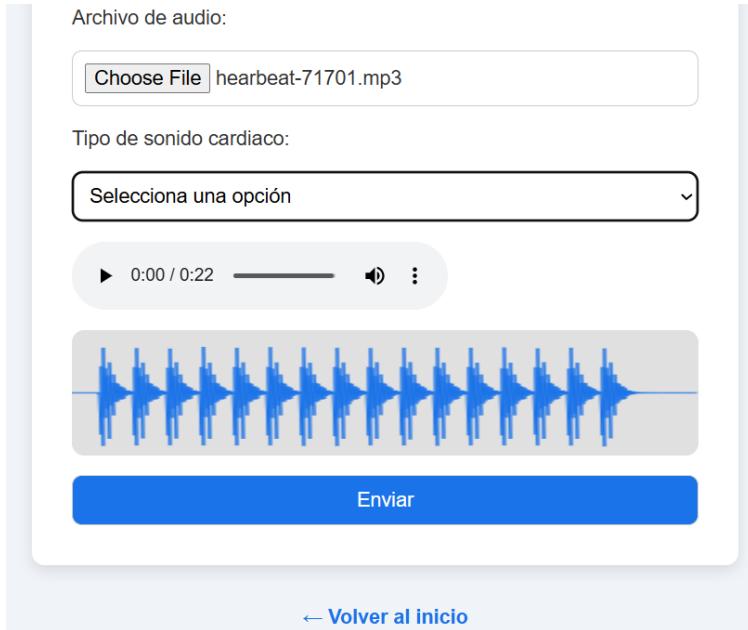


Figura 3.12. Reproducción y visualización del audio cardiaco.

El Código 21. permite generar una URL temporal con `URL.createObjectURL(file)` a partir del archivo cargado, asignarla al elemento reproductor de audio (`audioPlayer`) y mostrarlo al usuario.

```

1  document.getElementById('audio').addEventListener('change', function() {
2
3    const file = this.files[0];
4
5    const player = document.getElementById('audioPlayer');
6
7
8    if (file) {
9      const url = URL.createObjectURL(file);
10     player.src = url;
11     player.style.display = 'block';
12     visualizarAudio(file);
13   }
14 });

```

Código 21. Inicialización del reproductor y llamada a la visualización.

La Figura 3.13 muestra el popup de confirmación visual que se despliega tras la subida exitosa del archivo:

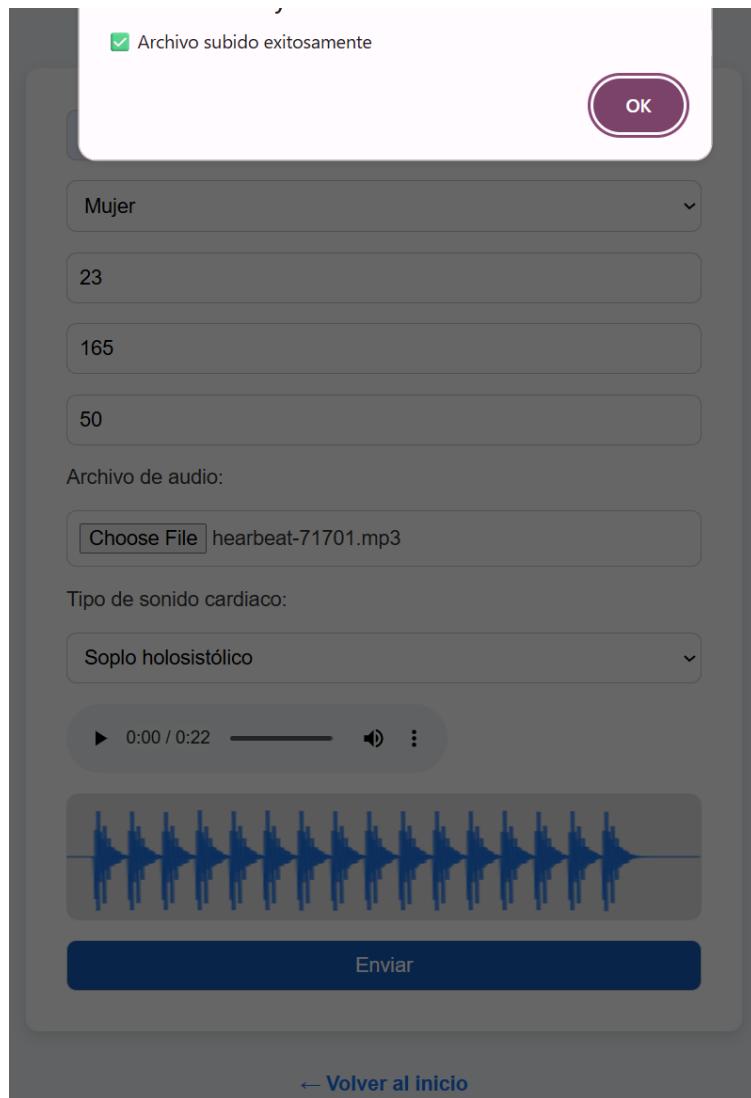


Figura 3.13. Popup de éxito tras la subida del audio.

### 3.3. Base de Datos

Para el correcto funcionamiento del sistema HeartForge, se diseñó e implementó una base de datos relacional utilizando MySQL, que permite gestionar eficientemente la información de los usuarios registrados y los archivos de audio cardíaco subidos al sistema. Esta base de datos actúa como el núcleo de almacenamiento de información, garantizando integridad, seguridad y disponibilidad de los datos.

La base de datos fue creada en un servidor local de MySQL, utilizando la herramienta de línea de comandos o interfaces gráficas como phpMyAdmin o MySQL Workbench. La creación se realizó mediante el siguiente comando SQL:

```
1 CREATE DATABASE HeartForge;
```

Código 22. Comando para crear la base de datos

El diseño lógico se centró en la creación de dos tablas principales: tabla de usuarios y tabla de audios.

La tabla usuarios almacena la información de los usuarios registrados en el sistema, tales como nombre, correo electrónico y contraseña encriptada.

Tabla 3.1. Estructura de la tabla `usuarios`

Campo	Tipo de dato	Descripción
id	INT (PK)	Identificador único del usuario
nombre	VARCHAR	Nombre del usuario
correo	VARCHAR	Correo electrónico del usuario
contrasena	VARCHAR	Contraseña en formato hash

Su función principal es autenticar el acceso y vincular a cada usuario con los archivos que sube, véase especificación en Tabla 3.1 El siguiente código SQL muestra la instrucción para crear esta tabla:

```
1 CREATE TABLE usuarios (
2     id INT PRIMARY KEY AUTO_INCREMENT,
3     nombre VARCHAR(100) NOT NULL,
4     correo VARCHAR(100) NOT NULL UNIQUE,
5     contrasena VARCHAR(255) NOT NULL
6 );
```

Código 23. Comando SQL para la creación de la tabla `usuarios`.

En la tabla audios se almacenan los registros correspondientes a los archivos de audio subidos por los usuarios, así como los metadatos clínicos asociados al paciente, la estructura se presenta en la Tabla 3.2.

Tabla 3.2. Estructura de la tabla `audios`

Campo	Tipo de dato	Descripción
id	INT (PK)	Identificador único del audio
usuario_id	INT (FK)	Identificador del usuario que subió el audio
nombre_archivo	VARCHAR	Nombre del archivo almacenado
fecha_subida	DATETIME	Fecha y hora en que se subió el archivo
categoria	VARCHAR	Categoría asignada al audio
nombre_paciente	VARCHAR	Nombre del paciente relacionado al audio
edad	INT	Edad del paciente
estatura	FLOAT	Estatura del paciente
peso	FLOAT	Peso del paciente
sexo	VARCHAR	Sexo del paciente

La base de datos establece una relación uno-a-muchos entre las tablas usuarios y audios, donde un usuario puede subir múltiples archivos de audio, pero cada archivo está vinculado a un único usuario.

Para la conexión entre Node.js y MySQL, se utilizó el paquete `mysql2/promise`, lo que permite ejecutar consultas de manera asincrónica y eficiente.

Los datos sensibles, como las contraseñas, se almacenan encriptados utilizando el algoritmo `bcrypt`, lo cual refuerza la seguridad de la información del usuario y evita el almacenamiento de contraseñas en texto plano.

Durante el proceso de registro, la contraseña se cifra mediante el siguiente comando:

```
1 const hashedPassword = await bcrypt.hash(password, 10);
```

Código 24. Encriptación de contraseña con `bcrypt` durante el registro.

Este método aplica un algoritmo de hash con 10 rondas de sal, generando una versión cifrada no reversible de la contraseña.

Para validar las credenciales en el inicio de sesión, bcrypt compara la contraseña ingresada con el hash almacenado:

```
1 const isMatch = await bcrypt.compare(password, user.contrasena);
```

Código 25. Comparación de la contraseña ingresada con la almacenada.

Si ambas coinciden, se permite el acceso al sistema, garantizando que solo usuarios autenticados puedan ingresar.

---

# CAPÍTULO 4

---

## Conclusiones

El presente proyecto permitió el desarrollo de una plataforma integral para la automatización del entrenamiento de clasificadores de audio cardiaco, dirigida a profesionales de la salud que requieren actualizar sus herramientas diagnósticas sin conocimientos técnicos especializados. La solución propuesta integra tecnologías como *Node.js* para la implementación del backend, *MySQL* para la gestión relacional de datos, y bibliotecas específicas para el tratamiento de archivos de audio, logrando así un sistema escalable, seguro y de fácil implementación.

Se estableció un flujo de trabajo automatizado que ejecuta las etapas esenciales del entrenamiento sin intervención manual, lo que garantiza la reproducibilidad y fiabilidad del proceso. La implementación de una interfaz gráfica amigable permite al usuario cargar señales cardíacas y seleccionar parámetros de reentrenamiento, cumpliendo con los principios de accesibilidad y usabilidad definidos desde el inicio del proyecto.

Asimismo, se diseñó una base de datos relacional que asegura la integridad de la información y facilita la trazabilidad de los datos de usuario y los archivos de audio cargados. A través de la implementación de rutas seguras y procedimientos de autenticación, se reforzó la seguridad del sistema y se protegieron los datos sensibles. La incorporación del algoritmo *bcrypt* para el almacenamiento de contraseñas en formato hash es un ejemplo de las medidas adoptadas para fortalecer la privacidad de la información.

Durante el desarrollo, se realizaron pruebas funcionales que permitieron validar el comportamiento del sistema ante distintos escenarios, ajustando parámetros del proceso de entrenamiento cuando fue necesario. Si bien no se alcanzó una validación clínica formal, los resultados obtenidos sentaron las bases para futuras iteraciones del sistema en contextos reales.

En cuanto al trabajo futuro, se recomienda incorporar capacidades de entrenamiento

en la nube, permitir el manejo de conjuntos de datos más amplios, y realizar pruebas en entornos clínicos para evaluar su impacto real en el diagnóstico médico. Además, se podrían integrar métricas de rendimiento del modelo y visualizaciones que ayuden al usuario final a interpretar los resultados del clasificador.

Finalmente, este proyecto permitió al estudiante aplicar competencias específicas como el diseño e implementación de bases de datos relacionales, la programación de servidores backend, la gestión de interfaces web y el uso de herramientas de versionado y despliegue. Al mismo tiempo, se desarrollaron competencias genéricas como el trabajo colaborativo, la resolución de problemas complejos, la capacidad de autogestión, y la comunicación efectiva en entornos interdisciplinarios.

## 4.1. Trabajo a futuro

A partir de los resultados obtenidos y de las áreas de mejora identificadas, se proponen las siguientes líneas de trabajo futuro:

- Incorporar gráficos interactivos, visualización de espectrogramas, análisis de frecuencia y otros indicadores biomédicos para facilitar la interpretación de las señales cardíacas por parte de los usuarios, mejorando así la experiencia de uso y la toma de decisiones clínicas.
- Permitir la creación de diferentes perfiles (administradores, médicos, estudiantes, etc.), cada uno con permisos específicos, para mejorar la gestión de accesos, la seguridad de los datos y la organización de la información médica sensible.

Estos avances sientan las bases para el desarrollo de soluciones tecnológicas accesibles que apoyen el diagnóstico médico y promuevan la integración de la inteligencia artificial en entornos clínicos reales.

---

# Bibliografía

- [1] A. Pacheco. M. L. González. Desarrollo de la telesalud en México. *CEPAL*, 2019.
- [2] Instituto Tecnológico de Tijuana. Misión, visión y valores. <https://www.tijuana.tecnm.mx/>.
- [3] Jose Antonio Prados Castillejo. Telemedicina, una herramienta también para el médico de familia. *Atención Primaria*, 45(3):129–132, 2018.
- [4] Secretaría de Salud. Avanza el uso de la telesalud o telemedicina en México. *Gobierno de México*, 2015.
- [5] P. Mayorga-Ortiz, J. A. Valdez-Gonzalez, C. Druzgalski, and V. Zeljkovic. Automatic detection and classification of cardiopulmonary events. *Revista Mexicana de Ingeniería Biomedica*, 39(1):65–80, Jan. 2018.
- [6] Miguel A. Becerra-Botero, Cristian Mejía-Arboleda, Milton Sarria-Paja, and Leonardo Duque-Muñoz. Análisis de clasificadores estocásticos para la detección efectiva de soplos cardíacos a partir de señales fonocardiográficas. *Revista de Investigaciones Universidad del Quindío*, 23(1):8–15, ago. 2022.
- [7] La importancia de las bases de datos para el entrenamiento en inteligencia artificial. *Revista Peruana de Investigación en Salud*, 7(3):121–122, Sep. 2023.
- [8] S. Álvarez I. E. Peón. F. S. Vela, J. R. Espinoza. Retos de ehealth en las zonas rurales en México y propuesta de innovación socio-técnica. *XVIII Congreso Nacional De Ingeniería Electromecánica y de Sistemas (CNIES)*, 2019.
- [9] INEGI. Estadística de defunciones registradas de enero a junio de 2022. *Comunicado de prensa No. 29*, 2023.
- [10] D. Garrido. Sistema cardiovascular: Desde el embrión al anciano. *RCA Grupo Editor*, VI(61):4–29, 2023.

- [11] Mario San Mauro. *Anatomía cardíaca. Una manera integral de estudiar las estructuras del corazón y los grandes vasos*. Editorial de la Universidad de La Plata, 2013.
- [12] J. Sánchez-Prieto Castillo and F.A. López Sánchez. Insuficiencia cardíaca. generalidades. *Medicine - Programa de Formación Médica Continuada Acreditado*, 12(35):2085–2091, 2017.
- [13] Hugo Alberto Cruz Ortega and Francisco Xavier Calderón Monter. El corazón y sus ruidos cardíacos normales y agregados. una somera revisión del tema. *Revista de la Facultad de Medicina (México)*, 59(2):49–55, 2016.
- [14] José F. Guadalajara. La auscultación del corazón, un arte en vías de extinción. *Gaceta médica de México*, 151(2):260–265, 2017.
- [15] Anzueto A. Rios R. Tovar, B. Auscultación cardiaca y fonocardiografía. *Instituto Politécnico Nacional*, 2020.
- [16] R. Tamariz-Martel Moreno. Auscultación cardiaca. *Sociedad Española de Pediatría Extrahospitalaria y Atención Primaria*, XX(8):560, 2016.
- [17] Makaryus AN. Thomas SL, Heaton J. Cardiovascular murmurs. *National Library of Medicine*, 2023.
- [18] Jorge Díaz-Ramírez. Aprendizaje automático y aprendizaje profundo. *Ingeniare. Revista chilena de ingeniería*, 29(2):180–181, 2021.
- [19] Marianella Álvarez Vega, Laura María Quirós Mora, and Mónica Valeria Cortés Badilla. Inteligencia artificial y aprendizaje automático en medicina. *Revista médica sinergia*, 5(8):11, 2020.
- [20] René Tobar-Díaz, Yan Gao, Jean François Mas, and Víctor Hugo Cambrón-Sandoval. Classification of land use and land cover through machine learning algorithms: a literature review. *Revista de Teledetección*, (62):1–19, Jul. 2023.

- [21] Mariana Belgiu and Lucian Drăguț. Random forest in remote sensing: A review of applications and future directions. *ISPRS journal of photogrammetry and remote sensing*, 114:24–31, 2016.
- [22] Courage Kamusoko and Courage Kamusoko. Remote sensing digital image processing in r. *Remote sensing image classification in R*, pages 1–24, 2019.
- [23] Thomas Lillesand, Ralph W Kiefer, and Jonathan Chipman. *Remote sensing and image interpretation*. John Wiley & Sons, 2015.
- [24] R. S. Olson and J. H. Moore. Tpot: a tree-based pipeline optimization tool for automating machine learning. *The Springer Series on Challenges in Machine Learning*, pages 151–160, 2019.
- [25] GeeksforGeeks. Tpot automl. <https://www.geeksforgeeks.org/tpot-automl/>. Accessed: May 16, 2025.
- [26] Pablo Enrique Fernández Casado. *Construcción y diseño de páginas web con HTML, CSS y JavaScript. Edición 2023*. Ra-Ma Editorial, 2023.
- [27] Alicia Durango. *Diseño Web con CSS: 2ª Edición*. IT Campus Academy, 2015.
- [28] Javier Eguíluz Pérez. introducción a javascript, 2019.
- [29] Hezbullah Shah and Tariq Rahim Soomro. Node. js challenges in implementation. *Global Journal of Computer Science and Technology*, 17(2):73–83, 2017.
- [30] Rafael Camps Paré, Luis Alberto Casillas Santillán, Dolors Costal, Marc Gibert Ginestà, Carme Martín Escofet, and Óscar Pérez Mora. Bases de datos, febrero 2007, 2010.

## **Anexo**

Link de Github al repositorio del proyecto: <https://github.com/chilakilll/Sistema-de-entrenamiento-para-clasificador-de-audio-card-aco>