Arquitectura para apoyo al diagnóstico de Leishmaniasis en Dispositivos de Bajo Procesamiento y Ambientes de Conectividad Limitada

Juan Miguel Gomez Ganem

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
MAESTRÍA EN INGENIERÍA DE DE SISTEMAS Y COMPUTACIÓN
BOGOTÁ, D.C.
2016

Arquitectura para apoyo al diagnóstico de Leishmaniasis en Dispositivos de Bajo Procesamiento y Ambientes de Conectividad Limitada

Autor:

Juan Miguel Gomez Ganem

MEMORIA DEL TRABAJO DE GRADO REALIZADO PARA CUMPLIR UNO DE LOS REQUISITOS PARA OPTAR AL TITULO DE

MAGÍSTER EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Directora

Andrea Del Pilar Rueda Olarte

Comité de Evaluación del Trabajo de Grado

Mariela Josefina Curiel Huérfano

Oscar Danilo Martinez Bernal

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
MAESTRÍA EN INGENIERIA DE SISTEMAS Y COMPUTACIÓN
BOGOTÁ, D.C.
Noviembre, 2024

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
MAESTRÍA EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Rector Magnífico

Jorge Humberto Peláez, S.J.

Decano Facultad de Ingeniería

Ingeniero Lope Hugo Barrero Solano

Director Maestría en Ingeniería de Sistemas y Computación

Ingeniera Mariela Josefina Curiel Huérfano

Director Departamento de Ingeniería de Sistemas

Ingeniero Cesar Julio Bustacara Medina

Artículo 23 de la Resolución No. 1 de Junio de 1946

"La Universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado. Sólo velará porque no se publique nada contrario al dogma y la moral católica y porque no contengan ataques o polémicas puramente personales. Antes bien, que se vean en ellos el anhelo de buscar la verdad y la Justicia"

AGRADECIMIENTOS

A mi querida esposa,

Su apoyo y compañía durante todo mi proceso académico me motivo a continuar con el estudio de la maestría. Fue gracias a ella que pude persistir en este proceso y no abandonarlo. Es gracias a su apoyo que continue con la maestría a pesar de múltiples veces considerar no seguir. Tendrás siempre mi más sincera gratitud y mis logros los hago por ti y nuestro futuro.

A la Ingeniera Andrea Rueda,

El mas sincero agradecimiento. Por todos los consejos y el apoyo durante esta tesis. También agradezco su infinita paciencia y agradezco mucho el tiempo que invirtió en apoyarme y asegurarse que llevase este proyecto a su final. Le agradezco un montón las palabras de alentó que ofreció en un momento critico del tumultuoso y complejo camino que fue llegar aquí. Gracias por todo profe.

CONTENIDO

ABSTRACT

Leishmaniasis is a neglected tropical disease caused by the leishmania parasite. Parasitological methods remain as the "gold standard" for diagnosis of the disease. There are many advances in medical image processing and artificial intelligence that could be of relevance to help in the diagnosis of this disease. Especially given the context where it is most likely to be found, which are areas far from medical institutions capable of running the procedure. A tool that could bring the diagnosis to remote places using modern easily accessible technology could be desired. Here, we propose an architecture, oriented at solving the issues that Colombia presents: the lack of conectivity and limited tecnological resources, and validate it via Micro Tracker, a progressive web application that implements the architecture.

RESUMEN

La Leishmaniasis es una enfermedad tropical descuidada, ocasionada por el parasito de la leishmania. Los métodos parasitológicos son el "estándar de oro" para el diagnóstico de la enfermedad. Existen una multitud de avances en procesamiento de imágenes médicas y en inteligencia artificial que podrían ser relevantes en el apoyo al diagnóstico de la enfermedad. Son aún más relevantes dado al contexto en el cual esta enfermedad es más comúnmente presentada. Suelen ser áreas lejanas de instituciones médicas capaces de llevar a cabo el procedimiento. Una herramienta que lleve el diagnostico a sitios remotos, usando tecnología de fácil acceso, puede ser deseable. Aquí, proponemos una arquitectura orientada a resolver los problemas que presenta el contexto colombiano: la falta de conectividad y la falta de acceso a dispositivos de alto rendimiento, y lo validamos a través de Micro Tracker, una "progressive web app" que implementa dicha arquitectura.

RESUMEN EJECUTIVO

La leishmaniasis es una enfermedad tropical descuidada [4], ocasionada por el parasito de la leishmania. Este parasito se transmite por medio de mosquitos de arena el cual prolifera en hábitats tropicales.

El diagnóstico de la leishmaniasis generalmente se realiza por medio de imágenes de microscopia en las cuales se puede detectar la presencia del parasito. Este trabajo es llevado a cabo por un médico y brinda oportunidad para que el error humano interfiera.

La Leishmaniasis puede ser letal cuando se presentan coinfecciones. Debido al ambiente común en donde se presenta la enfermedad, el diagnostico puede demorarse. Es por eso que una herramienta que agilice el proceso puede ser deseable.

Basado en esto, se hizo un análisis de las herramientas disponibles para el apoyo al diagnostico de la leishmaniasis cutánea con el objetivo de evaluar su compatibilidad con el contexto colombiano y proponer una arquitectura que se adecue más.

Las herramientas vistas en el análisis presentan ciertas limitantes que evitan su uso en el contexto colombiano. En particular, tienden a no considerar su uso por fuera de un computador de escritorio y no evalúan la posibilidad de funcionamiento sin conectividad.

Leishmaniapp es un proyecto previo de la Pontificia Universidad Javeriana en el que se elaboró un sistema que apoya el diagnóstico de la leishmaniasis usando imágenes de microscopia. Este proyecto es compuesto por una aplicación nativa que se conecta a un sistema en la nube. Adicionalmente, para el proyecto de Leishmaniapp se elaboraron múltiples modelos de inteligencia artificial debido a su enfoque de permitir el uso de múltiples modelos para diferentes enfermedades. Este presenta la dificultad de ser dependiente de acceso a conexión a internet. Aunque capaz de diferir el procesamiento para un momento posterior, no es capaz de procesar las imágenes de manera local.

Dentro de los modelos elaborados como parte de Leishmaniapp, se encuentra un modelo basado en máquinas de vectores de soporte, orientado a la detección de macrófagos en imágenes de microscopia con tintura Giemsa. Este se usó para la implementación de una “progressive web app” capaz de llevar a cabo el procesamiento de las imágenes de microscopia de manera local, denominada Micro Tracker. Esta PWA no solo procesa las imágenes si no que es capaz también de actuar como sistema de almacenamiento local de imágenes y resultados de procesamiento.

Micro Tracker demuestra la capacidad de llevar el procesamiento de Leishmaniapp de manera local y cumple con los objetivos de permitir el procesamiento sin necesidad de conectividad y de permitir estos en dispositivos móviles. La aplicación sigue siendo una prueba de concepto y para su exitoso uso por los usuarios finales es importante considerar la implementación completa de las funcionalidades descritas en este documento. También se considera importante hacer una evaluación de la aplicación por parte de los usuarios objetivo.

INTRODUCCIÓN

La leishmaniasis es una enfermedad parasitiva causada por el parasito de la leishmania. Este se presenta en regiones subtropicales y tropicales, en ambientes rurales donde hay presencia de micro hábitats que son favorables para el descanso y la reproducción de los mosquitos de arena que transmiten el parasito. La Leishmaniasis no suele darse en zonas urbanas, donde las condiciones para la supervivencia del mosquito de arena son mucho más escasas y, adicionalmente, la presencia de mayor acceso a instituciones clínicas y hospitales aumenta la posibilidad de detección temprana de la Leishmaniasis, disminuyendo el impacto de este.

Existen múltiples métodos de diagnóstico de Leishmaniasis. Los métodos considerados "el estándar de oro" para su detección son los métodos parasitológicos. Este proyecto se enfocará en el método parasitológico de examinación microscópica debido a que es el más común y su uso es extendido a muchas partes del mundo. Este se puede considerar como la primera línea de diagnóstico y no requiere de equipo altamente especializado, como en otras categorías de diagnóstico. Este método por examinación microscópica es un proceso manual que requiere varios pasos para un diagnóstico, siendo una técnica tediosa y con presencia de error humano aún en manos expertas [4], por esto sería lógico el uso de algún sistema computacional usado como apoyo al diagnóstico

El presente trabajo propone una arquitectura para llevar modelos de inteligencia artificial a procesamiento local, disminuyendo la dependencia con la conectividad, y permitiendo su uso dentro del contexto donde se da más comúnmente la leishmaniasis.

Se hizo un análisis del estado actual de las herramientas orientadas al procesamiento de imágenes medicas para el apoyo al diagnostico de enfermedades, para así proponer una solución que se adecue más al contexto colombiano.

Para demostrar la viabilidad de la arquitectura propuesta, se incluye una implementación de una prueba de concepto, llamada Micro Tracker. Esta es una aplicación web progresiva, que es capaz de almacenar y procesar imágenes de microscopia, sin necesidad de conectividad y haciendo uso de un modelo de IA capaz de detectar la presencia de macrófagos en imágenes de muestras de microscopia.

Incluido también esta una descripción general de los avances técnicos en el campo del desarrollo web que hicieron posibles el desarrollo de Micro-Tracker. junto con las necesidades específicas que suplen dentro del marco del contexto colombiano.

En este documento se encuentra una descripción detallada de la implementación de Micro-Tracker, donde se podrá evidenciar la relación entre las decisiones técnicas y la solución final y de como se valido el funcionamiento correcto de la implementación.

Finalmente, se propone posibles caminos por los cuales se sugiere que este proyecto se continue, caminos que siguen el objetivo de proveer un sistema que apoye al tratamiento de la Leishmaniasis.

1. DESCRIPCIÓN GENERAL

Oportunidad y problemática

La leishmaniasis es una enfermedad parasítica causada por el parásito de la leishmania. Este se encuentra en regiones subtropicales y tropicales, afectando cerca de 97 países en América, África, Asia y Europa[1] . La infección se transmite a través de picaduras del mosquito de arena. La presencia de Leishmaniasis en zonas tropicales se favorece por la mayor presencia de espacios como letrinas interiores, bodegas, establos, cuevas, fisuras en paredes, piedras o en el suelo, vegetación densa, huecos en árboles, madrigueras de roedores y de otros mamíferos y nidos de aves; micro hábitats que son favorables para el descanso y la reproducción de los mosquitos de arena [2]. Es por esto que la Leishmaniasis no suele darse en zonas urbanas, donde las condiciones para la supervivencia del mosquito de arena son mucho más escasas y, adicionalmente, la presencia de mayor acceso a instituciones clínicas y hospitales aumenta la posibilidad de detección temprana de la Leishmaniasis, disminuyendo el impacto del mismo.

La Leishmaniasis tiene múltiples métodos de diagnóstico. El diagnóstico convencional consta de la examinación microscópica de amastigotes en aspirados de tejidos provenientes de diferentes órganos como el bazo, los ganglios linfáticos, el hígado, la piel o también puede ser llevado a cabo haciendo un cultivo parasitológico de estas ubicaciones[3]. Estos procesos tienen desventajas claves: el método de aspirado puede ser incómodo para el paciente, mientras que el método de cultivo puede ser lento y costoso.

Los métodos parasitológicos son considerados “el estándar de oro” para el diagnóstico de la Leishmaniasis. Este consta de la demostración de la existencia de parásitos en preparaciones teñidas de médula ósea o aspirados esplénicos [3]. Sin embargo, los métodos parasitológicos presentan un alto riesgo de ocasionar hemorragias si son llevados a cabo por personal no capacitado. Esto se debe a la necesidad de realizar biopsias para tomar la muestra.

Este proyecto se enfocará en el método parasitológico de examinación microscópica debido a que es el más común y su uso es extendido a muchas partes del mundo. Este se puede considerar como la primera línea de diagnóstico y no requiere de equipo altamente especializado, como en otras categorías de diagnóstico. Adicionalmente, el método por examinación microscópica es un proceso manual que requiere varios pasos para un diagnóstico, siendo una técnica tediosa y con presencia de error humano aún en manos expertas [4], por esto sería lógico el uso de algún sistema computacional usado como apoyo al diagnóstico.

Como se puede evidenciar en C. Gonçalves et al. [5], el problema del diagnóstico ha sido abordado en numerosas ocasiones desde diferentes ángulos. Se presentan mayormente métodos basados en visión computacional. El uso de métodos basados en aprendizaje profundo, orientado al diagnóstico de Leishmaniasis en particular, es limitado debido a la reducida cantidad de datos que se encuentra disponible de manera pública.

Una gran variedad de técnicas se han utilizado para lograr apoyar el diagnóstico, como lo son el uso de umbralización por método de Otsu y máquinas de soporte de vectores (SVM), modelos de mezcla gaussiana, combinación de uso de diferencia gaussiana y método de umbrali-

zación de Otsu, agrupamiento por k-medios, conjunto de nivel de Chan-Vese, algoritmo de watershed, redes neuronales convolucionales usando la arquitectura de U-Net, crecimiento de regiones, uso de filtros de difusión isotrópica, y uso de morfología matemática [5]. En la mayoría de métodos, aunque se aborda directamente el diagnóstico, cabe destacar a Isaza et al. [6] debido a que toma en cuenta el contexto colombiano y propone el uso de dispositivos móviles para llevar a cabo el proceso. Sin embargo, no usan técnicas de aprendizaje de máquina y llevan a cabo el procedimiento en computadores de escritorio.

También es pertinente tomar en cuenta avaneces en el diagnóstico de enfermedades similares, como la malaria. Un ejemplo de estos es el sistema realizado por Dallet et al. [7] que es un intento de implementar un algoritmo que diagnostique la malaria, orientado a ser usado en dispositivos móviles, más específicamente, en celulares Android. Aquí se demuestra la viabilidad de celulares como unidades de procesamiento para llevar a cabo procesos de análisis gráfico de imágenes de microscopía. Sin embargo, este acercamiento no hace uso de modelos de inteligencia artificial.

Por la región donde se presenta, el médico que va a llevar a cabo el diagnóstico podría toparse con la dificultad de no tener acceso al equipo necesario para llevar a cabo el proceso, lo cual obliga a tener que tomar las muestras y verse obligado a transportarlas a zonas urbanas. De esta forma, sería deseable poder llevar a cabo el diagnóstico directamente en la zona donde se encuentra el paciente. Es por esto que en este proyecto se propone implementar una arquitectura para el uso de modelos de inteligencia artificial orientada a apoyar el diagnóstico de leishmaniasis usando imágenes de microscopía, en dispositivos de procesamiento limitado y en ambientes de baja conectividad.

2. DESCRIPCIÓN DEL PROYECTO

2.1. Objetivo general

Implementar una arquitectura para el uso de modelos de inteligencia artificial orientada a apoyar el diagnóstico de leishmaniasis usando imágenes de microscopía, en dispositivos de procesamiento limitado y en ambientes de baja conectividad.

2.2 Objetivos específicos

1. Elaborar estado del arte con respecto a los sistemas tecnológicos orientados a apoyar al diagnóstico de la leishmaniasis.
2. Evaluar arquitecturas existentes para determinar sus respectivas características y nivel de aplicabilidad.
3. Diseñar e implementar un prototipo de arquitectura alterna a las evaluadas.
4. Validar la implementación de la arquitectura, estableciendo métricas adecuadas de comparación y rendimiento.

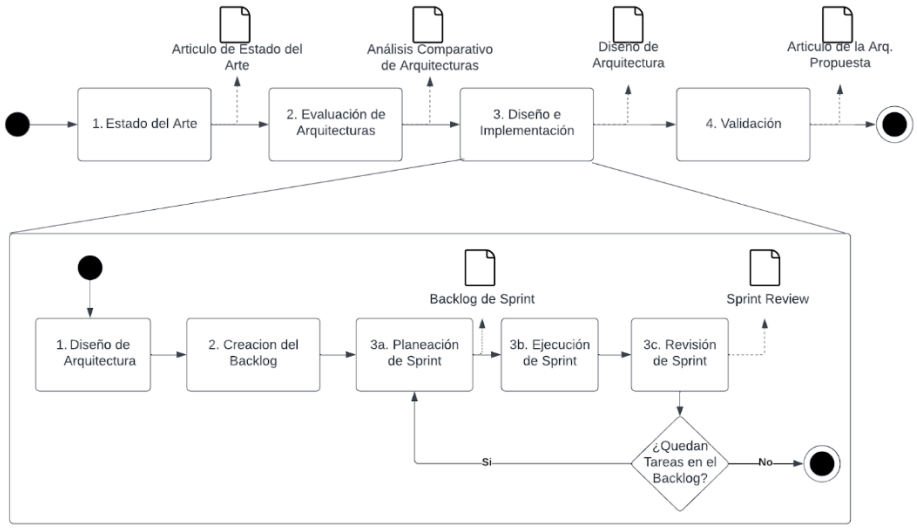


Figura 1: Fases del Proyecto

2.3 Fases de desarrollo

El proyecto se dividirá en 4 fases:
1. Estado del Arte
2. Evaluación de Arquitecturas
3. Diseño e Implementación
4. Validación

Cada fase está orientada a cumplir un objetivo específico, detallado en los objetivos del proyecto. La primera fase consta de elaborar un estado del arte para hallar que arquitecturas existen actualmente que permitan o apoyen el diagnóstico de la leishmaniasis, usando inteligencia artificial. Esto se usará como insumo para la segunda fase que trata de filtrar y evaluar las arquitecturas, buscando seleccionar aquellas que sean aplicables para el caso particular de este proyecto y compararlas en términos de aplicabilidad. A partir de la evaluación anterior, se hará el diseño e implementación de una arquitectura alterna, que busca cumplir de manera más adecuada los requerimientos específicos del proyecto. Por último, se validará la implementación por medio de una comparación contra al menos una de las arquitecturas evaluadas anteriormente.
Las fases 1 y 2 se llevarán a cabo en la asignatura MISyC proyecto 1, mientras que las fases 3 y 4 se llevarán a cabo en la asignatura MISyC proyecto 2.

Fase 1: Estado del Arte

El propósito de la fase fue elaborar un estado del arte con respecto a sistemas actuales, enfocados al apoyo del diagnóstico de la leishmaniasis o enfermedades que usen métodos similares, como la malaria.

Se realizaron las siguientes actividades:

1. Búsqueda de enfermedades con diagnósticos similares
2. Búsqueda de artículos asociados a sistemas para el apoyo
3. Selección de artículos relevantes.
4. Elaboración de articulo de estado del arte

El estado del arte fue enfocado a propuestas donde el diagnóstico sea basado en imágenes de microscopia. También, se tiene un enfoque a la arquitectura de las soluciones, por encima de su aplicación. Esto es para ser capaces de evaluar su relevancia para el contexto especifico y objetivo de este proyecto.

El entregable resultante de esta fase es el estado del arte.

Fase 2: Evaluación de Arquitecturas

Usando el estado del arte de la fase anterior, se hizo una descripción de cada una de las arquitecturas encontradas. Esta descripción se hará en términos de funcionalidades principales. En base a esas descripciones, se evaluaron que arquitecturas son aplicables para el caso específico de este proyecto. A cada arquitectura se le asoció un nivel de aplicabilidad basado en la cantidad de requerimientos de este proyecto que cumple la arquitectura evaluada. Estos requerimientos usados fueron: Uso de modelos de inteligencia artificial, permitir el uso de diferentes modelos de inteligencia artificial, funcionamiento con baja o sin conectividad al internet, funcionamiento en dispositivos de bajo rendimiento, existencia de implementación. Adicionalmente, se ofrece una descripción de cada arquitectura y listado de funcionalidades principales y descripción corta de casos de uso de cada una.

Las actividades que se llevaron a cabo son:

1. Listado de arquitecturas encontradas en el estado del arte
2. Caracterización de arquitecturas
3. Evaluación de arquitecturas

El resultado de esta fase fue un documento donde se podrá detallar y comparar las arquitecturas, para facilitar la búsqueda de vacíos en las mismas. Este resultado se puede revisar a detalle en el Anexo 1: Estado del Arte. Un resumen se encuentra en este documento en la Sección 4. Estado del Arte

Fase 3: Diseño e Implementación

La comparación de las arquitecturas identificadas se usa como insumo para el proceso de levantamiento de requerimientos. Para el diseño e implementación se llevarán a cabo las siguientes actividades:

1. Diseño de arquitectura propuesta
2. Implementación de arquitectura propuesta

La actividad de diseño se dará inicio con el levantamiento de requerimientos de la arquitectura. Posteriormente a tener el listado de requerimientos, se llevará a cabo un “sprint 0” cuyo único resultado es el backlog de tareas. Este sprint 0 tendrá una duración de una semana y marcará el inicio de las iteraciones de incremento del proyecto.

La implementación se llevará a cabo usando una versión modificada de Scrum [8]. Las modificaciones orientadas a ser acordes con el tamaño del equipo de desarrollo, que es el investigador mismo. Las modificaciones constan en reducir el número de ceremonias y de roles. No habrá un rol de scrum máster. El investigador actuara como equipo de desarrollo y el profesor asesor actuara como “Product Owner” (PO).

No habrá reuniones diarias. En cambio, se harán revisiones semanales de seguimiento entre el equipo y el PO. Los sprints tendrán una duración de 2 semanas, llevando a cabo una planeación de sprint al inicio de cada uno, donde se pondrán tareas del backlog para hacer durante el

sprint. Cada sprint tendrá asociado un documento de seguimiento (denominado "backlog de sprint") donde se detallará el resultado esperado del mismo y las tareas a ejecutar. Adicionalmente, al final de cada sprint se hará una revisión del sprint pasado, detallando en un documento el resultado del sprint (llamado "sprint review"). Se espera llevar a cabo un total de 7 sprints.

Todo el trabajo hecho deberá quedar dentro de un repositorio de git público, que estará disponible en github.

Los artefactos resultantes de esta fase serán los documentos de diseño de la arquitectura, la implementación de esta y todos los documentos de seguimiento del proceso.

Fase 4: Validación

Usando la implementación de la arquitectura, se ejecutan las siguientes tareas:

1. Especificación de pruebas
2. Ejecución de pruebas
3. Elaboración de articulo

Primero para iniciar esta fase, se definirá que pruebas se ejecutaran para evaluar la implementación de la arquitectura, para después ejecutar y registrar los resultados. Estas pruebas se deben llevar a cabo tanto con la arquitectura nueva, como con al menos una de las arquitecturas encontradas en la fase 2. Esto con el propósito de posteriormente comparar ambas arquitecturas. Usando el resultado de las pruebas, se elaborará un artículo que detalle tanto el trabajo efectuado en las fases 1-3, sino también el resultado obtenido de las pruebas realizadas en esta fase. Ese artículo será el resultado de esta fase.

3. MARCO TEÓRICO

Para la realización de "Micro Tracker", hay ciertas tecnologías que fueron criticas para que fuera posible. Estas demuestran avances importantes en las características que proveen los navegadores modernos. Es importante resaltar la necesidad de las siguientes tecnologías debido a la experiencia del investigador y las restricciones del proyecto.

ReactJS

También conocida como simplemente React, es una librería de JavaScript (JS), para la implementación de páginas web, basadas en componentes. React está brinda facilidad para crear un tipo especifico de páginas web: Single Page Aplications (SPA). Una SPA se distingue de una página web tradicional debido a su uso extensivo de JavaScript para modificar en el cliente la estructura de un único archivo HTML. Esto permite crear experiencias más cercanas a las que uno esperaría de una aplicación nativa.

React no solo es soportada en JavaScript, sino que también es soportada en otros lenguajes de programación, en particular tiene alto soporte en TypeScript (TS), un lenguaje de programación basado en JavaScript cuya propuesta de valor es brindarle a JavaScript un sistema de tipado estructural al lenguaje.

Effect-ts

Es una librería hecha en Typescript, orientada a proveer un mecanismo de concurrencia estructurada y un conjunto de estructuras de dato funcionales. Se basa alrededor de la idea de efectos: una estructura de datos que describe un cómputo en términos de dependencias y su posible error o éxito. Internamente incluye un runtime para uso de un sistema de Fibras para realizar tareas concurrentes. Las fibras son hilos virtuales que permiten concurrencia en un ambiente con un único hilo.

Adicionalmente, la librería permite lograr inyección de dependencias a través de un sistema de capas. A través del sistema de capas es posible definir unidades de negocio y dependencias entre esas unidades, usando el sistema de tipos de Typescript para garantizar la provisión de dependencias. Ese mecanismo de inyección de dependencias facilita la creación de pruebas unitarias por medio de la provisión de implementaciones de capas orientadas a pruebas.

En adición a las funcionalidades principales, existe un ecosistema de librerías adjuntas que proveen funcionalidades extra. Un ejemplo de estas en @effect/schema, un módulo para la definición de Esquemas que permiten tener una representación en ejecución de los tipos de dato definidos y garantiza la serialización y deserialización de los datos de este.

Progressive Web Application

Una "Progressive Web Application" (PWA), o aplicación web progresiva (AWP), nace a raíz de la necesidad de proveer experiencias web enriquecidas. La comunidad del desarrollo web venia creando sistemas cada vez más complejos para generar experiencias de usuario con más libertades. Las PWA son un avance que le permite a los desarrolladores crear aplicaciones web similares a una aplicación nativa, ofreciendo capacidades que no requieren de conectivi-

dad por medio del "Service Worker". Este se encargará de guardar en memoria caché los archivos descritos por el manifiesto, y de que cuando la aplicación web pida un recurso externo, atrapar la petición y enviar el recurso guardado en vez de ejecutar la petición HTTP de manera normal. Esto permite el uso de la aplicación sin necesidad de conectividad constante. Adicionalmente, las PWA vienen con un manifiesto que permite al usuario instalar la aplicación en sus dispositivos para así no necesitar conectividad para acceder a la aplicación. Esto se diferencia de simplemente descargar el HTML de una página web en el sentido que al instalar incluye todos los archivos necesarios para ejecutar la página y adicionalmente es posible habilitar una característica que actualice la versión de la página si es necesario. La instalación es opcional debido a que así no se instale, los archivos quedan en la memoria caché. La instalación solo es una funcionalidad que le permite al usuario tener un acceso directo a la PWA en su dispositivo.

Web Workers

JavaScript es, en su concepción, un lenguaje de programación mono-hilo. Eso quiere decir que por diseño es incapaz de lograr paralelismo. Permite asincronía, concurrencia y procesos no bloqueantes mediante un "Event Loop", un proceso que está constantemente verificando si hay tareas por ejecutar y cediendo control del hilo principal a las tareas que lo necesitan.

Con el tiempo, se vio la necesidad de ejecutar tareas en un hilo separado al hilo principal, con el objetivo de evitar sobrecargar este y mejorar la experiencia de usuario. Para ello se creó el concepto de "Web Workers". Los "Web Workers" permiten ejecutar archivos de JavaScript en hilos separados del hilo principal, con ciertas limitaciones. Los "Web Workers" ofrecen un mecanismo de comunicación con el hilo principal y otros "Web Workers" a traves de mensajes. En estos mensajes generalmente se copia el objeto a pasar, sin embargo, en ciertos casos como al enviar tramas de bytes, es posible transferir memoria. Esto último es crítico para lograr aplicaciones que procesen grandes cantidades de datos en segundo plano y evitar tener que duplicar la memoria reservada por el navegador.

WebAssembly

Es un tipo de código, basado en codigo de ensamblado, que puede correr en navegadores modernos. Es un lenguaje de bajo nivel transmitido en un formato binario compacto, que ha demostrado lograr un rendimiento semejante al visto en lenguajes como C/C++, C# y Rust. Es diseñado para ejecutar al lado de JavaScript.

WebAssembly fue pensado para ser un objetivo de compilación, lo que implica que la idea no era escribir directamente WebAssembly. Desde un inicio se pensó para ejecutar lenguajes diferentes a JavaScript dentro del navegador. Originalmente solo se soportaba compilar C/C++ a WebAssembly. Hoy en día ya se puede compilar la mayoria de lenguajes tienen algun nivel de soporte para compilar a WebAssembly. Particularmente en el caso de Python, existen múltiples proyectos pensados para ejecutar Python en el navegador, resaltando Pyodide que es un port de CPython compilado a WebAssembly. Este permite la ejecución de código de Python directamente en el navegador, incluso permitiendo el uso de librerías de terceros.

IndexedDB

Generalmente se piensa que una página web carece de estado, mucho menos posee la capacidad de lograr persistencia de datos. Por ello, normalmente se hace uso de un sistema separado para lograr dicha persistencia, normalmente denominado el “Backend”. Sin embargo, las páginas web no son totalmente incapaces de persistencia. El primer mecanismo de persistencia provisto fueron las cookies, pequeños tramos de data que son almacenados por el navegador. Inicialmente las cookies eran inaccesibles por JavaScript y solo el servidor podía manipular las cookies que manejaba el navegador. Sin embargo, eventualmente se habilitaron mecanismos para que un archivo de JavaScript pudiera acceder y manipular cookies propias, permitiendo cierto nivel de independencia del Backend. Eventualmente en vista de la necesidad de persistencia a nivel de página, los navegadores agregaron dos mecanismos de persistencia: almacenamiento local (Local Storage) y almacenamiento de sesión (Session Storage). Estos muy similares a las cookies, pero solo son visibles para la página web, no al servidor. Este almacenamiento, aunque útil y conveniente, es pensado para datos simples y pequeños. Debido a la ausencia de un mecanismo de almacenamiento para datos estructurados de tamaño significativo, los navegadores crean la “IndexedDB”. Esta es una base de datos orientada a objetos, basada en llave-valor, que permite almacenar grandes cantidades de información. Permite almacenar cualquier objeto creando por JavaScript y provee un sistema de transacciones. Esta es especialmente importante para las PWA que requieren persistencia de datos más allá de datos simples y que no tienen acceso a un Backend que los provea esa facilidad.

4. ESTADO DEL ARTE

La leishmaniasis se considera enfermedad desatendida [9]. Sin embargo, esto no evita que haya habido múltiples intentos previos para elaborar herramientas que podrían verse útiles para el apoyo del diagnóstico. Como parte de este proyecto se hizo un análisis de las herramientas que se consideraron prometedoras. Este análisis dentro del contexto del problema, evaluando su aplicabilidad en términos de conectividad, portabilidad y uso para imágenes de microscopia. Un listado general de las herramientas encontradas se puede ver en el Anexo 1: Estado del Arte. Se considera importante destacar los siguientes proyectos:

4.1 Slim

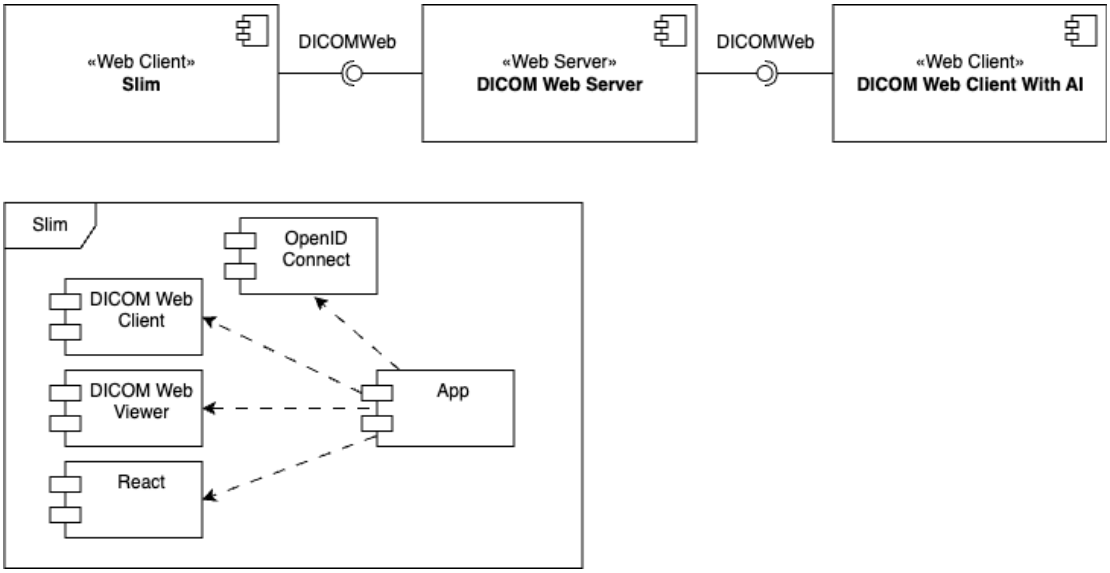


Figura 2: Arquitectura de Slim

Slim es una SPA orientada al procesamiento de imágenes DICOM en computadores de escritorio [10]. Sin embargo, no pretende funcionar en dispositivos móviles y aunque provee la capacidad de conectarse a un sistema de persistencia, no provee un mecanismo para lograr persistencia de manera local. Mas información del análisis detallado se puede revisar en el Anexo 2: Análisis de Arquitecturas

4.2 CellTrackViz

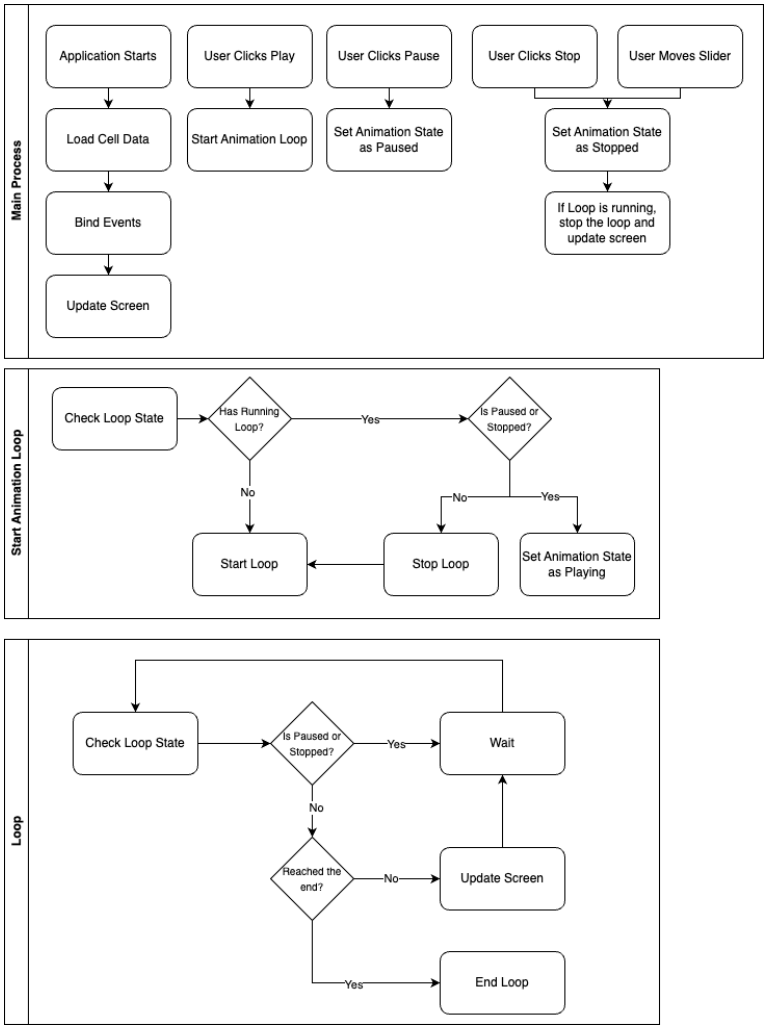


Figura 3: Diagrama de Flujo de CellTrackViz

CellTrackViz es una aplicación web orientada a ser ejecutada como una aplicación de escritorio. Su implementación es bastante sencilla y trivial. Consta de un único archivo HTML y un script de JavaScript que usa una librería de visualización de datos llamada D3 [11]. Aunque útil para el manejo de imágenes de células, no se adecua al contexto de este proyecto y el diseño de CellTrackViz dificulta su uso en otros contextos. Puede encontrar más acerca del análisis efectuado en Anexo 2: Análisis de Arquitecturas.

4.3 Leishmaniapp

Este proyecto se basa principalmente en el trabajo efectuado en el proyecto de grado titulado "Leishmaniapp: Implementación de Arquitectura para Aplicación Móvil como Herramienta de Detección de Leishmaniasis Cutánea y Otras Enfermedades Parasitarias" (en adelante referida como Leishmaniapp). Esta tenía como objetivo implementar una arquitectura que apoyara la detección de enfermedades parasitarias (…) usando tecnologías móviles y computación en la nube [12].

Leishmaniapp se compone de una aplicación móvil para Android, y de un componente que se encarga de la detección de macrófagos usando Inteligencia Artificial (IA). El componente de IA se despliega en la nube de Amazon usando los diferentes servicios de AWS. El modelo de IA esta desarrollado en Python usando OpenCV. Debido al uso de servicios en la nube, la aplicación móvil necesita conexión a internet para llevar a cabo el procesamiento de las imágenes. Cuando no tiene conectividad, alberga las peticiones de procesamiento localmente, a la espera de volver a tener conectividad para enviar las imágenes a procesar.

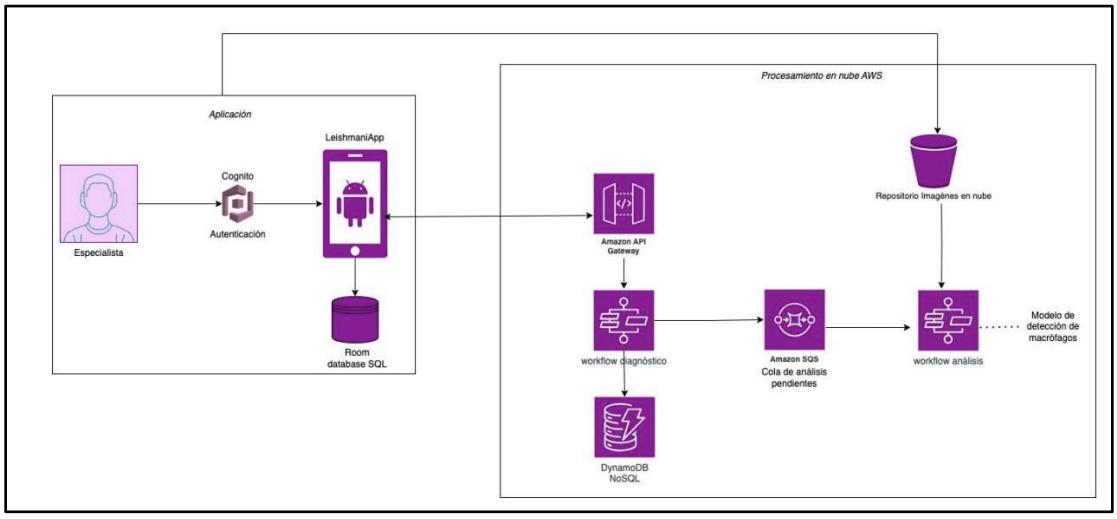


Figura 4: Arquitectura de Alto Nivel de Leishmaniapp (Tomado de [12])

El modelo IA fue desarrollado usando la metodología 90/10 donde 90% de los datos son usados para entrenamiento y 10% son usados para la validación. Para la validación del modelo de IA, en el proyecto de Leishmaniapp, se usaron las siguientes métricas:

- **MSE (error cuadratico medio)**: Para comparar dos imágenes se utiliza el error cuadrático medio de los valores de los píxeles de ambas imágenes. Imágenes similares tienen MSE más bajo. La utilización de este método aplica para imágenes de identificas dimensiones y canales, lo que lo hace ideal para el escenario de aplicación[12]
- **Índice de similaridad de Jaccard:** Es usado para determinar qué tan similar son las dos imágenes, los píxeles compartidos entre las dos entre el total de píxeles presentes en ambas imágenes, en otras palabras, la intersección sobre la unión del conjunto. Se prefirió esta métrica porque en el modelo, las imágenes ground truth tienen muchos

píxeles negros catalogados como fondo y una métrica diferente evaluaría erróneamente con un puntaje alto a una imagen de salida que contenga 100% de sus píxeles como fondo, en este caso negros[12]

- **Coeficiente de Sorensen-Dice:** Comparte comportamiento con el índice de similaridad de Jaccard en la medida en que, si el uno coincide en el desempeño, el otro también lo hará. Se define como el cociente de 2 veces la intersección de los píxeles compartidos por las dos imágenes entre la suma de los cardinales de ambos conjuntos, en este caso, la suma de la resolución de las dos imágenes. Al igual que Jaccard, tampoco da crédito por píxeles identificados como fondo [12].

El resultado de la validación del modelo usando las métricas descritas anteriormente se encuentra resumido por la siguiente tabla:

	1-MSE	SORENSEN	JACCARD
PROMEDIO	0,9540200459	0,8323911438	0,7407362532

Tabla 1: Metricas Pos-Analisis Validación de Modelo IA (Tomado de [12])

Este proyecto planea hacer uso extenso de este modelo para realizar el procesamiento de las imágenes. Sin embargo, no es posible hacer uso directo debido a la implementación concreta. Para eso, se hicieron adecuaciones al modelo desarrollado. Los detalles de la adecuación efectuada se encuentran en la sección **6.2 Integración de Modelo IA** de este documento.

Dentro de los trabajos futuros, Leishmaniapp identifica que migrar los modelos a un entorno local permitiría la utilización de esto sin conexión, ya que actualmente se encuentra ligadoo a un servicio backend provisto por AWS [10], y este proyecto en particular se enfoca en esta posibilidad. Uno de los objetivos principales de este trabajo de grado es proponer una arquitectura que permita llevar modelos de IA a dispositivos móviles, para así ser usados sin requerir un sistema externo y poder manejar todo el procesamiento incluso cuando no haya conexión.

5. ARQUITECTURA

5.1 Actores

Los actores que se consideran como posibles usuarios del sistemas son los siguientes:

Medico Experto	Usuario experto que usara el sistema para llevar a cabo diagnósticos. Debido a que es experto en el área, sería deseable que el sistema no solo de una respuesta, si no proveer la mayor cantidad de información relevante para el diagnóstico. Podría ser de utilidad albergar diferentes datos de entrada para su uso posterior.
Usuario General	Usuario con las funcionalidades básicas del sistema. Es capaz de mantener imágenes guardadas pero su vista es simplificada. Debe tener la capacidad de comunicarse con un Médico Experto para llevar a cabo el diagnostico.
Administrador	Usuario con permisos elevados que es capaz de acceder a datos de otros usuarios con el fin de brindar apoyo.

Tabla 2: Descripción de Actores

5.2 Requerimientos y Atributos de Calidad

Los siguientes atributos de calidad, descritos en la **Tabla 3: Atributos de Calidad**, son las características que el sistema deberá cumplir para lograr su objetivo a nivel arquitectónico. Estos atributos de calidad se basan en los siguientes requerimientos:

- **Requerimiento 1:** El sistema debe ser capaz de funcionar en dispositivos móviles
- **Requerimiento 2:** El sistema debe ser agnóstico a hardware
- **Requerimiento 3:** El sistema debe ser capaz de hacer procesamiento local y/o diferir el procesamiento a un momento en el futuro donde tenga conexión.
- **Requerimiento 4:** El sistema deberá proveer alguna funcionalidad de persistencia para almacenar las imágenes.
- **Requerimiento 5:** El sistema deberá manejar ciertos formatos de imagen y ser capaz de procesarlos para la transmisión.

Nombre de Cualidad	Descripción
Longevidad	Es el sistema capaz de albergar los datos de manera consistente hasta que el usuario decida que no los necesita.
Diferabilidad	El sistema es capaz de esperar a un momento en el futuro para ejecutar el procesamiento de los datos
Portabilidad	El sistema debe ser capaz de ser usado independiente de ubicación física
Distributilidad	El mecanismo de distribución del sistema debe ser simple en la medida de lo posible
Extensibilidad	El sistema debe ser capaz de procesar múltiples formatos de imagen y extender a nuevos formatos.

Tabla 3: Atributos de Calidad

5.3 Casos de Uso

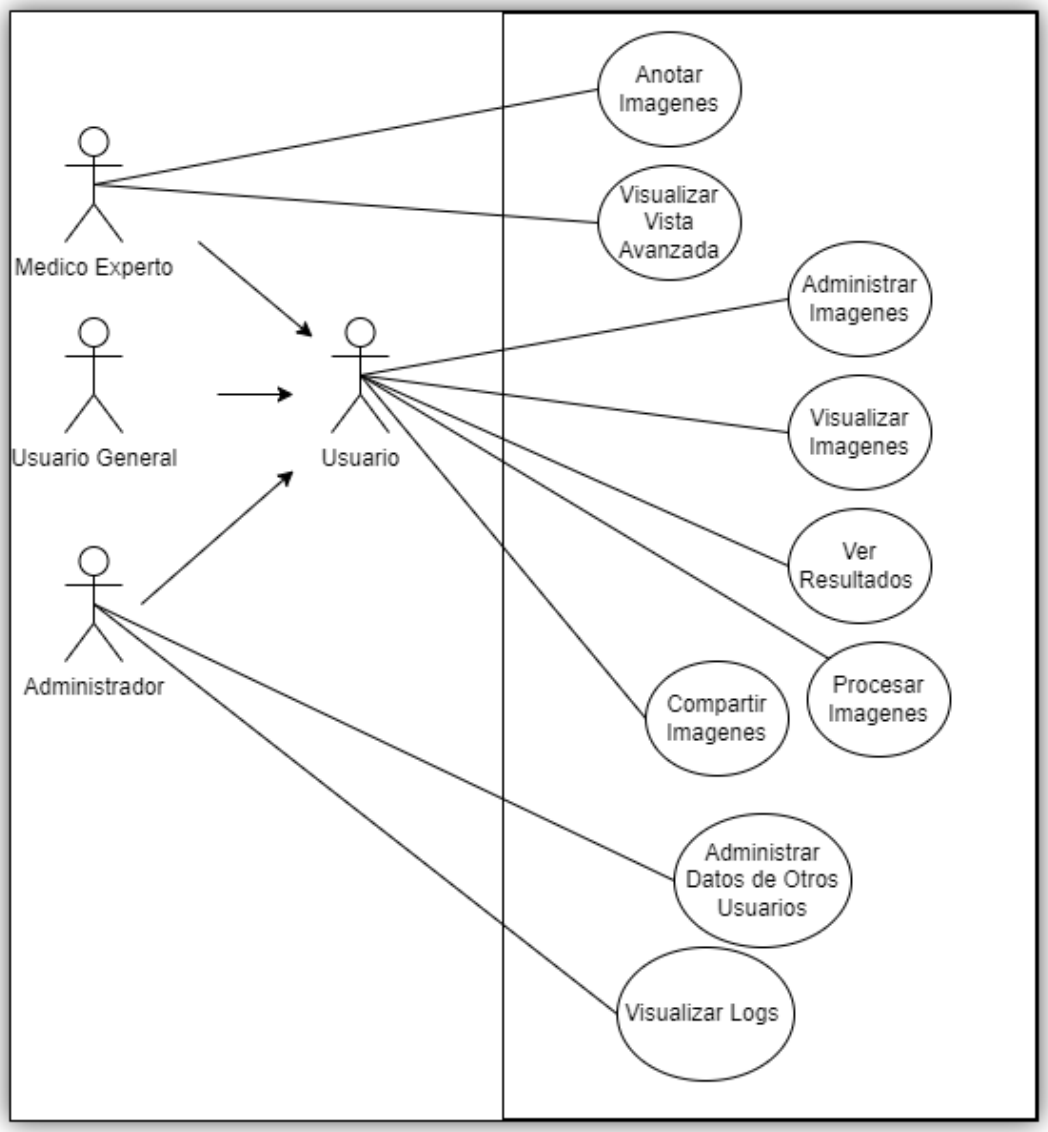


Figura 5: Casos de Uso Micro Tracker

Caso de Uso	Descripción
CU1: Anotar Imágenes	Como usuario medico experto quiero ser capaz de hacer comentarios sobre resultados de imágenes compartidas conmigo
CU2: Visualizar Vista Avanzada	Como usuario medico experto quiero ser capaz de visualizar una vista de alto detalle sobre el resultado de procesamiento de una imagen para tener mas contexto con respecto al mismo
CU3: Administrar Imágenes	Como cualquier usuario del sistema, debo ser capaz de agregar y borrar imágenes almacenadas, con sus resultados de procesamiento asociados
CU4: Visualizar Imágenes	Como cualquier usuario del sistema, debo ser capaz de visualizar imágenes almacenadas
CU5: Ver Resultados	Como cualquier usuario del sistema, debo ser capaz de ver los resultados de procesamiento de cualquier imagen a la que tenga acceso
CU6: Procesar Imágenes	Como cualquier usuario del sistema, debo ser capaz de iniciar el procesamiento de una imagen y visualizar el estado de procesamiento de la imagen. Múltiples resultados deben ser posibles para una imagen.
CU7: Compartir Imágenes	Como cualquier usuario del sistema, debo ser capaz de compartir imágenes a otros usuarios del sistema
CU8: Administrar Datos de Otros Usuarios	Como usuario administrador del sistema, debo ser capaz de visualizar las imágenes y los resultados de procesamiento de otros usuarios a los cuales se me ha dado permiso previo
CU9: Visualizar Logs	Como usuario administrador del sistema, debo ser capaz de ver mensajes emitidos en el funcionamiento del sistema.

Tabla 4: Descripción de Casos de Uso

Alcance del POC

Para la validación del sistema, se implementará una prueba de concepto (POC) con el propósito de evaluar la eficacia de la arquitectura. Para la elaboración del POC solo se tomará en cuenta los casos de uso CU3, CU4, CU5 y CU6 ya que estos son el flujo principal del sistema.

5.4 Listado de Tecnologías

La herramientas que fueron clave para el desarrollo del POC se encuentras descritas en la siguiente tabla (**Tabla 5: Tecnologias Clave usadas en POC**):

Nombre	Descripción
React	Librería para desarrollo de interfaces graficas de usuario en javascript
Typescript	Lenguaje de programación que se transpila a javascript. Usado para agregar un sistema de tipado estructural a javascript
Effect-ts	Librería de typescript para elaborar aplicaciones concurrentes y uso de estructuras de programación funcional
IndexedDB	Base de datos no relacional, incluida en los navegadores, para la persistencia de recursos de gran tamaño.
VSCode	Editor de texto orientado a desarrollo de software
Git	Un sistema de control de versión distribuido. Se va a usar para manejar el código fuente del POC
Github	Plataforma de hosting de repositorios de Git

Tabla 5: Tecnologías Clave usadas en POC

5.6 Arquitectura de Alto Nivel

Para modelar la arquitectura del sistema, se optó por usar el modelo C4, el cual consta de 4 diagramas con diferentes niveles de detalle, con el propósito de comunicar como el sistema va a proveer las diferentes funcionalidades. Dentro del modelo se definen los siguientes conceptos:

- **Nivel 1: Diagrama de Contexto:** Es el punto de inicio del sistema, enmarcando el alcance. Se muestra que componentes son externos y cuales se van a desarrollar. También se describe como el usuario interactuara con el sistema.
- **Nivel 2: Diagrama de Contenedor:** En este diagrama se describen los componentes principales del sistema. Describe la interacción entre los componentes principales.
- **Nivel 3: Diagrama de Componentes**: Ofrece una vista a elementos individuales que componen a los contenedores y las tecnologías asociadas en el caso de ser necesario.

- **Nivel 4: Diagrama de Código:** Es una vista detallada de la implementación del código. Este diagrama puede ser un diagrama de clases o cualquier diagrama necesario para la implementación del sistema

Debido al propósito de este documento, el diagrama de Nivel 4 no será provisto.

Nivel 1: Diagrama de Contexto

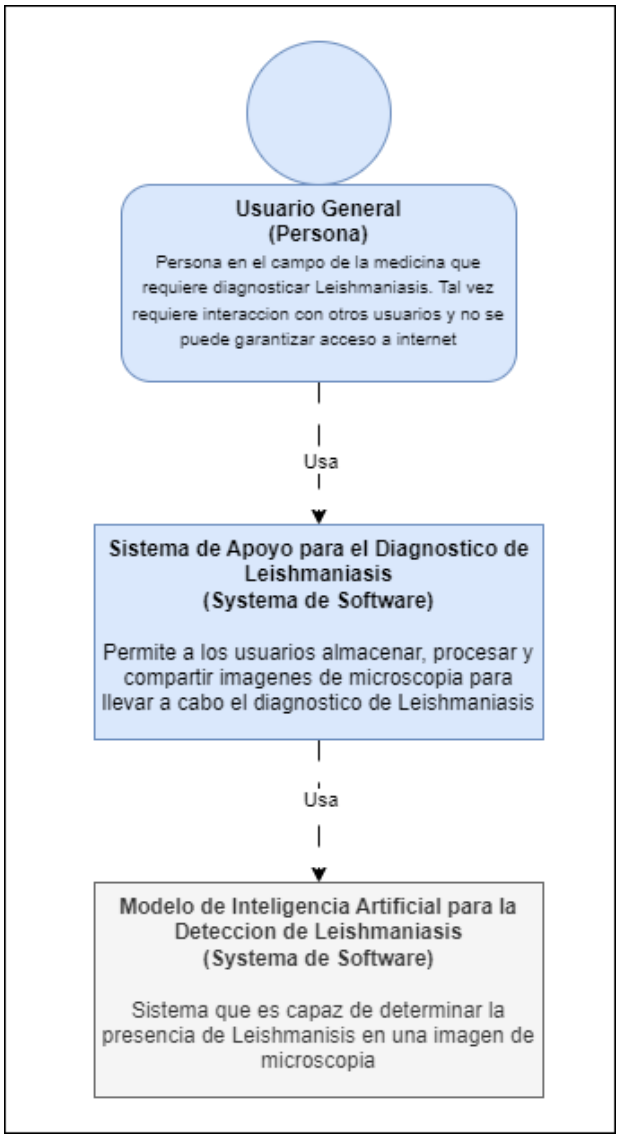


Figura 6: Diagrama de Contexto

En este diagrama se hace referencia al sistema a desarrollar y su relación con el modelo de inteligencia artificial. Los detalles de cómo el sistema se comunicará con el modelo se encuentran descritos en la **Sección 6.2 Integración con el Modelo IA**. El sistema actuara como

interfaz entre el Usuario y el Modelo IA. El sistema no debería depender de conectividad para hacer uso del Modelo IA. Sin embargo, si esto no es posible, proveer una alternativa para diferir el procesamiento a un momento futuro con conectividad.

Nivel 2: Diagrama de Contenedor

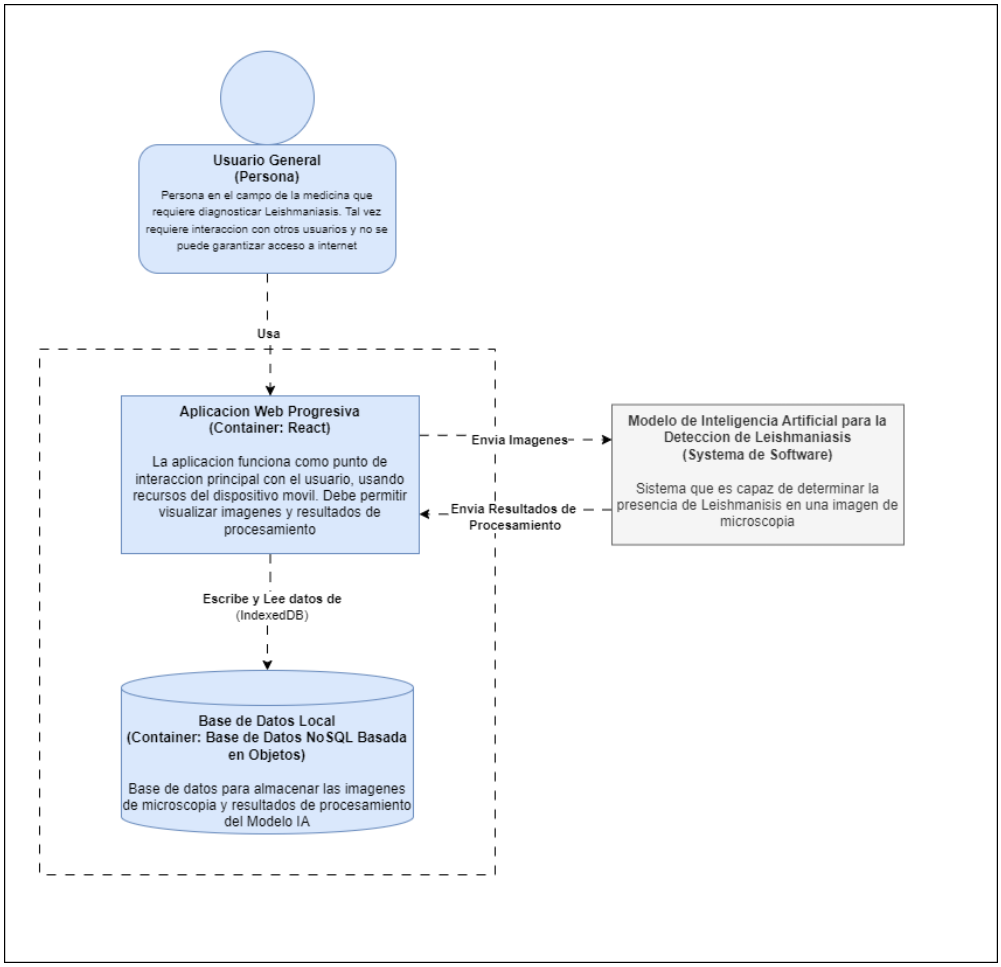


Figura 7: Diagrama de Contenedores

Los contenedores principales del sistema son: La aplicación web progresiva (AWP) y la base de datos. La AWP va a encargarse de la mayor parte de las funcionalidades y va a ser el punto de interacción con el usuario. La base de datos se encargará de la persistencia. Esta debe estar en el dispositivo móvil, no ser albergada por fuera de él. Esto para garantizar la disponibilidad de los datos a pesar de perdida de conectividad. Dentro del sistema, también estará incluido el modelo de inteligencia artificial que se encargará de llevar a cabo el procesamiento de las imágenes.

Nivel 3: Diagrama de Componentes

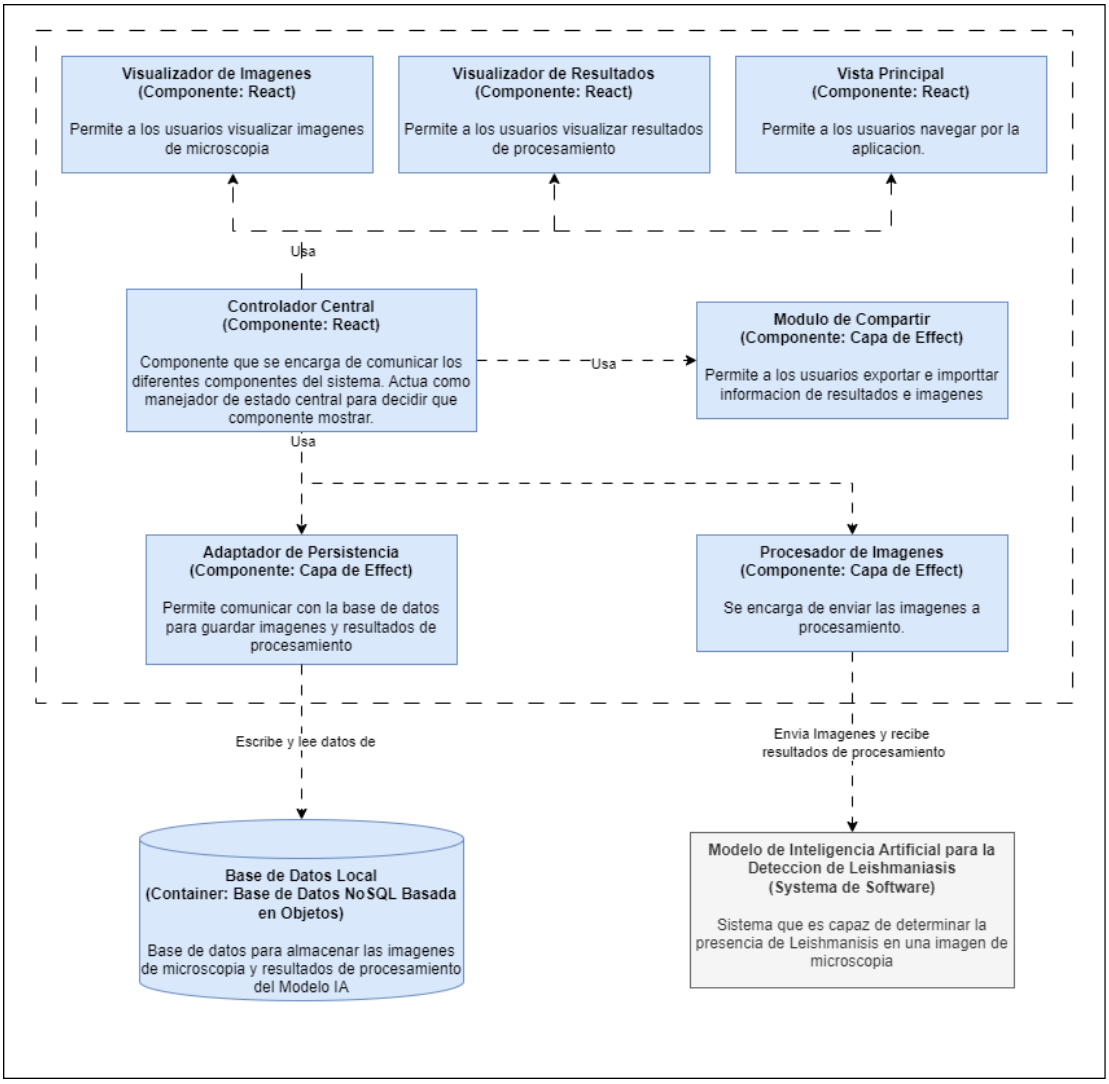


Figura 8: Diagrama de Componentes

La **Figura 8: Diagrama de Componentes** hace referencia al contenedor de AWP. La aplicación será compuesta por tres vistas, cuyo estado será controlado por el Controlador Central. Este es el componente principal y proveerá acceso a los módulos de negocio del sistema. Las vistas se implementarán como componentes de React y los módulos de negocio como capas de Effect-ts.

6. IMPLEMENTACIÓN

Micro Tracker fue implementado como prueba de concepto y para la validación de la arquitectura. Como tal, Micro Tracker es una aplicación web progresiva, elaborada en TypeScript con React, orientada a la captura, almacenamiento y procesamiento de imágenes de microscopia. Internamente contiene un modelo de inteligencia artificial [12] que detecta la presencia de leishmaniasis en muestras de microscopia.

6.1. Base de Datos

Micro Tracker para la persistencia de datos hace uso de IndexedDB. Hace uso de tres almacenes de objetos: uno para imágenes, uno para tareas y uno para resultados de procesamiento. Para el uso de IndexedDB se desarrolló una capa ligera de integración. Esta capa permite la definición de la base de datos, pero no provee funcionalidades para migración. Se usó una combinación del patrón Repositorio y de esquemas de Effect para elaborar capas para crear, leer, actualizar y borrar objetos de los almacenes. Los esquemas permiten definir la estructura de los objetos de cada almacén y además proveen funcionalidad de validación, serialización y deserialización. Usando esto, se elaboraron tres repositorios: “images”, “jobs” y “model result”. Cada almacén usa identificadores secuenciales automáticos y debido a que son almacenes de objetos, es posible describirlos usando un diagrama de clases. Cabe aclarar que, debido al uso de patrones de programación funcional, algunos tipos de datos no son representables de manera estándar en un diagrama UML. Sin embargo, el siguiente diagrama de clases lo representa parcialmente, usando los tipos primitivos de Typescript:

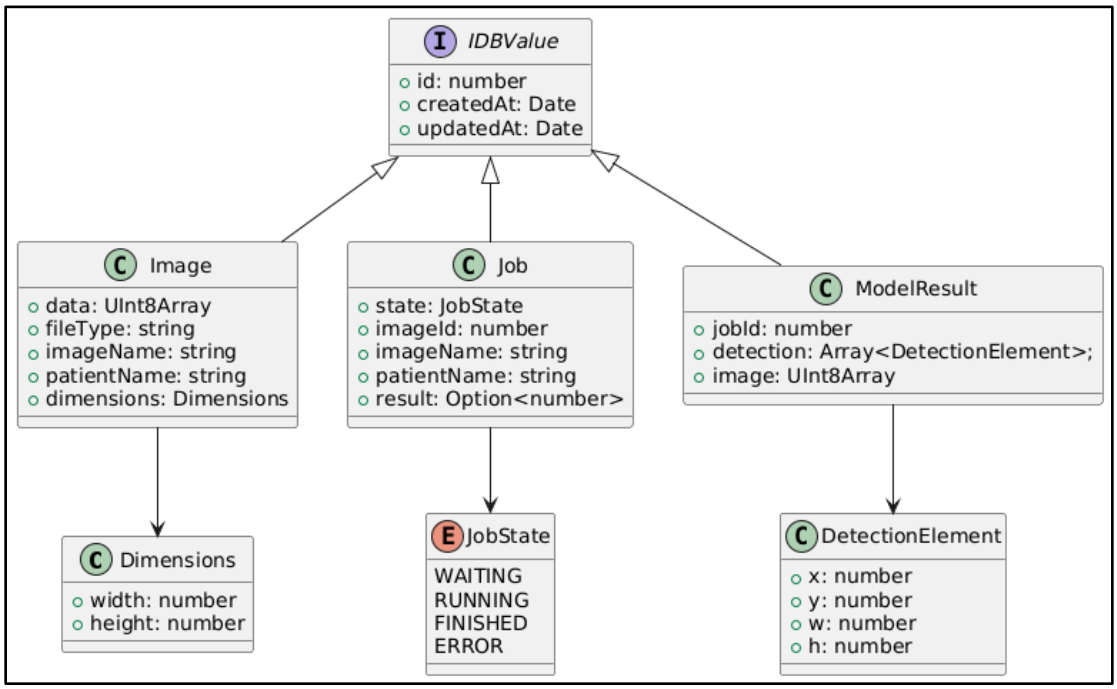


Figura 9: Diagrama de Clases De Objetos en IndexedDB

6.2. Integración de Modelo IA

El modelo de IA desarrollado en el proyecto de Leishmaniapp, hace uso de un formato específico para la transmisión de datos. Debido a que este proyecto pretende ejecutar de manera local el procesamiento de imágenes, se extrajo el componente que se encarga de la detección de macrófagos, sin la adaptación al formato ALEF (Adapter Layer Exec Format). Este consta de un archivo en Python y un archivo binario con el objeto de Python del modelo de IA. El archivo de Python se encarga de preparar la imagen y de cargar el modelo del archivo binario. El modelo en formato binario es una maquina de vectores de soporte (SVM). Esto se logra ejecutar en el navegador a través de Pyodide.

WebAssembly tiene un sistema de archivos simulado que permite a código ejecutado en Pyodide usar los paquetes estándar para lectura y escritura de archivos en Python, de manera transparente. Antes de ejecutar el modelo, se interpola el nombre del archivo de entrada y del archivo de salida en el código fuente del script (usando variables globales identificadas para interpolación por medio de un doble guion bajo como sufijo y prefijo, por ejemplo "__INPUT_PATH__"). Las tres variables inyectadas a través de este mecanismo son "__INPUT_PATH__" para la imagen que se va a procesar, "__PRETRAINED_PATH__" para la dirección del archivo binario que contiene la maquina de vectores de soporte, y por último, "__OUTPUT_PATH__" que determina la dirección del archivo con el resultado de procesamiento.

A través del sistema de archivos de WebAssembly, se escribe la información de la imagen para que esté disponible desde Python. El script de Python al finalizar escribe la imagen resultante en el camino inyectado previamente. Después, desde JavaScript se lee el archivo y se guardan los bytes a IndexedDB a través del repositorio de ModelResults. Adicionalmente, el modelo provee la información de detección a través de la salida estándar. Esta información se incluye en el resultado de procesamiento para uso posterior, en caso de que se llegase a necesitar.

Para disminuir el peso en disco de Pyodide, y por consecuencia de Micro-Tracker, se hizo un análisis manual de las dependencias usadas por el modelo IA y se borraron todas las dependencias incluidas en Pyodide que no eran necesarias. Este ambiente de ejecución (o runtime) reducido de Pyodide fue nombrado pyodide-mini. El paquete completo de Pyodide ocupa 1.18 GB y contiene 592 archivos. Usando pyodide-mini, esto se redujo a 367 MB y 116 archivos, evitando el almacenamiento en memoria caché de paquetes de Python innecesarios. Este análisis fue posible debido a que Pyodide ofrece una funcionalidad que lista que dependencias que son usadas por el código fuente a ejecutar.

Esta es una adaptación ad hoc del modelo de IA basada en interpolación de código fuente, aunque funcional, no se recomienda por dos razones. La primera siendo que es altamente dependiente de la implementación concreta del modelo, debido a que el código fue analizado y modificado para poder inyectar los valores. Esto hace al modelo divergir de la implementación que lleva el proyecto Leishmaniapp. Se hizo de esta manera debido a que al inyectar valores por medio del mecanismo que provee Pyodide, se perdían las referencias a las dependencias necesarias para ejecutar el modelo. Se recomienda regresar y evaluar como se logra la inyección de los valores de entrada y salida. Esto se considera esencial para la evolución de

este sistema debido a que, sin este cambio, no se puede generalizar la comunicación con los modelos de IA.

La segunda razón por la que no se recomienda usar esta metodología de integración es que el proceso de construcción de el runtime reducido de pyodide, fue un proceso manual. Si se considera extender esta implementación para proveer soporte a otros modelos de IA, seria recomendable llevar a cabo ese proceso de manera automática, haciendo un componente que se encargue de actuar como un registro de modelos IA y sus dependencias. Esto con el propósito de borrar las dependencias a las que no se hace uso en ningún modelo y generar el runtime reducido al momento de compilar. Pyodide ya provee el mecanismo para listar las dependencias en uso, pero carece de una herramienta para reducir el ambiente de ejecución.

Un detalle de implementación para destacar de esta integración es que por medio del uso extenso de la librería de effect-ts, es completamente posible integrar con modelos remotos de IA sin necesidad de que todo el sistema sea consiente de este hecho. Actualmente, él envió de una petición de procesamiento se describe como un efecto, el cual puede ser ejecutado de manera asincrónica o sincrónica. Generalmente se ejecutan de manera asíncrona y por esto, si el modelo fuera remoto, ese hecho tendría una superficie de impacto reducida a solo cambios en la capa de integración hacia los workers. Los componentes de React que actúan como consumidores de las capas no notaria cambio. Se recomienda mantener este detalle en cuenta.

6.3. Funcionamiento General de la Aplicación

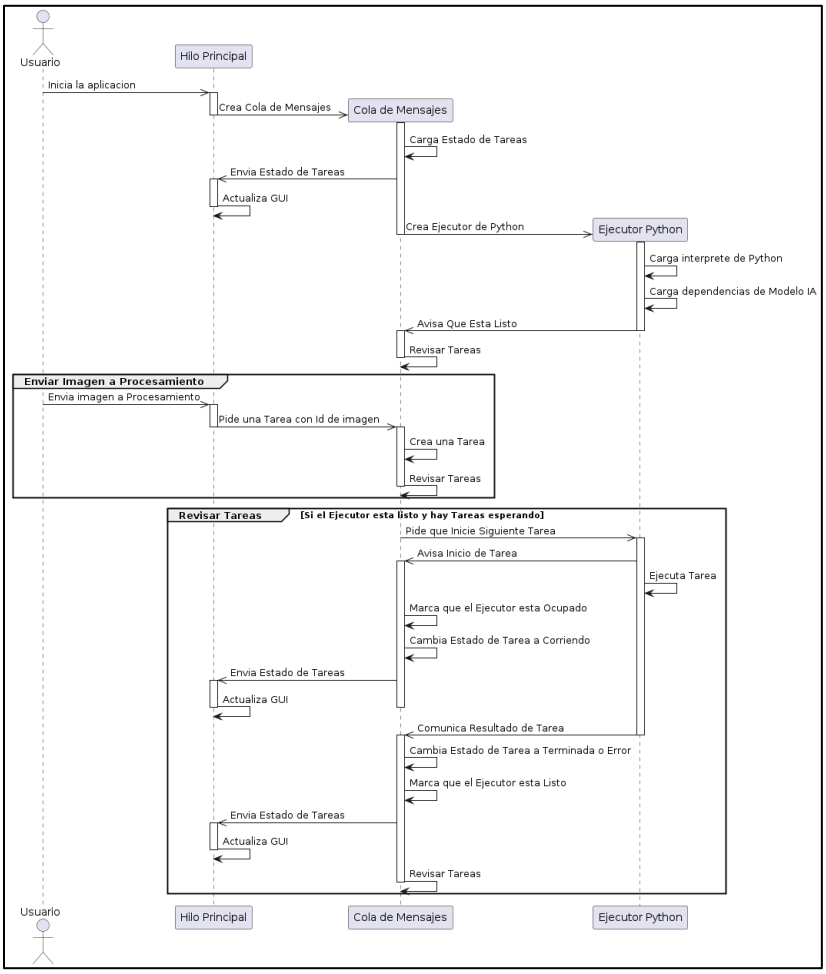


Figura 10: Funcionamiento General de Micro Tracker

El funcionamiento de Micro Tracker se basa en tres hilos de ejecución: El hilo principal se encarga de la interfaz gráfica, un hilo dedicado a una cola de mensajes que guarda el estado de las tareas de procesamiento y un hilo dedicado a la ejecución del modelo de inteligencia artificial. Los hilos son implementados usando Workers de Javascript. Aunque, se intentó tener múltiples hilos para ejecución del intérprete de Python, debido a límites de memoria se restringe a un solo hilo dedicado a ejecución de código en Python. Este requiere alrededor de 800 MB de memoria al iniciar debido a que carga múltiples paquetes necesarios para la ejecución del modelo de IA. Después de la carga inicial, su uso de memoria disminuye y se mantiene en 650 MB, hasta que se haga una petición de procesamiento. En ese caso, su uso de memoria aumenta, según el tamaño de la imagen enviada a procesamiento.

El hilo principal mantiene una copia ligera del estado actual de tareas de la cola de mensajes. Esta es una vista reducida de la estructura interna que la cola de mensajes usa para almacenar

el estado de las tareas. La cola de mensajes se implementa como una cola de prioridad de tareas que se comunica basada en eventos y mensajes de workers. La cola maneja un mecanismo "push" de envío de mensajes: es la cola quien le notifica al hilo de Python de tareas disponibles. Existen cuatro momentos en los que la cola de mensajes revisa las tareas por ejecutar: al momento que una tarea es puesta en cola por el hilo principal, al momento que el ejecutor termina de inicializar, al momento que el ejecutor termina una tarea y al momento que el ejecutor comunique que una tarea falló.

Las tareas (internamente llamadas "jobs") representan el procesamiento de una imagen. Estas pueden ser descritas como una máquina de estado finita, con 4 posibles estados: esperando ("Waiting"), en ejecución ("Running"), terminada ("Finished") o en estado de error ("Error"). Se crean usando el id de una imagen guardada en la aplicación. Inician en estado de espera hasta que el hilo ejecutor reciba la tarea y le comunique a la cola de mensajes que está ejecutando esa tarea. Existen dos maneras que una tarea puede pasar a un estado de error. La primera es cuando la imagen de entrada es borrada antes de que la cola de mensajes mande la tarea al ejecutor. Esto es debido a que, al mandar la tarea a ejecución, la cola de mensajes obtiene los datos de la imagen de entrada de la base de datos para enviársela al ejecutor. La segunda manera que pasa a un estado de error es si el modelo de IA falla en ejecución por cualquier motivo.

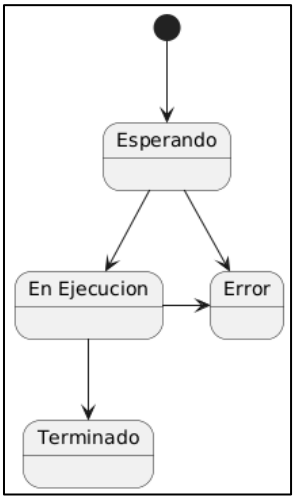
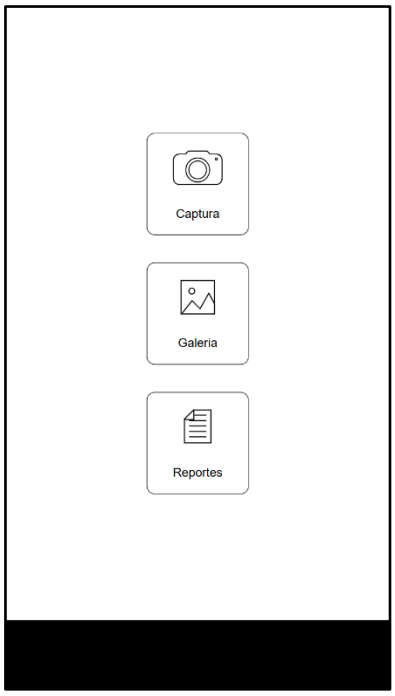


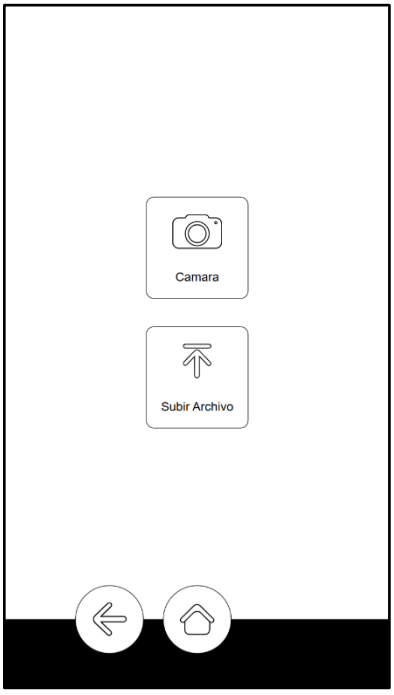
Figura 11: Diagrama de Estado de Tareas



Pantalla 1: Menú Principal

6.3.1 Menú Principal

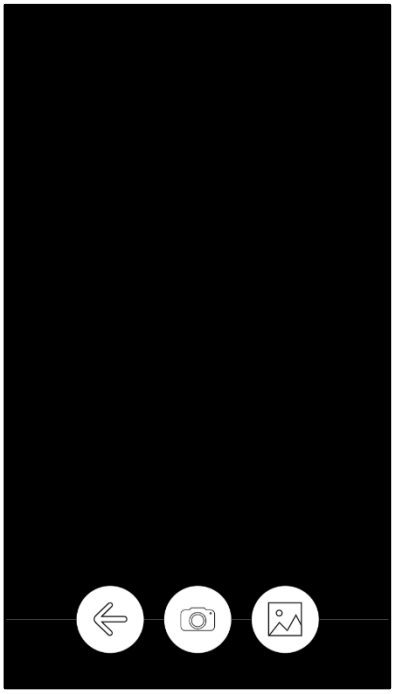
Esta es la pantalla principal de la aplicación. Desde aquí, el usuario puede acceder a las vistas y usar las diferentes funcionalidades provistas en cada una. Algunas vistas tienen un icono que se asemeja a una casa (⌂) en el botón central inferior. Cuando ese sea el caso, oprimir el botón lleva al usuario a esta vista.



Pantalla 2: Menú de Captura

6.3.2 Menú de Captura

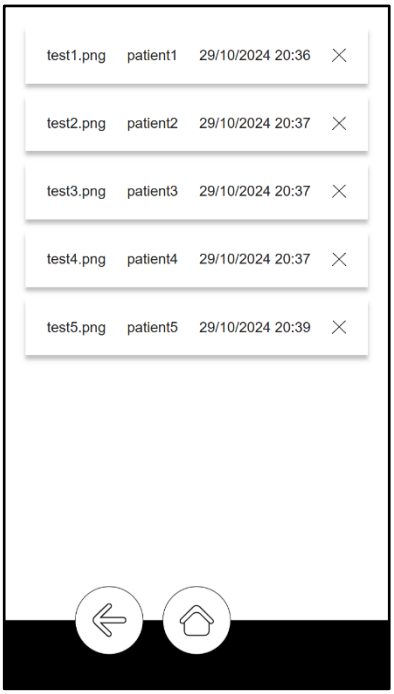
Este menú le permite al usuario escoger la manera que desea ingresar una imagen. La opción de cámara lo lleva a la vista de **6.3.3 Cámara** para tomar fotos usando las cámaras que provee el dispositivo. La opción de subir archivo abre el explorador de archivos para subir una imagen y guardarla en el sistema para su posterior procesamiento. Al seleccionar un archivo valido, lo lleva a la vista **6.3.5 Previsualización** o **6.3.6 Selección de Corte** dependiendo de si el archivo es de formato soportado nativamente por el navegador o de si es de formato Tiff, respectivamente.



Pantalla 3: Cámara

6.3.3 Cámara

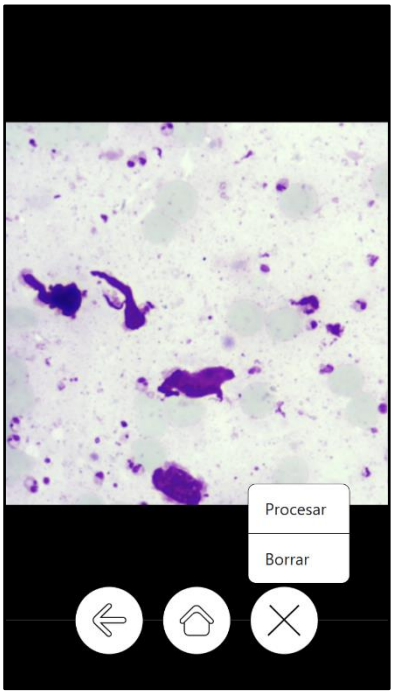
Esta vista le permite al usuario tomar fotos usando la cámara y guardar fotos tomadas. Para facilitar la navegación se provee la opción de navegar a galería. La vista garantiza mantener la razón de aspecto de las fotos que normalmente toma el dispositivo.



Pantalla 4: Galería

6.3.4 Galería

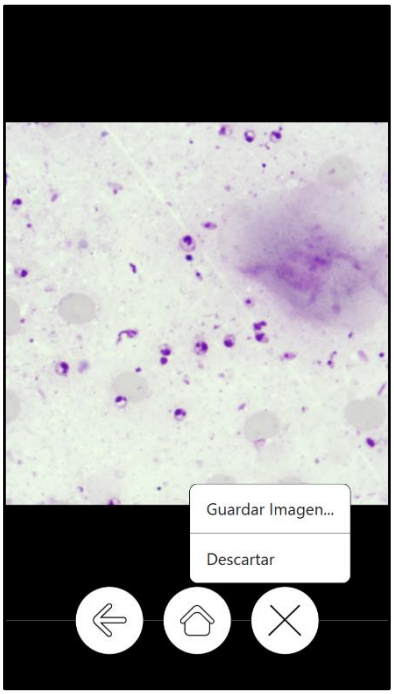
Esta vista le muestra al usuario las diferentes imágenes guardadas. Aquí se le muestra el nombre de la imagen, el nombre del paciente asociado y la fecha de la última actualización. Puede eliminarlas usando la X a la derecha de la fecha de modificación. Al pulsar en cualquier parte del renglón, la aplicación navega a la vista **6.3.5 Visualizador** con la imagen asociada.



Pantalla 5: Visualizador

6.3.5 Visualizador

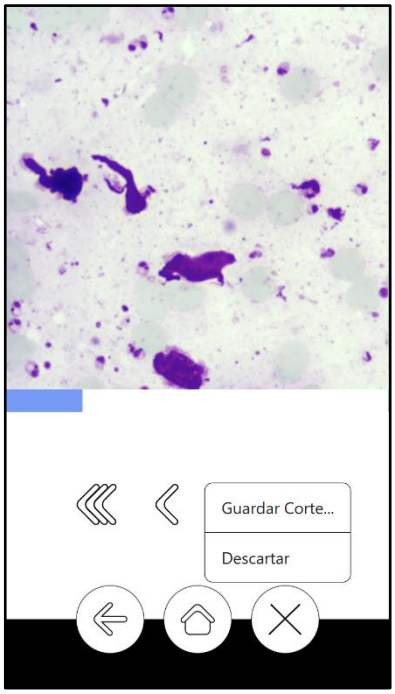
Esta vista le permite al usuario visualizar una imagen guardada y es la única manera de enviar una imagen a procesamiento. También, por conveniencia, permite eliminar la imagen.



Pantalla 6: Previsualización

6.3.6 Previsualización

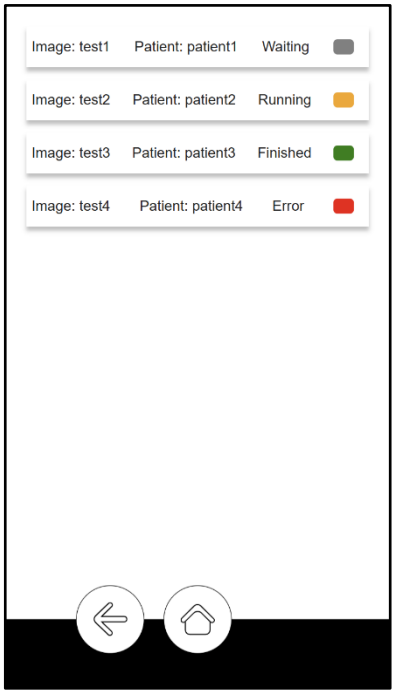
Esta pantalla se muestra cuando un usuario selecciona subir un archivo de un formato de imagen soportado por el navegador (cualquiera de formato png, apng, avif, gif, svg, jpeg, jpg o webp). Desde aquí puede guardar o descartar la imagen usando el menú de opciones. Al seleccionar guardar abre un modal para ingresar un nombre para la imagen y nombre del paciente asociado. Los nombres de las imágenes no son únicos y pueden ser repetidos. Al guardar exitosamente la imagen, navega a **6.3.5 Visualizador** con la imagen guardada automáticamente. Sin importar el formato original de la imagen, esta es convertida en png para ser guardada.



Pantalla 7: Selección de Corte

6.3.7 Selección de Corte

Esta vista se muestra cuando un usuario decide subir un archivo de formato Tiff. Esta le permite mirar los diferentes cortes de la imagen. Sin embargo, solo permite guardar un único corte, que se guarda como una imagen png. Al igual que la vista **6.3.6 Previsualización**, esta le permite descartar la imagen usando el menú y la opción de "Descartar" y un modal aparece cuando selecciona "Guardar Corte…" para ingresar un nombre para el corte y nombre del paciente asociado. Al igual que con otros formatos de imágenes, el corte es transformado a formato png, los nombres de los cortes guardados no son únicos y pueden ser repetidos. Al guardar exitosamente la imagen, navega a **6.3.5 Visualizador** con la imagen guardada automáticamente.



Pantalla 8: Lista de Tareas

6.3.8 Lista de Tareas

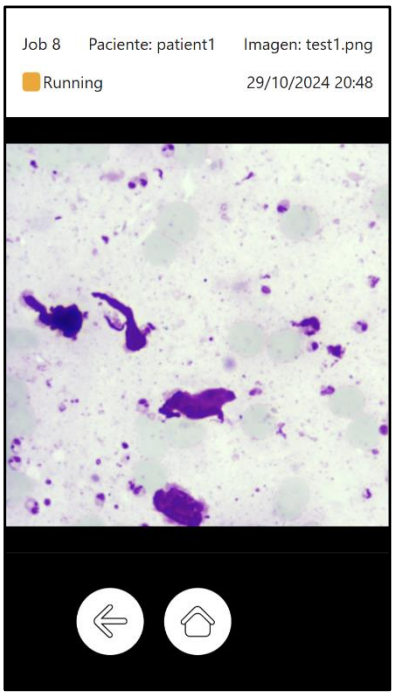
Esta vista le permite al usuario ver que tareas hay en la cola y el estado actual de cada tarea. Por tarea muestra el nombre de la imagen de entrada, el nombre del paciente asociado y el estado actual en inglés, con un color para fácil reconocimiento. Oprimir cualquiera navega a la vista **6.3.9 Visualizador de Tarea** para ver los detalles de la tarea.

6.3.9 Visualizador de Tarea

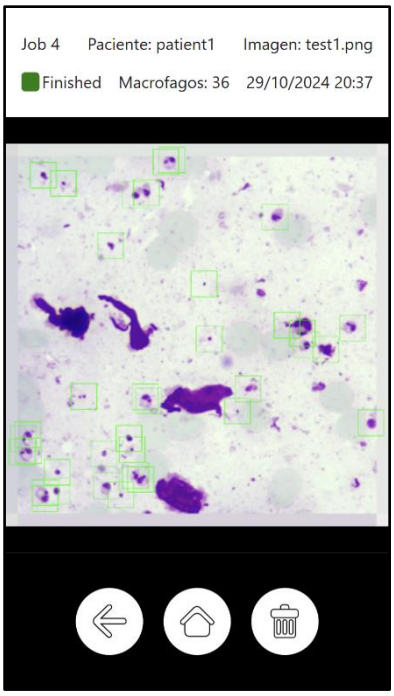
Esta vista le permite al usuario mirar los detalles de una tarea de procesamiento. La información provista es el identificador interno de la tarea (mostrado con el formato “Job [id]”), el nombre de paciente asociado con la imagen de entrada, el nombre de la imagen de entrada, el estado actual de la tarea y la fecha de última actualización. El número de macrófagos y la imagen resultante del procesamiento solo son visibles cuando la imagen se encuentra en el estado de terminada. En caso contrario, se intenta mostrar la imagen de entrada, si esta se encuentra disponible. Si no se encuentra disponible, el espacio queda en blanco.

La única manera de borrar información de tareas es a través de esta vista. La opción de borrar una tarea no se encuentra disponible en tareas en estado de ejecución. Eliminación es permitido para cualquier otro estado.

Esta vista se actualiza automáticamente y no requiere de acciones por el usuario para actualizar la información de la tarea.



Pantalla 9: Tarea en ejecución



Pantalla 10: Tarea Exitosa

7. VALIDACION

Para garantizar el correcto funcionamiento de la aplicación, se hizo un uso extenso de pruebas unitarias. Particularmente a nivel de componentes, haciendo uso del sistema de inyección de capas para simular casos de uso reales en un ambiente de pruebas. Para comprobar que la aplicación funciona sin conectividad y en dispositivos con recursos limitados, se usa Playwright, una librería para elaborar pruebas automatizadas que permiten emulación del navegador y de dispositivo. La emulación de Playwright es a nivel de funcionamiento de navegador, no emula hardware. Las pruebas de Playwright cubren los flujos principales, es decir los casos de uso CU5 y CU6. La ejecución de las pruebas se llevó a cabo en una laptop con 32 GB de RAM y un procesador Intel i7-1355U, corriendo con Windows 11 23H2 de 64-bit. La imagen de prueba usada es una imagen en formato PNG, con dimensiones de 1944x1944, que pesa 4,12 MB y contiene 36 elementos de interés (macrófagos).

Por medio de la emulación provista por Playwright, se verificó que Micro Tracker es capaz de funcionar sin conexión a internet, en navegadores basados en chromium, en navegadores basados en firefox, en un iPhone 10 y en un Pixel 3, siempre y cuando tenga conectividad hasta terminar la carga del service worker.

Se realizó una prueba end-to-end para validar el flujo principal de la aplicación, sin conexión, ejecutando los siguientes pasos en las diferentes plataformas:

1- Abrir el navegador en la página raíz de la aplicación
2- Esperar que el service worker termine su registro
3- Activar la emulación de falta de conectividad
4- Hacer clic en el botón de “Captura”
5- Hacer clic en el botón de “Subir Archivo”
6- Seleccionar el archivo a subir
7- Esperar que el sistema renderice el archivo
8- Hacer clic en el menú de opciones
9- Hacer clic en el botón de “Guardar Imagen…”
10- Llenar nombre de archivo y nombre de paciente
11- Hacer clic en el botón de “Guardar”
12- Hacer clic en el menú de opciones
13- Hacer clic en la opción de “Procesar”
14- Hacer clic en el botón de “Ok” para esconder el modal de confirmación
15- Hacer clic en el botón de navegación al menú principal
16- Hacer clic en el botón de “Reportes”
17- Hacer clic en el trabajo de procesamiento en estado de “Running” para pasar a la vista de trabajo individual
18- Esperar que el trabajo pase a estado “Finished”
19- Verificar que aparezca la palabra “Macrofagos” en pantalla para garantizar que se muestra el resultado de procesamiento de manera correcta.

En los resultados de la ejecución de la prueba anterior se observa que Firefox es el navegador con el mayor tiempo de ejecución, demorándose alrededor de 2 minutos. Por otro lado, la

ejecución en chromium, en el iPhone 10 y en el Pixel 3 presentan tiempos similares, alrededor de 24 segundos. El reporte de la ejecución de la prueba se puede ver en el Anexo 4: Resultado de Pruebas de Conectividad.

Adicionalmente, se realizó una prueba de carga en la que se verifica que el sistema puede guardar 100 imágenes de manera exitosa. Para esta prueba se ejecutan los siguientes pasos:

1- Abrir el navegador en la página raíz de la aplicación
2- Hacer clic en el botón de “Captura”
3- Hacer clic en el botón de “Subir Archivo”
4- Seleccionar el archivo a subir
5- Esperar que el sistema renderice el archivo
6- Hacer clic en el menú de opciones
7- Hacer clic en el botón de “Guardar Imagen…”
8- Llenar nombre de archivo y nombre de paciente
9- Hacer clic en el botón de “Guardar”
10- Hacer clic en el botón para navegar hacia atrás
11- Repetir los pasos 1 a 10, 100 veces.
12- Hacer clic en el botón para navegar al menú principal
13- Hacer clic en el botón de “Galeria”
14- Esperar a que se muestren las imágenes en pantalla
15- Verificar que se muestren 100 imágenes en la vista de galería para garantizar que se guardaron exitosamente.

En los resultados de la prueba anterior se muestra que Firefox falla por límites de memoria. En cuanto a tiempos de ejecución, Chromium, el Pixel 3 y el iPhone 10 tienen tiempos de 2 minutos. Firefox toma 3 minutos en cargar todas las imágenes, pero falla al momento de hacer lectura de IndexedDB. El Pixel 3 falla cuando se ejecuta en conjunto con la prueba de Firefox entonces se hace en aislamiento la prueba. Los detalles de ejecución de esta prueba se pueden ver en el Anexo 5: Resultado de Prueba de Carga y Anexo 6: Resultado de Prueba de Carga Pixel 3

8. CONCLUSIONES

El diagnostico de leishmaniasis por medio de muestras de microscopia es un proceso tedioso, propenso al error humano. En este proceso existe la posibilidad de apoyar al especialista por medio de herramientas tecnológicas. Una herramienta que busca esto mismo es Leishmaniapp, que a través de tecnologías de inteligencia artificial y una aplicación Android nativa, apoya al médico en la ejecución del diagnóstico haciendo detección automática de posibles parásitos en la muestra de microscopia. Sin embargo, Leishmaniapp no implementa procesamiento local de las imágenes y requiere de conectividad a internet para lograrlo. Debido a que la leishmaniasis se presenta generalmente en ambientes rurales donde la conectividad no está garantizada, sería deseable un sistema capaz de llevar a cabo el procesamiento de manera local, preferiblemente en un dispositivo de bajo rendimiento.

En el proyecto se hizo un análisis de las herramientas que podían ser utilizadas para el apoyo al diagnóstico de la leishmaniasis. Posteriormente se basó en el proyecto de Leishmaniapp para así proponer una arquitectura y construir una prueba de concepto que sirviese para la validación de hacer el procesamiento de imágenes de manera local.

La implementación de la arquitectura ha demostrado la viabilidad de las paginas web como alternativa a aplicaciones stand alone: la alta disponibilidades de funcionalidades avanzadas dentro de los navegadores permitió la elaboración de este prototipo. Es importante destacar el protagonismo que tienen los Workers, WebAssembly e IndexedDB que, si los navegadores no tuvieran amplio soporte de estas funcionalidades, este prototipo no seria posible. Sin embargo, es importante notar la alta dependencia con el navegador. Incluso es importante mencionar que, aunque la mayoría de los navegadores soportan PWAs, no todos proveen la capacidad de instalar PWAs. Este limitante, aunque es un problema para la proliferación de PWAs, para este proyecto no se considera un problema debido que el usuario objetivo tiene la posibilidad de usar un sistema basado en Android, el cual incluye un navegador basado en Chromium por defecto, y este soporta instalación de PWAs.

Otro limitante a tomar en cuenta es la capacidad de almacenamiento de IndexedDB. Si se requiere más de 500 MB de imágenes almacenadas, se recomienda usar Google Chrome para escritorios y Safari para dispositivos móviles. Mientras no haya cambios en este aspecto, se recomienda cambiar IndexedDB por otra solución de almacenamiento.

La integración con el modelo de IA, aunque funcional, no es adecuada en términos de mantenimiento y evolución: la interpolación de código fuente como mecanismo de comunicación e inyección de valores no es generalizable. Seria importante evaluar las alternativas que provee Pyodide para lograr esta tarea. Se sugiere elaborar un formato de comunicación común que permita abstraer la inyección de valores de entrada y salida para generalizar la integración con otros modelos.

A pesar de que los objetivos de este proyecto son llevar el procesamiento de la nube a local, si se considera el acercamiento tomado por Micro Tracker para un proyecto futuro, se recomienda evaluar la posibilidad de permitir al usuario hacer tanto procesamiento local como

remoto. La implementación no impone limitantes para la extensión del proyecto para permitir procesamiento remoto y asume que un paso futuro es permitirlo.

Por cuestiones de tiempo y logística no fue posible evaluar la usabilidad y relevancia en manos de un usuario objetivo de este proyecto. Seria importante en un futuro recibir retroalimentación de usuarios objetivo para tomar en cuenta e incluir dentro del proceso. Esto es aún mas relevante si se planea llevar este proyecto a un ambiente real.

Un aspecto que se promueve y se considera importante es el contexto donde la leishmaniasis se presenta. Es por eso que el procesamiento local, la independencia de la conectividad y la necesidad de permitir el procesamiento en dispositivos móviles es de vital importancia para este proyecto. Si trabajos futuros consideran basarse en Micro Tracker, se sugiere mantener estos pilares principales del proyecto: independencia a conectividad, recursos de maquina disminuidos.

Micro Tracker se considera un éxito en llevar el procesamiento, previamente en la nube, a procesamiento local. Tiene múltiples caminos de mejora que serian interesantes de evaluar: mecanismos de persistencia alternos, mejora de manejo de paralelismo, procesamiento remoto o local.

BIBLIOGRAFIA

[1] D. Steverding, "The history of leishmaniasis," *Parasit Vectors*, vol. 10, no. 1, p. 82, Dec. 2017, doi: 10.1186/s13071-017-2028-5.

[2] K. Khan, S. Wahid, N. H. Khan, and N. Ali, "Potential Resting and Breeding Sites of Sand Flies (Diptera: Psychodidae) and Their Habitat Characteristics in Leishmaniasis Foci of Dir Districts, Khyber Pakhtunkhwa, Pakistan," *J Med Entomol*, vol. 54, no. 5, pp. 1390–1396, Sep. 2017, doi: 10.1093/jme/tjx098.

[3] S. Thakur, J. Joshi, and S. Kaur, "Leishmaniasis diagnosis: an update on the use of parasitological, immunological and molecular methods," *Journal of Parasitic Diseases*, vol. 44, no. 2, pp. 253–272, Jun. 2020, doi: 10.1007/s12639-020-01212-w.

[4] M. Górriz, A. Aparicio, B. Raventós, V. Vilaplana, E. Sayrol, and D. López-Codina, "Leishmaniasis Parasite Segmentation and Classification Using Deep Learning," 2018, pp. 53–62. doi: 10.1007/978-3-319-94544-6_6.

[5] C. Gonçalves *et al.*, "Computer Vision in Automatic Visceral Leishmaniasis Diagnosis: a Survey," *IEEE Latin America Transactions*, vol. 21, no. 2, pp. 310–319, Feb. 2023, doi: 10.1109/TLA.2023.10015224.

[6] A. Isaza-Jaimes *et al.*, "A computational approach for Leishmania genus protozoa detection in bone marrow samples from patients with visceral Leishmaniasis," 2021, doi: 10.5281/ZENODO.4426403.

[7] C. Dallet, S. Kareem, and I. Kale, "Real time blood image processing application for malaria diagnosis using mobile phones," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, Jun. 2014, pp. 2405–2408. doi: 10.1109/ISCAS.2014.6865657.

[8] K. Schwaber and J. Sutherland, "The 2020 Scrum Guide™," https://scrumguides.org/scrum-guide.html#purpose-of-the-scrum-guide. [Online]. Available: https://scrumguides.org/scrum-guide.html#purpose-of-the-scrum-guide

[9] P. J. Hotez and A. Kamath, "Neglected Tropical Diseases in Sub-Saharan Africa: Review of Their Prevalence, Distribution, and Disease Burden," *PLoS Negl Trop Dis*, vol. 3, no. 8, p. e412, Aug. 2009, doi: 10.1371/journal.pntd.0000412.

[10] C. Gorman *et al.*, "Interoperable slide microscopy viewer and annotation tool for imaging data science and computational pathology," *Nat Commun*, vol. 14, no. 1, p. 1572, Mar. 2023, doi: 10.1038/s41467-023-37224-2.

[11] C. Shim, W. Kim, T. T. D. Nguyen, D. Y. Kim, Y. S. Choi, and Y. D. Chung, "CellTrackVis: interactive browser-based visualization for analyzing cell trajectories

and lineages," *BMC Bioinformatics*, vol. 24, no. 1, p. 124, Mar. 2023, doi: 10.1186/s12859-023-05218-y.

[12] N. Pérez Fonseca, N. D. Jaime Castañeda, J. D. Romero Torres, and A. D. Talero Peñuela, "Implementación de arquitectura para aplicación móvil como herramienta de detección de Leishmaniasis cutánea y otras enfermedades parasitarias," Dec. 07, 2023, *Pontificia Universidad Javeriana*. [Online]. Available: http://hdl.handle.net/10554/67405

ANEXOS

Anexo 1: Estado del arte

Anexo 2: Análisis de Arquitecturas

Anexo 3: Arquitectura

Anexo 4: Resultado de Pruebas de Conectividad

Anexo 5: Resultado de Prueba de Carga

Anexo 6: Resultado de Prueba de Carga Pixel 3