

UNIVERSIDAD TECNICA FEDERICO SANTA MARIA

DEPARTAMENTO DE ELECTRÓNICA

VALPARAÍSO - CHILE

“DESARROLLO DE APLICACIÓN WEB DE UN SISTEMA DE MONITOREO DE SIGNOS VITALES”

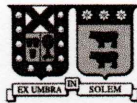
MEMORIA DE TITULACIÓN PARA OPTAR AL TITULO
INGENIERO CIVIL ELECTRÓNICO MENCIÓN COMPUTADORES

DIEGO ANDRÉS VÁSQUEZ CANCINO

Profesor Supervisor
Dr. AGUSTÍN GONZÁLEZ VALENZUELA

Profesor Co-Referente
Dr. RUDY MALONNEK WETZEL

Enero de 2025



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Desarrollo de aplicación WEB para monitoreo de signos vitales

Nombre del candidato(a): Diego Vásquez Cancino

Carrera / Grado: Ingeniería Civil Electrónica

Campus: Casa Central

Departamento: Electrónica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Agustín J. González Valenzuela, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por (**marcar una opción**):

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 29.10.2025

Firma:

Estudiante o Candidato(a):

Fecha: 29.10.2025

Firma:

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

*Es el tipo de cosas que te hace agradecer que paraste a oler los pinos en el
camino, ¿sabes?*
-Gabbro, *Outer Wilds* (Mobius Digital, 2019)
Traducción: Diego Vásquez

*Dedico este trabajo a mi familia, amigos, y todos los que quedaron en el
camino.*

AGRADECIMIENTOS

QUISIERA expresar mi gratitud por todos estos años en la universidad a mis compañeros, por su presencia en los tiempos de alegría, incertidumbre, y dificultad académica. A mis amigos y familia que han estado apoyándome y alentándome a completar este proceso. Y también agradecer a mi profesor, Agustín González, por la oportunidad de ser parte de su equipo para el proyecto, y por su intachable diligencia con su trabajo y sus estudiantes.

-Diego Andrés Vásquez Cancino

RESUMEN

LOS accidentes son hechos comunes en todo el mundo, una variedad especialmente peligrosa son los accidentes de tránsito; causan el fallecimiento de muchas personas cada año. Sólo en Chile se registraron más de 86000 accidentes vehiculares, con 1745 muertes asociadas a ellos en 2022 [1]. El personal de urgencias que acuda a la escena demorará varios minutos y hasta horas en el proceso de llegar, ejecutar la atención médica necesaria para estabilizar a las víctimas, y llevarla a un centro hospitalario para salvar sus vidas. El concepto de 'Hora dorada' existe en la medicina, el enfoque del concepto se puede acotar a: **Entre antes se le preste atención médica a las víctimas de un accidente, mayor será su probabilidad de supervivencia** [7]. Para ayudar a agilizar la labor de los equipos de urgencia, en 2012 se propone y crea un sistema de comunicación e intercambio de datos para enriquecer el reporte de la situación al hospital destino y permitir que un médico del mismo pueda analizar los datos recibidos, para preparar medidas con antelación a la llegada de los pacientes en tránsito. Éste consiste de tres módulos: Una aplicación para dispositivos móviles: 'ERPHA Mobile', una aplicación en un servidor: 'ERPHA Server', y una aplicación para navegadores: 'ERPHA Monitoring', que permite visualizar y navegar entre los datos generados y almacenados por los dos módulos anteriores. El presente proyecto de Memoria se enfoca en actualizar, modernizar y mejorar el módulo 'ERPHA Monitoring', ya que la aplicación originalmente fue desarrollada en una plataforma ya obsoleta y sin soporte desde 2021: Microsoft Silverlight. El objetivo del trabajo será aplicar las funcionalidades de la versión anterior en un lenguaje de programación y formato más actualizados, específicamente JavaScript, además de considerar la posibilidad de implementar más características adicionales posibles que beneficien las aplicaciones del sistema. El trabajo logra los objetivos aquí propuestos a excepción del módulo de seguimiento por GPS, implementando varios módulos con total eficacia y alta eficiencia, habiendo desarrollado un software que muestra un uso de CPU menor a 10 % y menos de 400MB de uso de memoria RAM.

Palabras Claves

Telemedicina, Telemonitoreo, Signos vitales, Sensores médicos inalámbricos

ABSTRACT

ACCIDENTS are common occurrences around the whole world, an especially dangerous variety being traffic accidents; causing many fatalities every year. Only in Chile, over 86000 vehicular accidents, with 1745 associated deaths were registered in 2022. First responders for the incident may take minutes or even hours in the process of arriving at the scene, performing necessary medical service to stabilize victims, and then transport them to a hospital to save their lives. The 'golden hour' is a concept in medicine, the focus of the concept can be narrowed down to: **The sooner the victims of an accident receive medical attention, the higher their survival rate is.** To help agilize the labor of emergency teams, in 2012 a communications and data exchange system was proposed and created to enrich the situation report to the destination hospital and allow a resident medic to analyze the related data, to prepare measures for the arrival of the patients in transit. The system consists of three modules: An application for mobile devices: 'ERPHA Mobile', an application in a server: 'ERPHA Server', and a browser application: 'ERPHA Monitoring', which allows for visualization and navigation between data generated and stored by the previous modules. The present graduation project focuses on updating, modernizing, and improving the 'ERPHA Monitoring' module, since it was originally developed in an outdated platform without active support since 2021: Microsoft Silverlight. The objective of the work will be to apply the functionalities of the previous version in a more recent programming language and format, specifically JavaScript, as well as considering the possibility of implementing additional possible features that benefit the applications of the system. The work done achieves the proposed objectives with the exception of the GPS tracking module, implementing several modules with complete efficacy and high efficiency, developing a software which exhibits less than 10 % CPU usage and less than 400MB of RAM usage.

Keywords

Telehealth, telemonitoring, vital signs, wireless medical sensors.

ÍNDICE

AGRADECIMIENTOS	I
RESUMEN	II
ABSTRACT	III
ÍNDICE DE FIGURAS	VI
ABREVIACIONES	VII
GLOSARIO	VIII
1. INTRODUCCIÓN	1
1.1. Contexto	1
1.2. Descripción del proyecto	1
1.3. Innovación	2
1.4. Alcances y Contribuciones	2
1.5. Estructura del documento	2
2. REQUERIMIENTOS Y ARQUITECTURA PARA ERPHA MONITOR	3
2.1. El problema planteado	4
2.2. Lógica de manejo de datos	4
2.3. Flujo de trabajo de ERPHA Monitor	4
2.4. Interfaces asociadas	4
2.5. Requisitos no funcionales	5
2.6. Concepto del sistema ERPHA	5
2.7. Arquitectura	6
3. DISEÑO E IMPLEMENTACIÓN FUNCIONAL - SOFTWARE Y CÓDIGO	7
3.1. Conceptos para el diseño de la aplicación	7
3.1.1. Interfaz Gráfica - Concepto inicial	7
3.1.2. Interfaz Gráfica - Implementación de contenido	9
3.2. Planeación e implementación de la parte funcional	10
3.2.1. Node.js	10
3.2.2. React	10

3.2.2.1. Framework Remix	10
3.2.3. Descripción de módulos - Primera aproximación	11
3.2.4. Descripción de módulos - Revisión después de consulta con pares	11
3.2.5. Optimización, mejora e integración de módulos	13
3.3. Detalle de código	15
3.3.1. Cimientos de ERPHA Monitor	15
3.3.2. Sección de interfaz gráfica	16
3.3.3. Sistema de petición y muestra de datos	16
3.3.4. Gráficos con Chart.js	17
3.3.5. Captura de gráficos	19
3.3.6. Mapas de Google Maps	19
3.3.7. Función Loader, inicio de la ejecución	20
3.3.8. Función Index y ejecución de la aplicación	20
3.3.9. Interacción entre Javascript y HTML	22
4. PRUEBAS Y RESULTADOS	24
4.1. Pruebas aisladas de módulos	24
4.2. Pruebas de módulos integrados	25
4.2.1. Respuesta de conexión a la aplicación	25
4.2.2. Métodos de conexión	26
4.2.3. Estabilidad de conexión	26
4.2.4. Uso de recursos	26
5. CONCLUSIONES Y TRABAJO FUTURO	27
5.1. Conclusiones	27
5.2. Trabajo Futuro	28
6. ANEXO	29
6.1. Notas adicionales	29
BIBLIOGRAFÍA	30
COPYRIGHT	32

Índice de figuras

2.1. Diagrama simple del sistema ERPHA y su funcionamiento	3
2.2. Visualización práctica del sistema ERPHA	6
2.3. Gráfica de la arquitectura de Node.js (desde litslink.com) [3]	6
3.1. Bosquejo de colores del diseño de la aplicación	8
3.2. Interfaz del proyecto original, con miniatura de datos en vivo a la izquierda	9
3.3. Captura de aplicación, con vista de signos permanente agregada	12
3.4. Captura de navegador con la aplicación y pacientes simulados como ejemplo, con vista de signos vitales permanente agregada.	12
3.5. Diagrama de flujo de datos mostrados en la aplicación, en su versión de muestra	14
3.6. Captura de pantalla de función que captura instantáneas de los gráficos, habilitada en la aplicación	15
3.7. Fragmento de código HTML, comentado, que describe varias secciones de una página de la aplicación	16
3.8. Diagrama general de funcionamiento de la aplicación	17
3.9. Captura de gráficos de ECG	17
3.10. Captura de gráfico de presión vs. tiempo de recepción, con medidas excesivas graficadas para visualizar diferencias	18
3.11. Captura de gráfico de presión vs. número de unidad de tiempo, en funcionamiento	18
3.12. Captura de gráfico de oximetría, en funcionamiento	19
3.13. Figura del sistema de prueba local	21
4.1. Vista de la pestaña de Mapa en la aplicación	25
4.2. Consola de Node, luego de iniciar la aplicación	25

ABREVIACIONES

Mayúsculas

CSS: Cascading Style Sheets (*Hojas de estilo en cascada*).

Lenguaje que sirve para personalizar el aspecto visual y diseño de una página web.

ERPHA: Emergency Response Pre-Hospital Assistance (*Asistencia Pre-Hospitalaria de Respuesta de Emergencia*).

EMT: Emergency Medical Technician (Médico Técnico de emergencia).

ECG: Electrocardiograma.

ECMA/ECMAScript: European Computer Manufacturers Association (*Asociación europea de fabricantes de computadores*).

ECMAScript es una especificación de lenguaje de programación, de la cual nació Javascript.

ES6: ECMAScript 6.

JSON: Javascript Object Notation (*Notación de Objeto Javascript*).

JSX: Javascript XML.

URL: Uniform Resource Locator (*Localizador de recursos uniforme*).
También llamadas direcciones web.

XML: Extensible Markup Language (*Lenguaje de marcado extensible*).

GLOSARIO

Frontend: *Parte delantera*, término de diseño web que abarca todo lo que el usuario final ve al utilizar una página web; desde el diseño gráfico hasta las interfaces y botones.

Backend: *Parte trasera*, término de diseño web que abarca lo que el usuario no ve como los scripts, lógica de comunicación o validación, o en general cualquier manipulación y almacenamiento de datos.

Build: *Construcción*, término usado en programación que a grandes rasgos, significa que se procesa todo el código escrito a una forma que las máquinas puedan ejecutarlo.

Bundler: *Empaquetador*, herramienta que combina muchos archivos de código Javascript en uno solo, listo para producción, y que se pueda cargar en el entorno del navegador

Framework: *Marco/Estructura*, conjunto de herramientas, guías y estructuras predefinidas que se utilizan para desarrollar y organizar software de manera eficiente

Firewall: *Muro de fuego*, sistema de seguridad empleado en computación, que consiste en un equipo o software cuya función es impedir la salida o entrada de conexiones con ciertas direcciones, según su configuración.

Hook: *Gancho*, en el contexto de programación, un Hook es una característica que permite insertar código personalizado en el flujo usual de otro programa. Similar a una función pero más modular, para permitir flexibilidad sin alterar el código base donde se inserta.

Script: *Guión*, código que automatiza tareas que debe realizar un programa o aplicación, en lugar de una máquina completa

Open source: *Fuente abierta*, denota software cuyo código fuente está disponible al público, y puede ser modificado y/o redistribuido.

Oximetría: Técnica no invasiva para medir la saturación de oxígeno en la sangre periférica

INTRODUCCIÓN

1.1. Contexto

EN la línea de trabajo de los paramédicos, existe el rol de ser parte del equipo a bordo de las ambulancias que atienden emergencias, donde, dependiendo del país, van acompañados de un conductor y posiblemente técnicos de salud (EMT) o un segundo paramédico. Por ende, al acudir a un rescate de alguien en condición grave o crítica, el equipo prioriza que el paciente no empeore su situación mientras se le transporta a un hospital para su recuperación.

La situación anterior se da considerando que los paramédicos tienen formación especializada diferente a los médicos, y que aunque sean preparados, no podrían cubrir absolutamente todos los casos o diagnosticar correctamente ciertas situaciones de emergencia. Para esto se ha considerado incluir médicos en las ambulancias en algunos casos, en los cuales Chile no está incluido. Sin embargo, siempre se puede contactar a los hospitales por radio o teléfono en algún caso particular.

El detalle más importante a considerar dada la descripción anterior, es que la comunicación verbal es muy limitada para las situaciones de gran importancia en las que se desarrollan, por lo que incluir la mayor cantidad de detalles de manera clara y concisa es imperativo para aumentar la eficacia del servicio que proveen los profesionales de la salud.

1.2. Descripción del proyecto

ERPHA Monitoring [8] es un sistema que grafica oximetría + pulso, ECG y presión sanguínea. El uso del sistema ERPHA apunta a una ambulancia que va con un paciente con monitores de signos vitales inalámbricos puestos, cuyos datos se envían a través de un dispositivo móvil al servidor y mientras se almacenan, *Monitoring* los recibe y grafica. El sistema fue originalmente desarrollado con la restricción de apuntar su uso exclusivamente a productos de Microsoft como teléfonos que usan Windows Phone, que en el presente se hallan

descontinuados u obsoletos. Por ende, el trabajo del equipo actual consiste en actualizar los módulos de todo el sistema, y específicamente para lo que representa el presente documento, mejorar el despliegue de los signos vitales a través de una aplicación web.

Para esto se consideran lenguajes open source como Javascript, que se ha desarrollado como un lenguaje compatible con todos los navegadores populares en el presente, además de la presencia de los variados frameworks que permiten su uso facilitado para muchos enfoques diferentes, el más relevante para el trabajo actual siendo la creación de aplicaciones web con profundidad en características visuales.

1.3. Innovación

Desde el 2012, año en el que el trabajo del sistema ERPHA original fue concluido, se han efectuado varias actualizaciones a JavaScript, en cambio Microsoft Silverlight vio su soporte terminado en 2021. De hecho, hasta 2023, se ha liberado una actualización al año para el estándar ECMAScript, agregando características como clases asíncronas, más métodos para las clases nativas como arreglos, declaraciones de clases, nuevas sintaxis y compatibilidad con distintos navegadores conforme pasan los 7 años de diferencia.

1.4. Alcances y Contribuciones

El usuario final para el cual el trabajo se apunta principalmente, es personal de la salud: médicos, enfermeros y equipos de ambulancia. Sin embargo la estructura de funcionamiento del sistema ERPHA es relativamente simple, de baja complejidad frente al usuario, y emplea herramientas ampliamente reconocidas en el área técnica, por lo que se podrían transformar una o varias de sus partes para emplearlas en cualquier otra situación que requiera monitoreo y almacenamiento de datos, especial pero no exclusivamente en tiempo real.

1.5. Estructura del documento

Este documento se divide en 5 capítulos. Capítulo 1 contiene la introducción y el fundamento conceptual para el trabajo, como innovación relevante y su contribución. Capítulo 2 expresa los requerimientos y arquitectura para la nueva versión de ERPHA Monitor. Capítulo 3 muestra la implementación funcional, esto es software y código creados e implementados. Capítulo 4 expone pruebas hechas y resultados obtenidos para comprobar el rendimiento de la aplicación. Finalmente, el capítulo 5 lleva las conclusiones obtenidas y consideraciones para trabajo futuro que se puede implementar después de lo que se ha completado.

REQUERIMIENTOS Y ARQUITECTURA PARA ERPHA MONITOR

Una manera de ver el sistema es con un diagrama como el de la figura 2.1, donde se establece de manera simplificada la interacción entre las partes involucradas. En rojo las interacciones como pedir datos o cambios en ellos, y en azul de entrega o subida de datos.

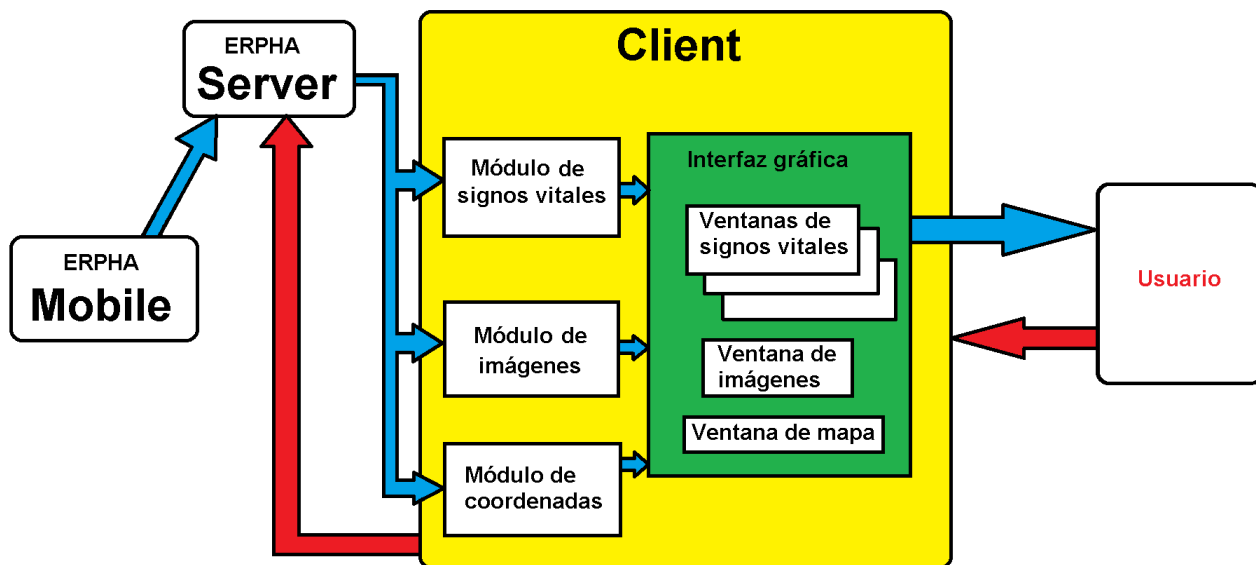


Figura 2.1: Diagrama simple del sistema ERPHA y su funcionamiento

2.1. El problema planteado

El trabajo hecho por Cristian Carrasco [8] desarrolla e integra el sistema ERPHA completo, sólo que todo fue realizado en una plataforma que se ha visto abandonada y dispositivos que ya no existen en el mercado, y por ende se ve obsoleto en su propósito como consecuencia del paso del tiempo. Entonces, se presenta el desafío en actualizar y transferir el sistema; cambiándolo de plataforma por completo para darle uso hoy en día, y tener la capacidad de mantenerlo al día fácilmente conforme la tecnología evolucione.

2.2. Lógica de manejo de datos

Como ERPHA Monitor es una aplicación de recepción y muestra en pantalla, todo lo que se debe hacer con los datos es asignarlos ordenadamente a variables para su posterior procesamiento y exposición, resultando en uso principalmente de números y texto en formato string, con la excepción de imágenes que se guardarán con un formato de compresión estándar como JPEG o PNG, transmitido mediante un enlace en formato string. Los valores numéricos se procesan para su observación en los gráficos o traducción de coordenadas en un mapa visual, el texto en general se muestra como nombres e información personal en una lista, y las imágenes se han de mostrar en un visualizador.

2.3. Flujo de trabajo de ERPHA Monitor

Se espera procesar la información entrante de las aplicaciones activas asociadas a un monitor específico (i.e. ambulancias pertenecientes a un hospital), haciéndola llegar en una forma fácilmente comprensible y rápida de leer en la pantalla de los dispositivos donde se quiera recibir, como computadores o celulares. Para ésto se sigue la secuencia de eventos a continuación planteada.

En primer lugar, se espera la comunicación con servidor ERPHA, lo cual dará paso a la recepción de datos del mismo. La aplicación permite en cualquier momento seleccionar una persona registrada como paciente en una lista para exposición de sus datos en vivo, o revisión de sus datos históricos. Si se encuentra una transmisión activa, se debe reflejar el flujo de los mismos en las gráficas y pantallas correspondientes, constantemente hasta que se pierda la conexión o se cierre el navegador en el cual la aplicación se ejecuta.

2.4. Interfaces asociadas

Para las interfaces de usuario, se emplea el concepto más simple y enfocado a tareas individuales posible: una pestaña para cada sección, tal que se abarque un aspecto con completa atención a la vez.

En cuanto a las interfaces de software, la comunicación con el servidor, según el planteamiento del sistema completo, será mediante protocolo TCP/IP, con el envío de paquetes del lado del servidor en estilo push.

Al ser una aplicación que se vale de un navegador para operar, los requerimientos e interfaces de hardware son los mismos que necesita el navegador elegido, que son

generalmente mínimos en cuanto se refiere a dispositivos como laptops o computadores integrados, que se fabrican y venden con partes predeterminadas.

2.5. Requisitos no funcionales

Seguridad, en particular la protección de los datos que se transmiten. Esto se logra mediante encriptación sincrónica, lo cual es implementado tanto en ERPHA Server como en Monitor.

Usabilidad, dado el contexto de urgencia y riesgo vital de personas en el que se usará la aplicación, es imprescindible que el diseño haga veloz el acceso y simple su utilización en todo momento, permitiendo que la aplicación se despliegue en segundos o menos una vez hecha la petición web, osea, escribir la dirección mediante la cual se accede a la aplicación y presionar ENTER en un teclado, o hacer click en un enlace hacia la misma.

La compatibilidad de ERPHA Monitor se cubre desde el comienzo del desarrollo, considerando el extendido abanico de opciones de lenguajes de programación, frameworks para los mismos, y compiladores. Esto permite no tener que investigar muy profundamente para encontrar herramientas que permitan crear un programa capaz de ejecutarse en más de un tipo de navegador, e incluso más de un dispositivo.

Mantenimiento de la plataforma, se ha definido como punto importante en el planteamiento inicial del problema. El código de la aplicación se escribe con un enfoque en la simpleza de implementación, para facilidad de desarrollo como de uso, lo que ayuda a personalizar y/o resolver posibles problemas que surjan con el paso del tiempo y uso en distintos casos. Además, las opciones consideradas de frameworks se basan en lenguajes que se mantienen a la vanguardia (e.g. JavaScript, PHP, Python, entre otros). El hecho de que la gran comunidad que los utiliza los actualiza constantemente, tanto los lenguajes base como las herramientas para manejarlos, permite un pronóstico razonable de más de dos años donde se seguirán actualizando, y donde las aplicaciones que se desarrollen con ellos puedan superar los problemas que el tiempo traiga.

2.6. Concepto del sistema ERPHA

El sistema completo es una comunicación de tres partes que intercambian varias series de datos con su contexto propio: La aplicación móvil envía los datos recopilados de los sensores hacia el servidor, y el servidor además de conservar y almacenar, envía al cliente web ERPHA Monitor en todas las iteraciones que estén funcionando.

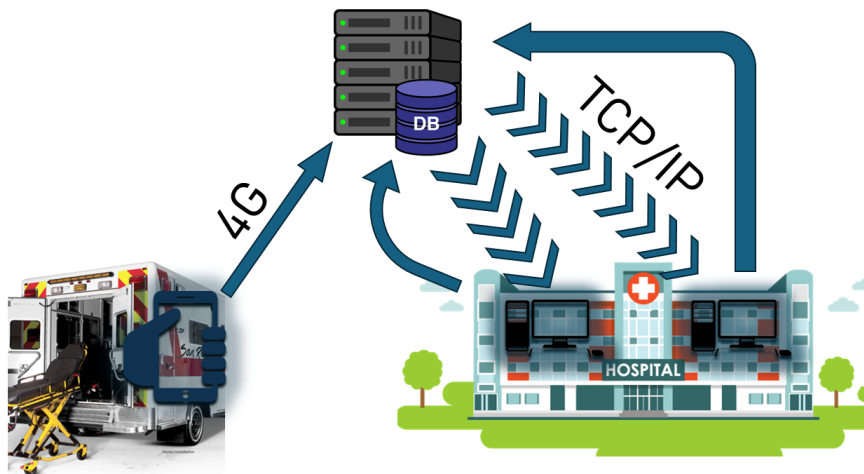


Figura 2.2: Visualización práctica del sistema ERPHA

El trabajo actual se destina a la implementación del cliente, tanto su aspecto como su funcionamiento interno, para interacción con el usuario y métodos de comunicación con las otras partes.

2.7. Arquitectura

Al construir la aplicación sobre el entorno node, se utiliza su propia arquitectura de software, basada en javascript. La aplicación utiliza el motor JavaScript V8, e internamente se genera una cola de eventos, la cual se ejecuta en un bucle, en las hebras “worker” implementadas a través de la librería Libuv. A continuación una gráfica de esta estructura.

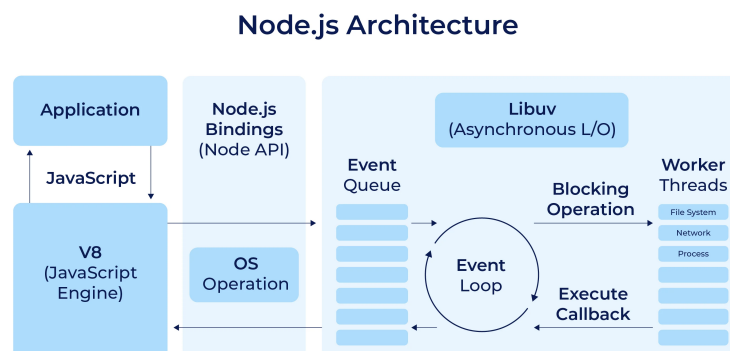


Figura 2.3: Gráfica de la arquitectura de Node.js (desde litslink.com) [3]

DISEÑO E IMPLEMENTACIÓN FUNCIONAL - SOFTWARE Y CÓDIGO

A continuación se detallan los conceptos considerados y la planeación de la parte técnica del trabajo realizado.

3.1. Conceptos para el diseño de la aplicación

En las siguientes secciones se explica el proceso de creación e implementación de la parte visual de la aplicación.

3.1.1. Interfaz Gráfica - Concepto inicial

Se crea un bosquejo para distribuir las proporciones de las distintas regiones de la aplicación en una ventana de navegador dentro de un computador, ésto se refiere a la resolución de aspecto ampliamente usada 16:9.

En el trabajo de planeación del proyecto, se establece que la aplicación consiste en una cantidad de ventanas distintas que contienen, cada una, la información más relevante para el usuario final en categorías. Ésta cantidad se establece en 5 inicialmente, una para elegir y seleccionar en la base de datos, de todos los pacientes registrados, a aquel cuyos datos se mostrarán, una para observar las fotos recibidas asociadas al paciente seleccionado, y otras tres para distintos gráficos asociados a los distintos signos vitales medidos: Electrocardiograma, oximetría, y presión sanguínea.

Al crear el proyecto en el ambiente de desarrollo Node.js y observando las capacidades de las herramientas, se cambian las ventanas en el proyecto por pestañas, siendo el usuario capaz de cambiar entre cada una con un click en su título, entre la lista de todas ubica-

das en una barra en un extremo de la ventana completa. Simulando así la apariencia de una aplicación de escritorio, ejecutándose en un navegador. La descripción estándar de una RIA es "una aplicación basada en web que tiene características de aplicaciones gráficas de [computadores de] escritorio" [4], por lo que se logra mantener la coherencia en la acción respecto a la planeación.

Para el espacio visual de la aplicación, se establece la distribución más general como se declara en la siguiente lista, con el código de color que se aprecia en la figura 3.1:

- Barra de navegación (Azul)
- Barra de pestañas (Naranja)
- Ventana activa (Verde)
 - Contenido visual o gráfico (Contorno Negro)
 - Contenido escrito (Barras Amarillas)

Es necesario aclarar que las últimas dos secciones son internas a la 'Ventana Activa', que varía según la pestaña que se desee visualizar.

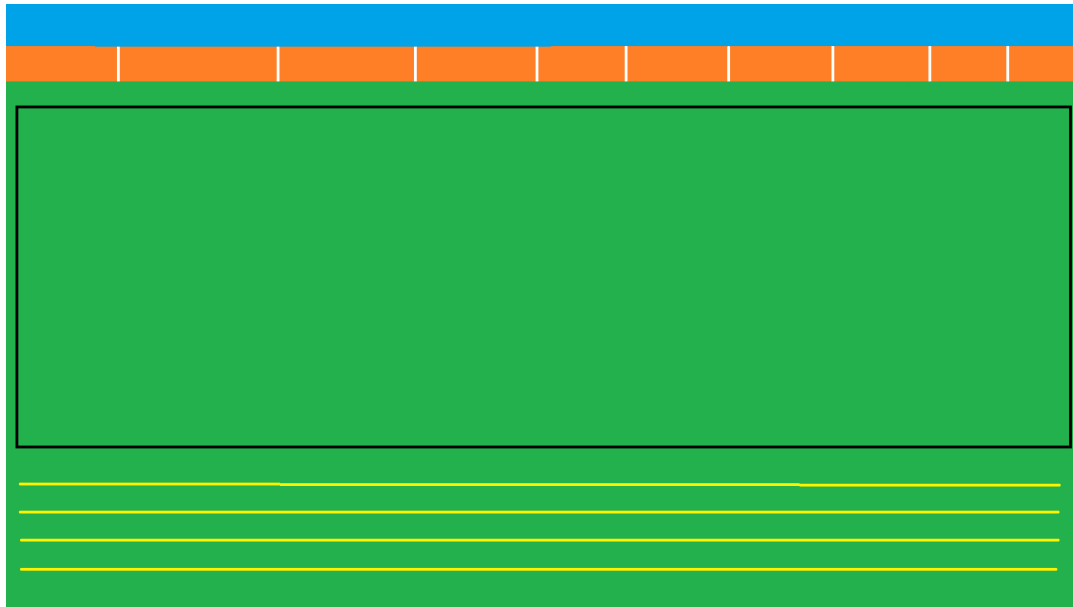


Figura 3.1: Bosquejo de colores del diseño de la aplicación

Este simple prefacio no abarca el contenido de cada área, y se basa en un enfoque de simpleza y alta área visual utilizada por los datos considerados, dada la naturaleza urgente de uso de la aplicación.

3.1.2. Interfaz Gráfica - Implementación de contenido

Considerando los contenidos a insertar en cada zona, tomando como referencia principal la versión anterior de la aplicación [8]: En la parte superior, dentro de la barra de navegación, el nombre de la aplicación y opciones de usuario y autenticación; práctica común en muchos sitios web.

En la sección de pestañas se considera solamente el título de la ventana correspondiente en cada una, pero también se observa la posibilidad de incluir una miniatura de los datos en vivo, como en el proyecto original [8], ilustrado en la siguiente figura.

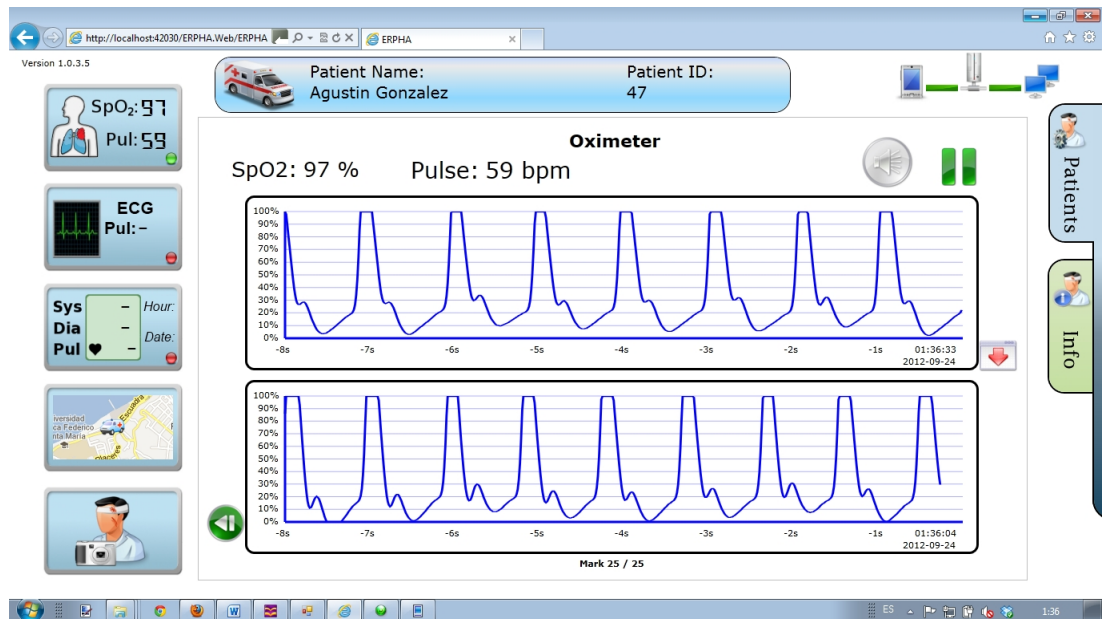


Figura 3.2: Interfaz del proyecto original, con miniatura de datos en vivo a la izquierda

La sección en verde del bosquejo es altamente dependiente de cada una de las pestañas, sin embargo lo que todas tienen en común es la inclusión de un visualizador gráfico además de actualizaciones de texto relevantes a la sección en la que se encuentran. Por lo mismo se propone la distribución dominante del contenido gráfico.

Profundizando en lo anterior, las pestañas permitirán visualizar cada una de las señales asociadas a los sensores presentes en el lado del equipo en terreno, además de la opción de ir al historial del paciente actual o cualquier otro registrado.

Al implementar las secciones, se agrega un espacio en la parte superior de la ventana, donde se ubicarán en todo momento el nombre completo, RUN y fotografía del paciente actual, si están disponibles. Posteriormente, se añaden páginas de registro o autenticación basadas en una plantilla, sin base de datos específica aún. Gracias a esto, se considera la adición de páginas adicionales para la selección y visualización de pacientes y su historial.

A partir de éste punto se pasa a la implementación de funcionamiento, considerando el plan trazado pero manteniendo mentalidad flexible en el caso de que se presenten problemas que obliguen a cambiar tácticas.

3.2. Planeación e implementación de la parte funcional

Se comienza estudiando distintas opciones de frameworks de JavaScript para utilizar, como *Qoosdoo*, *Express.js*, o *React*. Se pone enfoque en la opción más prometedora seleccionada llamada **React**, que funciona sobre el entorno Node.js.

3.2.1. Node.js

Se trata de un entorno runtime JavaScript multiplataforma, Node utiliza el motor JavaScript "V8". Para ofrecer perspectiva: este el núcleo que utiliza el navegador Google Chrome. Una aplicación sobre Node.js es un solo proceso que evita usar múltiples hebras, en cambio utiliza primitivas asíncronas en sus librerías por defecto. En general las librerías para Node se escriben evitando características bloqueantes, todo lo anterior permite evitar la *gestión de concurrencia de hebras lo cual sería fuente de muchos bugs* [9]. Además, en lugar de bloquear operaciones y gastar ciclos de CPU esperando, Node retoma el flujo cuando se obtiene una respuesta. Finalmente, el desarrollador escoge qué versión de ECMAScript usar basado en su versión de Node.js, lo cual implica que no será necesario mantener todas las plataformas relacionadas al uso o desarrollo actualizadas todo el tiempo, sean los navegadores de los usuarios o el entorno del programador. [9].

3.2.2. React

React es un framework que emplea el *bundler* Webpack para compilar código React, JSX, y ES6 mientras maneja prefijos CSS. Se habla de código React pues el framework nace de una librería de JavaScript del mismo nombre, que utiliza estructura declarativa para el lenguaje, con el objetivo de hacer la programación más predecible. Creada por Facebook/-Meta y también open-source, es utilizada especialmente para desarrollar características para interfaces gráficas.

Se debe destacar que el React utilizado en este proyecto es estricta y solamente una librería, y para hacer la aplicación se necesita un framework de fullstack, es decir que permite construir la parte del cliente tanto como la del servidor. Con este motivo se empareja React con el framework llamado **Remix**.

3.2.2.1. Framework Remix

Remix actúa como un compilador de código Javascript, administrador de HTTP del lado del server para mantener la seguridad de la conexión, un framework para mantención del funcionamiento del server, y framework para interacción navegadores, todo al mismo tiempo [10]. Está armado sobre la API WebFetch (Manera de implementar peticiones, compatible con todos los navegadores actuales al momento de escribir), lo que significa que puede ejecutarse sobre cualquier servidor similar a Node.js e incluso entornos como

Cloudflare Workers o Deno Deploy [11]. Remix también está enfocado a la interfaz de usuario, o UI en inglés, por lo que el método de funcionamiento interno cuando se trata del servidor funciona de manera que las rutas manejan URLs (Uniform Resource Locator) completas o parciales, y si se da una URL parcial, el compilador genera un controlador a partir de las direcciones completas posibles, y luego agrega lo necesario para mantener la UI funcional.

3.2.3. Descripción de módulos - Primera aproximación

En la presente sección se comienza describiendo simple y brevemente cada módulo y su función en la aplicación, para luego entrar en detalle técnico sobre su implementación y código.

Al construir la aplicación a partir de una plantilla provista por el mismo Remix, se le lleva a un estado suficientemente diferente para descartar los objetos de la plantilla, y comenzar a incluir gráficos en su interior con datos estáticos de ejemplo. Se continúa el desarrollo con las páginas de Registro e Ingreso (Sign in/Log in), las cuales cambian la ventana a una visual estándar de ingreso de correo electrónico y contraseña. Al momento de crear estas páginas no se implementa una base de datos de usuarios válidos, y se continúa el desarrollo con la creación de todo el esquema exterior o visible de la página, estructurando contenido variado y poco profundo para luego desarrollar en detalle por cada uno de los componentes.

Entonces, se genera una pestaña con un gráfico de muestra en ella, y una vez que se logra una presentación de resolución grande y que usa la mayor parte del espacio para facilitar su lectura a simple vista, el diseño se copia y pega para generar las tres pestañas para cada gráfico, modificando los parámetros para mostrar distintos colores y estilos en cada caso.

Se implementa adicionalmente una sección de ubicación: mostrando un mapa mediante la tecnología de Google Maps, con el objetivo de visualizar la ubicación del dispositivo que corresponda a la ambulancia seleccionada, respecto al hospital desde el cual se hace la petición.

Como función primaria para aplicar todas las anteriores, se crea una sección de selección de paciente. Su función es desplegar una lista de todos los pacientes históricos, incluida la señal en vivo que se pueda estar transmitiendo, para que el usuario pueda seleccionar los datos que desee observar, relacionados al paciente en el presente o en el pasado. Además, esta se plantea utilizar como pantalla inicial a la aplicación, con la perspectiva de abordar la prioridad de los datos en un flujo general de búsqueda: eligiendo el nombre de la persona/paciente primero, y luego dejando observar sus estadísticas.

3.2.4. Descripción de módulos - Revisión después de consulta con pares

A lo largo del trabajo, se considera el funcionamiento y eficacia de la interfaz tanto en ideas propias del autor como consultando a pares y al profesor referente. Gracias al intercambio de ideas se resuelve en alterar la aplicación basándose en un aspecto del trabajo original de Cristian Carrasco; se reimplementan las vistas permanentes de los valores de los signos vitales del paciente, desplazando la información personal a la barra superior del sitio, llamado usualmente encabezado o header. Las figuras 3.3, 3.4 muestran el estado de

la interfaz hasta este punto, en su primera corrección y su versión final, respectivamente.

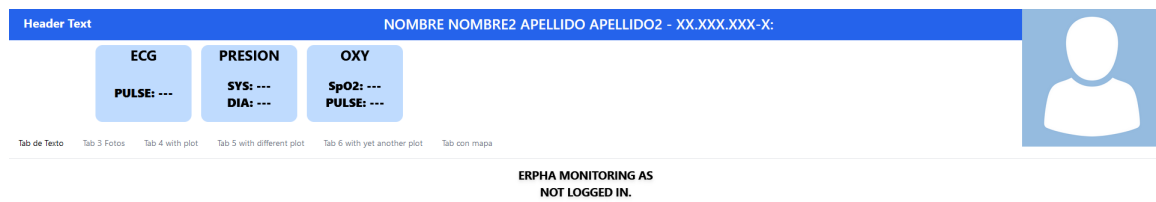


Figura 3.3: Captura de aplicación, con vista de signos permanente agregada

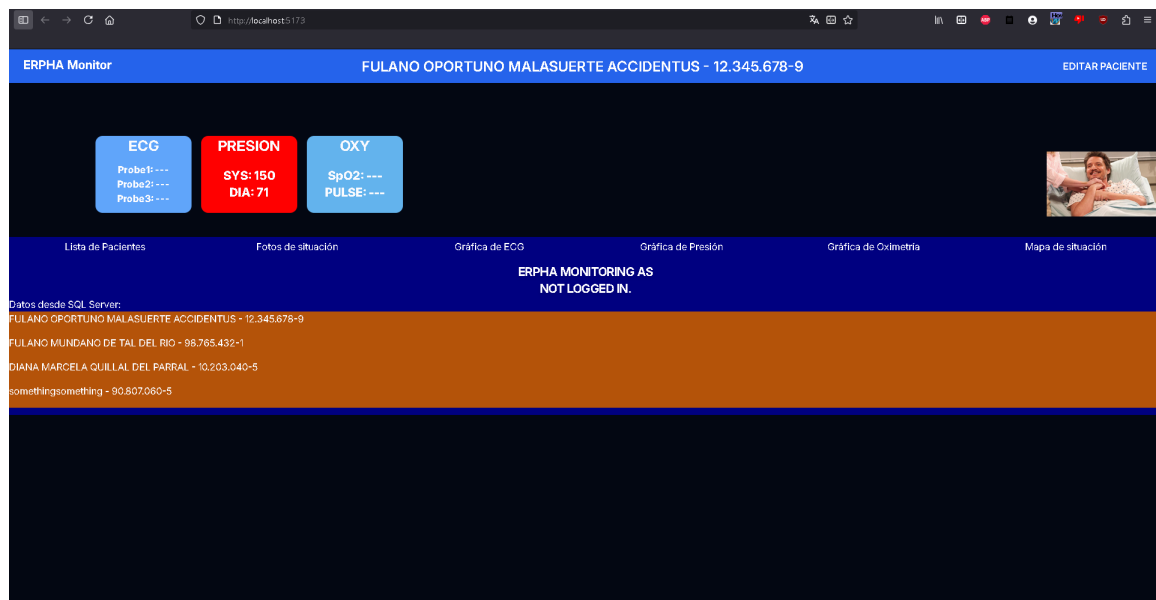


Figura 3.4: Captura de navegador con la aplicación y pacientes simulados como ejemplo, con vista de signos vitales permanente agregada.

El mapa también se profundiza, en primer lugar agregando un sistema de marcadores de lugar múltiples, para seguimiento de ubicaciones, e implementar seguimiento en tiempo

real, equivalente a seguimiento con actualización periódica de alta frecuencia.

Finalmente, para concluir la creación de todas las pantallas que se establecieron, se crea la visual de selección de pacientes. Ésta genera una lista basada en la información de la base de datos de pacientes, específicamente nombre completo y RUN, para que al hacer click o tap en el ítem de cierta persona, la aplicación cambie el contexto de los datos mostrados al de dicho paciente, cargando los datos históricos y comenzando a actualizar aquellos que se reciben en vivo, si aplica.

3.2.5. Optimización, mejora e integración de módulos

Una vez cumplidos los objetivos de ejecutar con éxito la aplicación básica de los sistemas anteriormente mencionados en la sección 3.1.1, se procede a trabajar en pulir su interacción y apariencia en la interfaz. Pues si bien cada uno cumple de manera aislada la función que estrictamente debe cumplir, se pueden hacer más cambios y afinaciones para que el funcionamiento como un todo sea óptimo.

Se comienza dando títulos y contenido adecuado a las distintas pestañas y secciones con texto de la aplicación, con el fin de no generar duda alguna sobre el contenido de cada pestaña o el propósito del texto en su ubicación. Para cumplir lo anterior junto a una muestra práctica del funcionamiento de las funciones desarrolladas, se crea una base de datos con SQLServer en el equipo local de desarrollo, que contendrá varios nombres y RUN ficticios, junto a datos de mediciones ficticias también.

Acto seguido, se implementan las funciones `useEffect` y `useState` nativas de React para manejar el cambio de los valores de las variables en el backend, y hacerlo visible en el frontend.

Lo anterior se emplea primero en hacer que el sitio efectivamente cambie la información del paciente en la barra superior al hacer click en algún miembro de la lista, iniciando la aplicación con el texto “ELEGIR PACIENTE” en el lugar, y reemplazándolo con el nombre y RUN del paciente de la lista sobre el cual se haga click. Al completarse dicha tarea, se pasa a la siguiente: El bucle de actualización del sitio.

Se continúa estructurando un prototipo de rutina de actualización, que hace cambiar los valores de las ventanas flotantes y actualiza el estado de la data asociada a los gráficos, para que se muestren en tiempo real o lo más cercano a ello posible en cada uno. El ritmo de actualización se establece en base al *bitrate* y para su fácil observación, en 1 Hz.

Para verificar el correcto funcionamiento del programa de manera expedita, en primer lugar se evita el uso de la conexión externa y se aplica generación de números aleatorios en su lugar. Así, en cada ciclo de la rutina de actualización del sitio se genera automáticamente un valor al azar, dentro de rangos promedio para las señales a probar (como presión sistólica, diastólica, o saturación de oxígeno en la sangre) con el motivo de ver cambios reflejados en la aplicación en funcionamiento sin tener que depender de una parte ajena al sistema, evitando mayores problemas. Tras ajustes en la aplicación correcta de las funciones utilizadas en su contexto, la táctica implementada muestra resultados satisfactorios.

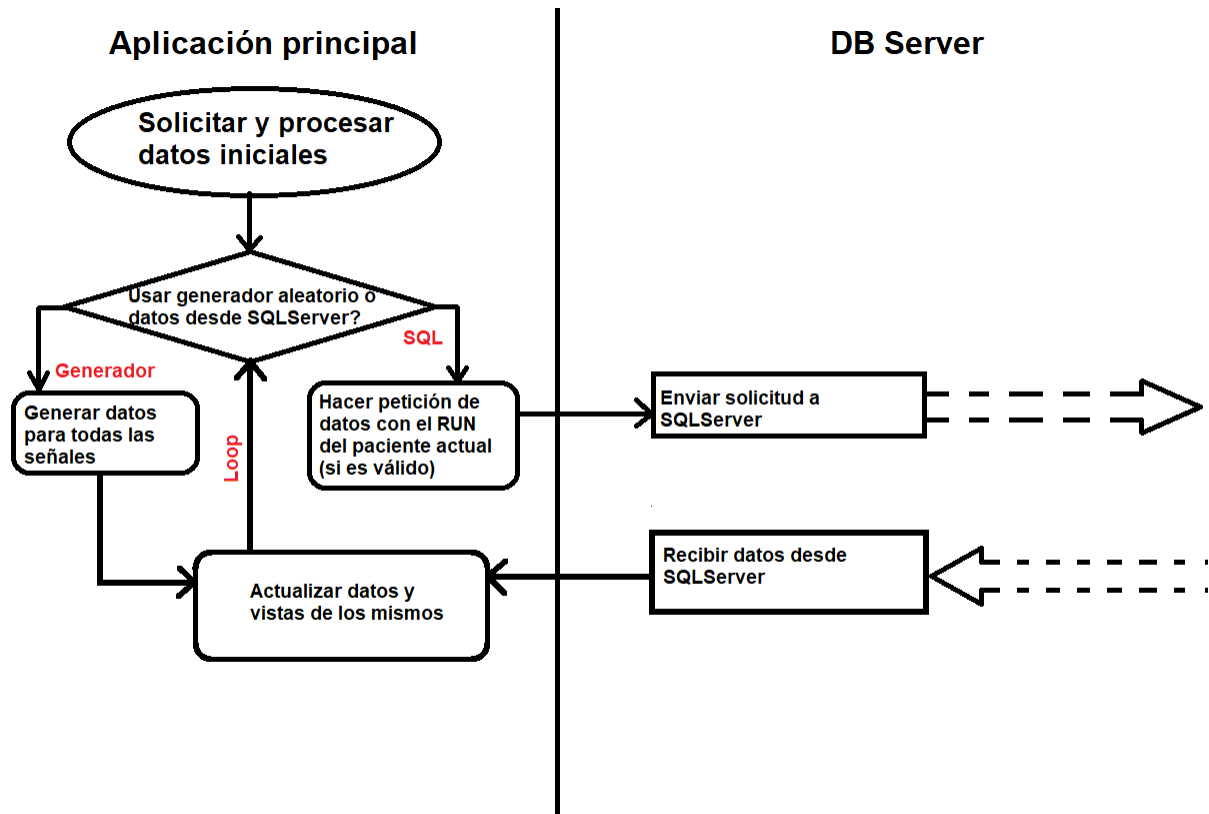


Figura 3.5: Diagrama de flujo de datos mostrados en la aplicación, en su versión de muestra

Revisando todas las funciones dinámicas que se implementan hasta este punto, se establece el siguiente objetivo inmediato: implementar la función de tomar capturas de los gráficos y guardarlas para su posterior revisión o referencia. Junto a esto se observa la posibilidad de implementar la característica de ver gráficas de datos pasados.

Junto a lo anterior se considera también un sistema de advertencia visual; hacer que las ventanas flotantes de signos vitales cambien de color dado el estado de las señales asociadas a ellas, al momento de escribir este documento se conviene, para conveniencia de observación, en utilizar un promedio de todo el arreglo de las señales i.e. los valores visibles en la ventana de gráficos, y compararlos con un valor ligeramente superior a la mediana del rango promedio de cada señal, con el único fin de observar el comportamiento de cambio más frecuentemente que en un patrón de valores sanos.

Se implementa también la función de tomar instantáneas de los gráficos en ciertos momentos, permitiendo mantenerlas abiertas para consultar en cualquier momento, a un lado de los gráficos que se siguen actualizando, observable en la figura 3.6. Además, se aplica la opción de editar los datos del paciente que se visualiza actualmente para correcciones que se requieran después de su ingreso.



Figura 3.6: Captura de pantalla de función que captura instantáneas de los gráficos, habilitada en la aplicación

Luego de concluir el desarrollo de las funciones hasta este punto, se considera no proseguir con las demás ideas antes propuestas, dado el corto tiempo restante para dar conclusión al trabajo.

Finalmente, al tener todos los implementos previstos completados, se termina por configurar la asignación y cambio de todos los datos dependiendo del sujeto en el cual se seleccione. Para hacer la diferencia más aparente, al seleccionar un candidato diferente del primero en la lista de ejemplo para pruebas, el código de la aplicación hace solicitudes al servidor SQL, sobre los datos que estén asociados al RUN del sujeto correspondiente: Y si el servidor no está activado, la base de datos no poblada o no actualizada, la aplicación no mostrará cambios en las señales.

3.3. Detalle de código

En la presente sección se presentan y explican en detalle de las funciones, clases, métodos y demás formatos de código utilizado para solucionar los problemas necesarios para completar el trabajo, y los que surgen durante la implementación de las dichas soluciones.

3.3.1. Cimientos de ERPHA Monitor

Como se especifica en el Capítulo 3, se utiliza el runtime Node.js como la capa más basal de la ejecución, luego se utilizan las librerías React y el framework Remix para construir la

aplicación y su lógica en Javascript, con el ambiente **Visual Studio Code** para la interfaz de escritura y desarrollo, y finalmente el navegador Mozilla Firefox para ejecución y debugging utilizando la consola del mismo apoyada por un componente agregado para Firefox y otros navegadores llamado "React Developer Tools".

3.3.2. Sección de interfaz gráfica

La parte de cálculo, lógica y en general el backend de la aplicación componen la mayor parte del archivo origen, index.tsx. La parte final constituye el frontend, o la descripción de cómo se compone y ordena todo lo que se ve en el navegador al acceder a la aplicación. Ésta se escribe puramente en formato HTML, a continuación se presenta una muestra de dicho formato para entrar en contexto.

```
<div> { /* Barra */}
  <Navbar
    inputValue={Nombre}
  />
  { /* CUADROS FLOTANTES */}
  <div className="relative ml-32 w-96 w-full">
    <CajaECG Value1={[]} Value2={[]} Value3={[]} />
  </div>
  <div className="relative ml-80 space-x-44 w-96 h-12">
    <CajaPresion Value1={Arreglo1} Value2={Arreglo2} Value3={[]} />
    <CajaOxigeno Value1={Arreglo3} Value2={Arreglo4} Value3={[]} />
  </div>
  { /* IMAGEN */}
  <picture className="relative">
    <img srcSet="ImagenFuente.png" width="233" height="233" align="right" className="object-cover" />
  </picture>
</div>
```

Figura 3.7: Fragmento de código HTML, comentado, que describe varias secciones de una página de la aplicación

3.3.3. Sistema de petición y muestra de datos

El código contiene ciclos que envían peticiones al servidor SQL, luego actualizan las variables con los datos adquiridos de dichas peticiones, y también los elementos gráficos como las imágenes, indicadores, o gráficas. Todo se asocia al RUN de un paciente ya que es único, y funciona como clave ideal para un sistema de identificación. Se provee un concepto visual en la figura 3.8:

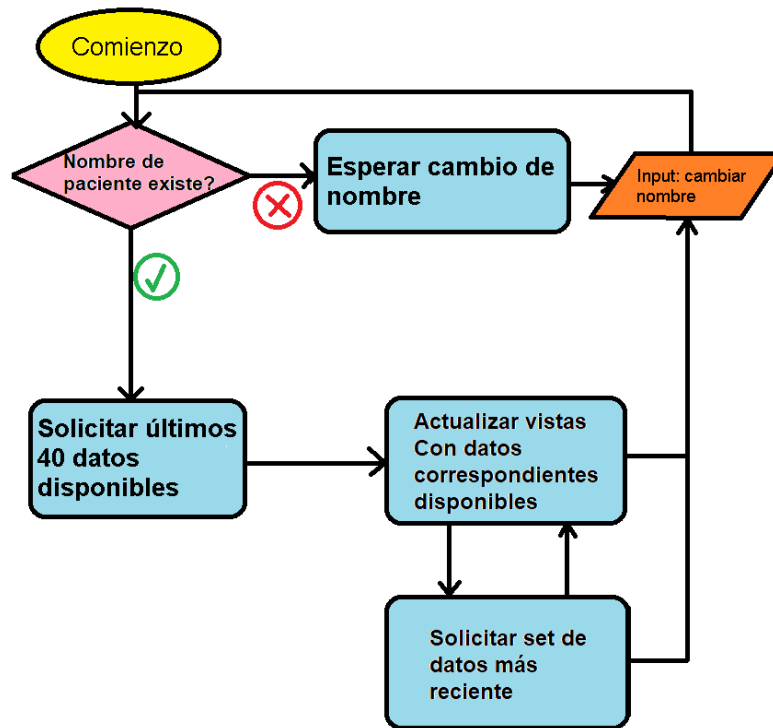


Figura 3.8: Diagrama general de funcionamiento de la aplicación

3.3.4. Gráficos con Chart.js

Cada visual de gráficos se logra con la librería Chart.js. La pestaña de los gráficos de ECG, observable en la figura 3.9, emplea 3 gráficos individuales para cada señal, en una disposición vertical. El gráfico para la presión es una ventana para dos señales, al igual que el gráfico para oximetría.

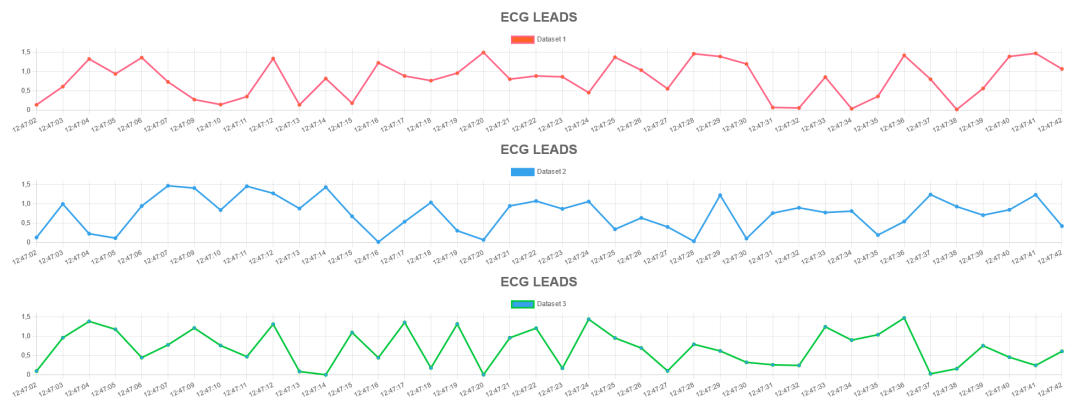


Figura 3.9: Captura de gráficos de ECG

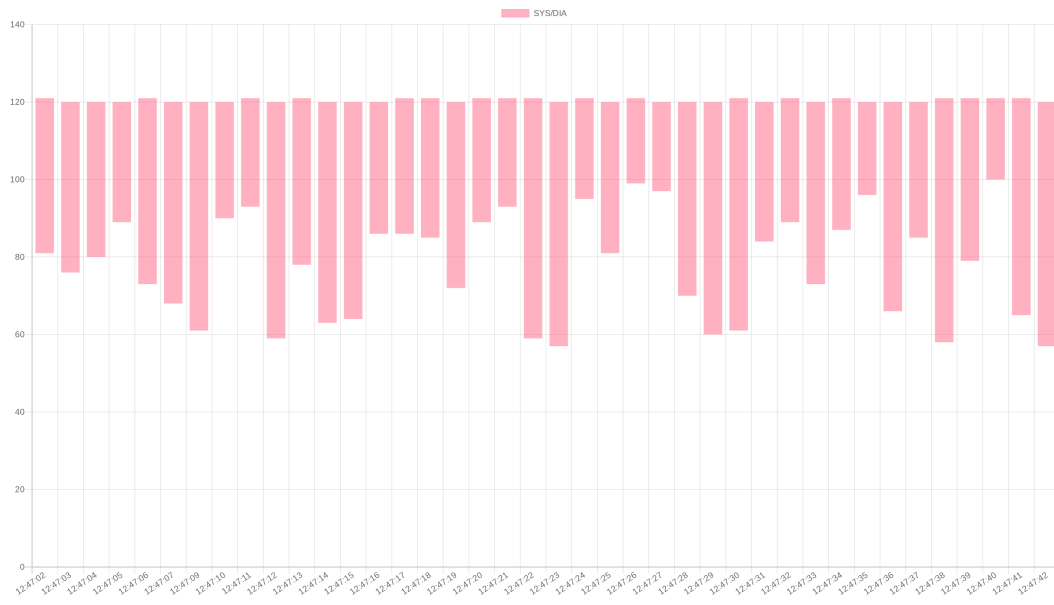


Figura 3.10: Captura de gráfico de presión vs. tiempo de recepción, con medidas excesivas graficadas para visualizar diferencias

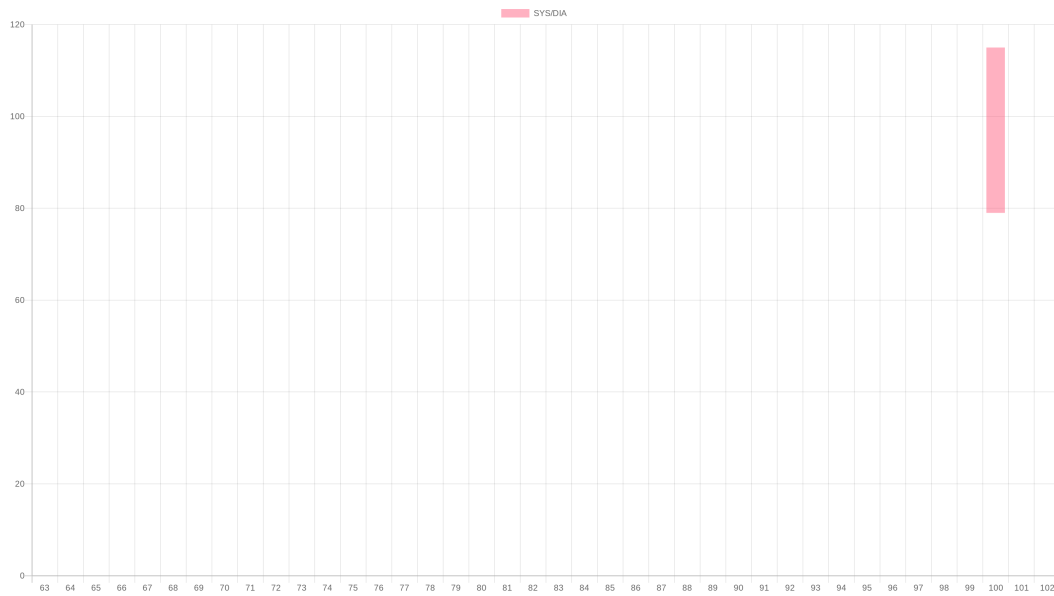


Figura 3.11: Captura de gráfico de presión vs. número de unidad de tiempo, en funcionamiento

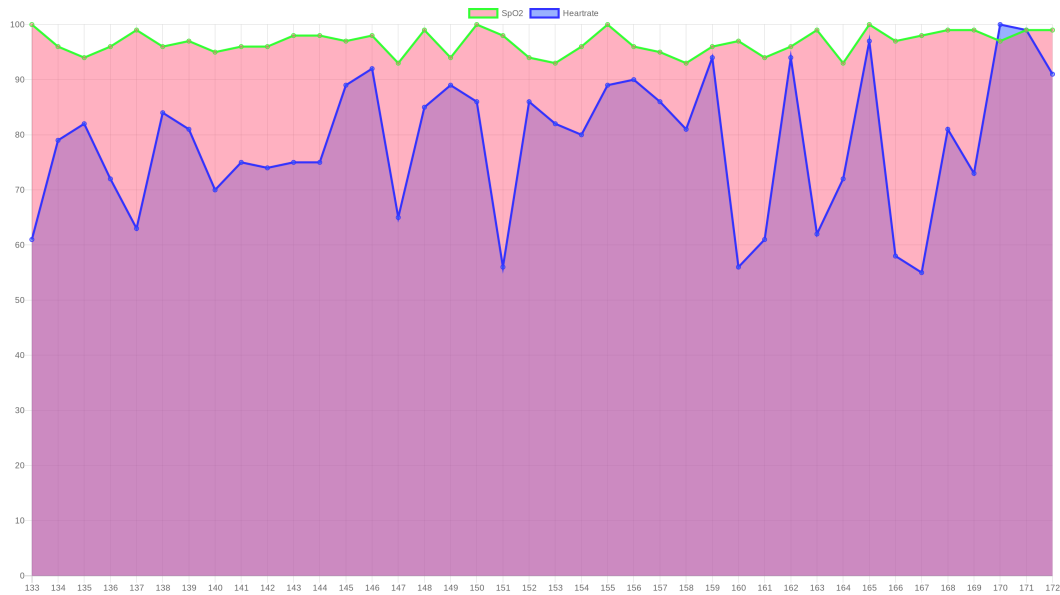


Figura 3.12: Captura de gráfico de oximetría, en funcionamiento

Se debe aclarar que el gráfico de presión de la figura 3.10 es una simulación no fiel a la aplicación usual, y se muestra para poder observar una variedad de valores en la ventana.

3.3.5. Captura de gráficos

Para implementar la función de capturar una imagen del gráfico, se utiliza una librería llamada "Chartjs-to-image", la cual utiliza el mismo contexto y formato en las opciones para configurar un gráfico, sólo que genera una imagen en lugar de un objeto web. Por ende, se puede reutilizar la configuración del gráfico que se muestra en el instante en el que se hace click en el botón. Lo que genera una imagen que se abrirá en una nueva ventana, como alternativa al ser incapaz de aplicar correctamente la función de generar un archivo descargable y dar el mensaje para guardarlo, esta solución es una aproximación adecuada, dado que se puede guardar la misma imagen con dos clicks adicionales, y mantiene la ventana abierta a un lado para mantener la observación en vivo disponible.

Es importante destacar que lo anterior aplica sólo para los gráficos individuales y para el caso de los gráficos de ECG, que son varios con sus configuraciones por separado, se utiliza la biblioteca "html2canvas" para transformar el contenedor HTML que aloja a los 3 gráficos en una imagen, y abrirla en una ventana aparte para ser consistente con la aplicación anterior.

3.3.6. Mapas de Google Maps

A lo largo del año, se trabaja con la API de google maps, mostrando la ventana de mapa en la pestaña correspondiente y afinando su función. Para esto, Google requiere de una cuenta, que a su vez debe solicitar una "API Key": una clave para validar el uso responsa-

ble del sistema de mapas en un contexto ajeno a la organización que lo diseñó. Durante el tiempo dado para terminar el proyecto, no se logra implementar correctamente un sistema de marcadores móviles para seguimiento en tiempo real de la posición de los pacientes. Sin embargo, se conoce la posibilidad de dicha característica, empleada como ejemplo demostrativo en la página de la librería *react-google-maps*, por ejemplo. [13]

3.3.7. Función Loader, inicio de la ejecución

Al trabajar en Remix, y escribiendo en la sintaxis de Javascript, se especifica una función loader por defecto, que funciona para efectuar acciones contenidas en ella antes o durante la carga del primer *render* del sitio. En ésta se ejecuta una llamada a la base de datos para rescatar la lista de pacientes junto a las últimas 40 entradas, cantidad arbitraria para efectos de prueba/muestra. Adicionalmente, éstos datos se configuran en varios objetos y variables necesarios para pasar a las componentes que generan los gráficos de la librería "Chart.js".

Todos los datos generados en este paso son empaquetados en formato JSON y entregados a la función "Index", que cumple la función de ejecutar la gran mayoría del programa, similar a la función "main" de otros lenguajes. Mediante la función nativa de Remix "useLoaderData" se reciben todos los datos. A continuación pseudocódigo para explicar este flujo:

```
mssql.Rescatardatos("DatosSQL");

Función-Loader(DatosSQL, datos1, datos2, [...]);
  Asignar(DatosSQL, datos1,datos2,[...]);
  dar-formato-para-chart(datos1);
  dar-formato-para-chart(datos2);
  ...
  Juntar-formato-JSON(datos1,datos2,[...],todo-en-JSON);

Index(){
  var arreglo1,arreglo2,[...];
  useLoaderData(todo-en-JSON, arreglo1, arreglo2, [...]);
  //Todo el resto del programa
}
```

3.3.8. Función Index y ejecución de la aplicación

Al pasar a la función Index, el código pasa a la sección que se ejecuta junto a la presentación del sitio en HTML. Aquí se declaran todas las variables finales a utilizar, junto con las funciones y bucles sobre los cuales el sitio funciona.

En éste punto se observa un problema en la implementación de la aplicación para su demostración: Al no tener una fuente de datos de signos vitales, no se puede observar ni afinar el funcionamiento al momento de crear la aplicación por vez primera. Entonces, se implementa un pequeño sistema local para emular las partes del sistema ERPHA completo:

Un script en Python que genera y envía signos vitales, cuyo paradero será en una instancia de SQLServer ya mencionada anteriormente y desde la cual se solicitarán en la aplicación.

El script de Python utilizado es una modificación ligera de "Vital-signs-simulator" por Michelle Bark, el cual se vale de generación aleatoria con cotas para distintas señales. [2] El servidor SQL se administra mediante la herramienta Microsoft SQLServer Management Studio 20, y las bases de datos se implementan en un esquema compacto.

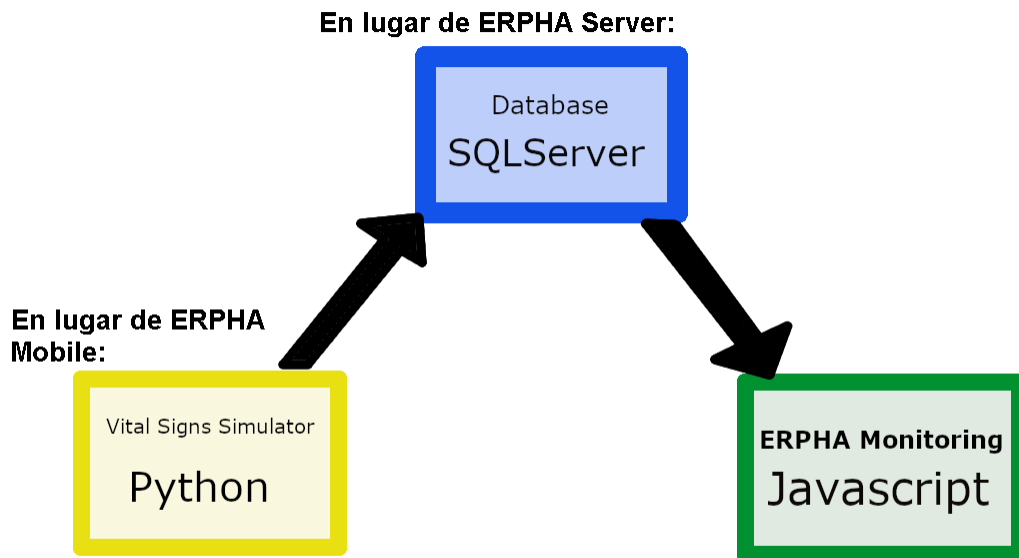


Figura 3.13: Figura del sistema de prueba local

Sin embargo, otro problema surge al utilizar la librería "mssql": sólo se puede tener un objeto de conexión activo, y al utilizarlo en la función loader de Remix, no se puede pasar ni utilizar en el contexto o scope de la función Index.

El inconveniente anterior se aborda utilizando un método distinto para incluir en el ciclo que ocurre mientras el sitio está activo. Se genera un archivo adicional llamado "db.server.ts", que usa las librerías *mssql*, *cors*, *express*, y *body-parser*. El rol de este archivo es montar un pequeño servidor local que responde a peticiones tipo POST en cierto puerto de la red donde se monta. La respuesta a éstas peticiones son los resultados de las *queries* de SQL dirigidas al servidor.

3.3.9. Interacción entre Javascript y HTML

El que convivan dos lenguajes en un sólo archivo lleva a pensar en cómo se podrían manejar los cambios en uno para reflejarse en el otro: si bien un sitio escrito en javascript tendrá sus cambios cuidadosamente estructurados y coherentes, no significa que la secuencia de HTML siga las mismas reglas, al mismo tiempo siquiera.

Se sabe que HTML incorpora secciones de código con la etiqueta `<script>`, sin embargo esto existe como una llamada activa desde el sitio hacia el código. Para implementar una solución adecuada con el presente trabajo, se necesita que el código se ejecute continuamente en el backend, y que el sitio en el frontend refleje los cambios que ocurren. Para resolver esta situación existe el *hook* `useState()` de React.

`useState` funciona al crear una variable de estado para el contexto del sitio, la cual tiene las propiedades de ser persistente entre renders, y que al ser cambiada activa un render adicional del sitio, volviéndolo a construir con el valor nuevo y por ende, mostrando los cambios que sufra en el frontend. [6]

Con esto en mente, todos los valores que se necesiten para mostrar cambios se inicializan de la siguiente manera:

```
const [variableName, SetvariableName] = useState(value)
```

Se utiliza *const* para evitar que se trate como variable según la definición del lenguaje Javascript y que dada alguna condición que escape al cuidado del programador, `variableName` pierda el valor guardado. Luego se le da un nombre a la referencia del valor y se crea un *'Setter'*, que será la única manera de que el valor de la variable original cambie. Finalmente se usa el Hook con el valor que se quiere dar a la referencia como único argumento, el cual puede ser un objeto, string, arreglo, etc. El Setter funciona para cambiar el valor cuando es llamado con un argumento diferente pero con la misma forma y/o tipo, pues al no ser una variable, no se le puede hacer asignación directa o modificarse con operaciones, lo cual da un nivel de seguridad y constancia a los valores que así se asignan.

Es de notar que React tiene un estándar cuando trata con Hooks: dos reglas que se deben seguir para que sean efectivos:

```
Debe de ser llamado en el nivel superior de la aplicación - Un hook nunca debe
de llamarse dentro de ciclos, condicionales o funciones anidadas [...].
Este uso únicamente en el nivel superior es lo que asegura
que el estado interno de React se preserve correctamente entre
diferentes llamadas del mismo hook.
```

```
Debe de llamarse en funciones o en otros hooks personalizados
de React - Un hook nunca debe de ser llamado
fuera de una función de React o de otro hook personalizado,
de forma que la lógica de estado del componente
sea claramente visible desde el resto del código para el scope establecido por React.
```

-Fuente: OpenWebinars. [5]

Lo anterior puede poner en duda el funcionamiento de la aplicación, en el sentido de que se alude a utilizar bucles para que funcione continuamente. Esto se soluciona mediante otros hooks, llamados **useEffect** y **setInterval**.

useEffect hace que el contenido reaccione y se ejecute frente a estímulos exteriores determinados, es decir: se le entrega una variable para que esa instancia del hook esté monitoreando, y cada vez que dicha variable cambie, se activará el código dentro de la función *useEffect*. Esto es útil en la situación del cambio de nombre y datos del paciente, pues se activa la funcionalidad de buscar los datos solamente cuando se da el evento de hacerle click a un nombre de la lista, y mientras eso no pase, el resto del código seguirá ejecutándose.

setInterval es un hook que permite el uso de bucles, pero es estrictamente temporal, pues se configura un periodo de tiempo en milisegundos que pase entre iteraciones. Este *hook* se utiliza para la actualización de los datos, poniendo condiciones dentro del mismo para consultar si los datos son efectivamente distintos para asegurarse de mantener la integridad y continuidad de la información plasmada en los gráficos, utilizando la hora en la que fueron subidos, la cual forma parte de los datos al menos en la prueba local efectuada.

De esta manera, se mantiene la regla de los hooks para su uso en toda la estructura, conservando la continuidad y congruencia del funcionamiento interno de Remix React, evitando bugs indescifrables que requieran interferir en el código interno y propio del framework.

PRUEBAS Y RESULTADOS

4.1. Pruebas aisladas de módulos

Las funciones de interacción con el usuario como clicks, respuesta a cambios hechos por los mismos, y deslizamiento de la página con las barras laterales o la rueda del mouse fueron probadas a lo largo del desarrollo, pues son parte clave de la aplicación y necesarias para avanzar en las siguientes pruebas, aunque sean individuales.

Al ejecutar los módulos de recepción y asignación de datos, primero observando a través del depurador en la consola de Firefox los valores de las variables, y luego asociándolo a los gráficos para que muestren las señales, se confirma que lo que se genera aleatoriamente de manera local, o aquello que se envía al servidor SQL se recibe en la aplicación correctamente y sin pérdidas; un 100 % de recepción de datos.

Lo anterior nos lleva naturalmente al funcionamiento de los gráficos, como muestra visible del funcionamiento de ambas partes.

Como cuantificador para el caso de los gráficos, en un promedio de 6 pruebas al día, a lo largo de 2 semanas acumulativas de desarrollo y pruebas de los sistemas, los sistemas de gráfico no muestran bugs ni cualquier comportamiento indeseado que no se atribuya a incongruencias en los datos que grafican, por lo que se les considera confiables.

Para el sistema de mapas, la vista e interacción funcionan tal como lo hacen en el sitio oficial de Google, por lo que la implementación es exitosa. Sin embargo, la clave API Key de Google requiere pagos periódicos para mantenerse habilitada, por lo que durante el desarrollo se utilizó el período de prueba gratuita ofrecido por Google.

Se destaca entonces: **La organización que vaya a utilizar ésta aplicación, debe asociar una API Key de google maps válida, e ingresarla en la constante "GOOGLE_MAPS_API_KEY" en los archivos del sistema: ClienteERPHA/app/components/MapApp.js, o reemplazar por completo el**

módulo de mapa implementado.

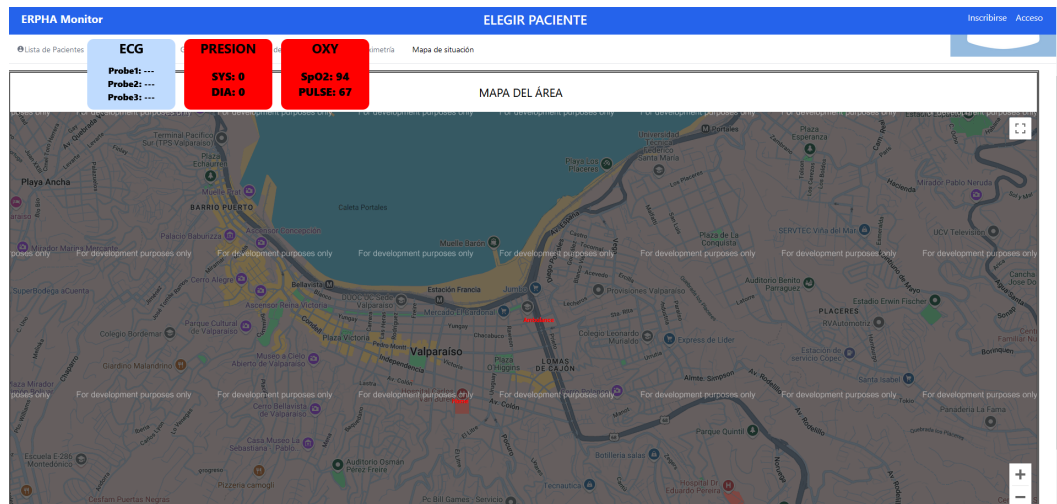


Figura 4.1: Vista de la pestaña de Mapa en la aplicación

4.2. Pruebas de módulos integrados

4.2.1. Respuesta de conexión a la aplicación

Se observa en primer lugar, la consola provista por el ambiente de runtime Node, que declara cada vez que una conexión se efectúa al sitio.

```
PS C:\Users\admin\Desktop\MEMORIA\React\ClienteERPHA> npm start

> start
> remix-serve ./build/index.js

[remix-serve] http://localhost:3000 (http://192.168.56.1:3000)
GET / 200 - - 175.928 ms
GET / 200 - - 63.403 ms
GET / 200 - - 51.881 ms
GET / 200 - - 52.854 ms
```

Figura 4.2: Consola de Node, luego de iniciar la aplicación

La figura 4.2 muestra una consola con cuatro instancias de solicitud GET, hechas por Wi-Fi en la red local de una vivienda, las dos primeras en el mismo computador en el que se ejecuta la aplicación (Win10), la tercera desde una laptop (También Win10), y la cuarta desde un celular (Android). Los mensajes de conexión se muestran gracias a la configuración por defecto del entorno

4.2.2. Métodos de conexión

Las pruebas realizadas para confirmar el correcto funcionamiento de la aplicación se hacen montando el programa con el entorno Node. Así, el computador donde se desarrolla la aplicación es el que compila el programa para luego entregarlo al navegador que se conecte. Esto significa que la aplicación sólo está disponible para la red local, conectando mediante la dirección IP.

4.2.3. Estabilidad de conexión

Al utilizarse Node.js como el framework para la aplicación, la labor de conexión queda para el proveedor de internet de quien, y la firewall del equipo donde se ejecuta la aplicación y los protocolos que ésta tenga. Sin embargo, Node cuenta con soporte nativo para conexiones con protocolo HTTP y UDP, así como librerías de terceros para implementar otros protocolos como FTP. Para los motivos prácticos de la aplicación y la simplicidad de uso, se sugiere y fomenta el uso de HTTP.

4.2.4. Uso de recursos

Al desarrollar pruebas con conexiones locales, se observa carga baja en el sistema, tanto en el uso de memoria del navegador como el CPU del equipo que ejecuta el cliente, y el que hace la petición; el porcentaje de uso de CPU no incrementa más allá de 10 % del valor base al cual opera sin programas en primer plano, mientras que el uso de la memoria no aumenta más de 400MB, según el software administrador de tareas nativo de Windows.

CONCLUSIONES Y TRABAJO FUTURO

5.1. Conclusiones

Al ver el trabajo integrado por completo, los resultados de las pruebas fueron los esperados a partir del planteamiento teórico. Además, se observa el formato relativamente compacto del código; en menos de 2000 líneas de código personal escrito se puede generar una aplicación de complejidad e interacción considerables, dejando en evidencia la accesibilidad del método empleado: El montaje de la aplicación en Node sumado a las librerías de acceso público entregan generosas opciones para trabajar en el desarrollo web, poniendo el enfoque del trabajo en el conocimiento del desarrollador en cuanto a los detalles de los computadores y sistemas en profundidad.

Los métodos utilizados para efectuar las pruebas, si bien son adecuados, no reflejan fielmente el ambiente de funcionamiento planeado. Se espera que se prueben todas las partes del sistema completo una vez terminadas todas las partes para ejecutar pruebas que entreguen resultados más robustos y que aporten con mejor precisión al proceso de corrección de problemas y ajuste de detalles de las partes, con el fin de que el sistema ejecute su función sin fallos mayores. Se debe tener en cuenta que durante el trabajo no se tuvo consultoría de personal médico o relacionados, por lo que la visualización de ciertas señales podría ser no óptima para el rubro. Además, el plazo de tiempo en el cual se desarrolla el trabajo no permite terminar las funciones de absolutamente todos los módulos, principalmente el de ubicación, sobre el cual no se pudo completar la función de seguimiento de ubicaciones en tiempo real o por períodos. Ésto debido al enfoque en el funcionamiento de transmisión de datos por sobre lo demás.

Existen alternativas en Node para montar servidores http, como Apache por ejemplo. Al hacerlo de ésta manera, se puede elegir el servicio que mejor acomode para servir la aplicación, además de la posibilidad de hacer pruebas abriendo la conexión local a la internet exterior haciendo un port-forwarding en el router de red de la ubicación o empresa donde se monte.

5.2. Trabajo Futuro

El paso más evidente a tomar es el de concluir el funcionamiento de los módulos logrados parcialmente, buscando métodos para implementar el funcionamiento necesario entre el código del backend y el del frontend. Además, se espera tener todo el sistema ERPHA en funcionamiento en el futuro cercano, para corroborar el correcto funcionamiento en conjunto de las tres partes, y corregir errores y bugs en caso de que aparezcan para proponer y liberar la solución al sector público, comenzando en la región de Valparaíso.

En un futuro más alejado se espera poder desarrollar software más sofisticado, de operación simple y/o de respuesta más veloz, considerando el amplio repertorio de librerías que Node contiene en la actualidad, y proyectando dicho ritmo de desarrollo al futuro. Se cuenta también con la evolución del hardware, observando la posibilidad de integrar dispositivos de monitoreo con transmisión directa a internet, evitando el teléfono celular como intermediario para el cliente del sistema, lo que conllevaría posiblemente a adaptar la aplicación de visualizador y la base de datos para el formato que dichas mejoras impongan.

6.1. Notas adicionales

Como consideración: durante el lapso de tiempo entre la creación del presente informe y su revisión final, se da una actualización relevante en la plataforma de Node. El cual adoptó Vite, una herramienta de compilación que acelera y hace más eficiente el proceso de *build* de las aplicaciones, junto con otros cambios menores varios. [12] Dicha actualización conlleva una revisión de la compatibilidad de la aplicación con la plataforma, que se resume en el siguiente párrafo.

Se observan errores que impiden el uso en un comienzo, y tras un análisis diagnóstico, se observa que la mayoría del código de base HTML y Javascript escrito sigue siendo funcional, siendo lo que causa errores las bibliotecas de componentes externos como “react image gallery”, lo que deja la constancia de los pros y contras de las mismas; se debe considerar que el desarrollador original adapte su componente, o darse el trabajo propio de comprender el código de otra persona para adaptarlo por iniciativa propia. Además, se da la necesidad urgente de reemplazar las funcionalidades perdidas para motivos de presentar el trabajo realizado.

En fin, para motivos de funcionamiento, se excluyen los módulos externos que causan los errores e incompatibilidades, tratando de mantener el aspecto anterior, ya que en su mayoría la funcionalidad se conserva sin sacrificar el aspecto.

BIBLIOGRAFÍA

- [1] Carabineros de Chile, Conaset - Estadísticas de accidentes de tránsito [conaset, en línea] <https://www.conaset.cl/programa/observatorio-datos-estadistica/> [consulta: 31 marzo 2024]
- [2] BARK, Michelle. Vital-signs-simulator [github, en línea] <https://github.com/MichelleBark/Vital-signs-simulator> [consulta: 3 julio 2024]
- [3] PETRENKO, Viacheslav. Node.js Architecture From A to Z: What Makes It a Top Choice [litslink, en línea] <https://litslink.com/blog/node-js-architecture-from-a-to-z> [consulta: 2 septiembre 2024]
- [4] KAY, Russel. Rich Internet Applications [computerworld, en línea] <https://www.computerworld.com/article/1681941/rich-internet-applications.html> [consulta: 17 diciembre 2024]
- [5] HUET, Pablo. React Hooks: Qué son y qué problemas solucionan [openwebinars, en línea] <https://openwebinars.net/blog/react-hooks-que-son-y-que-problemas-solucionan/> [consulta: 13 agosto 2024]
- [6] Documentación REACT. useState [react.dev, en línea] <https://react.dev/reference/react/useState> [consulta: 13 agosto 2024]
- [7] SUYOG Pallavi. "What Is the Golden Hour in Medicine?" [medicinenet, en línea] https://www.medicinenet.com/what_is_the_golden_hour_in_medicine/article.htm [consulta: 13 mayo 2024],
- [8] CARRASCO Cristian, "INTEGRACIÓN DE SERVICIO DE MEDICIÓN DE SIGNOS VITALES EN DISPOSITIVO MÓVIL CON CONECTIVIDAD 3G Y SU DESPLIEGUE EN UNA APLICACIÓN WEB" Memoria de titulación (Ingeniería Civil Electrónica). Valparaíso. UTFSM, Departamento de Electrónica. 2013.
- [9] HALVERSON Ben, et al., "Introduction to Node.js" [nodejs.org, en línea] <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs> [consulta: 2 septiembre 2024]
- [10] JACKSON Michael, et al., "Introduction, Technical Explanation" [remix.run, en línea] <https://remix.run/docs/en/main/discussion/introduction> [consulta: 2 septiembre 2024]

- [11] GIULIANI, Facundo. “¿Es Remix JS el Próximo Framework Para Ti?” [escuela frontend, en línea] <https://www.escuelafrontend.com/remix-js> [consulta: 10 agosto 2024]
- [12] WUNDE, Claudio, et al., “Node.js releases” [nodejs.org, en línea] <https://nodejs.org/en/about/previous-releases> [consulta: 20 enero 2025]
- [13] SCHUHFUSS, Martin, et al., “React Google Maps” [github, en línea] <https://visgl.github.io/react-google-maps/examples/markers-and-infowindows> [consulta: 4 enero 2025]

COPYRIGHT

Some sections and figures of this document are protected by copyright laws. Please contact the Institute of Electrical and Electronics Engineers, IEEE, the Universidad Técnica Federico Santa María or the author prior to any non-personal use of this material.

