CITEC

Sistema informático para evaluar riesgo de complicaciones respiratorias o muerte a pacientes con Covid-19.

Trabajo de diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autor(es): Reynaldo Cesar González Prieto
Tutor(es): Dayana Joseph Smarth

La Habana, diciembre de 2022

Año 64 de la Revolución

DECLARACIÓN DE AUTORÍA

El autor del trabajo de diploma con título "***Sistema informático para evaluar riesgo de complicaciones respiratorias o muerte a pacientes con Covid-19¨***. concede a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la investigación, con carácter exclusivo. De forma similar se declara como único autor de su contenido. Para que así conste firman la presente a los 2 días del mes de diciembre del año 2022.

<Reynaldo Cesar González Prieto>

_____
Firma del Autor

< Dayana Joseph Smarth >

_____
Firma del Tutor

DATOS DE CONTACTO

<Dayana Joseph Smarth, Ingeniera Informática, lugar de trabajo, lugar de trabajo: centro CREAD de la UCI, responsabilidad actual: especialista B en Ciencias Informáticas, graduada desde el 2006, Análisis de Datos, djoseph@uci.cu >

AGRADECIMIENTOS

A mi mamá Yodania por ser la principal causa de mi existencia, apoyarme constantemente en todos los planes que me propongo. Por la educación brindada en el transcurso de mi vida y por enseñarme a ser quien soy.

A mi papá Reynaldo porque desde pequeño me has apoyado en todo, la voz fuerte que me enseñó que tengo que seguir hacia adelante, no importan los obstáculos y que todo está en el empeño que les ponga a las tareas.

A mi novia Marian la cual creyó en mi en todo momento y siempre me ha apoyado en cualquier situación.

A mi abuela Norma por ser mi segunda madre y consentirme en todo, por ser tan importante en mi vida, por su apoyo incondicional en todo lo que necesité, por todo gracias.

A mi hermanita Aichel por llegar a mi vida cuando más necesitaba esa chispita que me animara con sus travesuras, sus besos, sus abrazos y su amor.

A toda mi familia por apoyarme y confiar en mí, por su preocupación durante mis años de estudio y su constante trabajo de darme ánimos para continuar.

A mis amigos Yainier, David y Fernando por compartir conmigo durante estos años, su confianza en todo momento y su apoyo incondicional.

A mis demás amigos que compartieron y se relacionaron conmigo a lo largo de estos años y a todos los que aportaron su granito de arena para que hoy yo sea la persona que soy.

A mi tutora Dayana que siempre se preocupó por mi tesis, me aconsejó y me ayudó en todo lo que estuvo a su alcance y a todos los profesores que participaron en mi formación como ingeniero.

Al tribunal y oponente por su valioso aporte con el cual contribuyeron a mejorar este trabajo.

DEDICATORIA

Dedico este trabajo de diploma a mis padres, mis abuelas y a mi hermanita, por confiar en mi todo el tiempo y apoyarme en todo; a toda mi familia que de una forma u otra siempre me brindaron su apoyo incondicional.

RESUMEN

Un centro de salud tiene como objetivo desarrollar un sistema informático para la aplicación de una escala médica que permita evaluar el riesgo de complicaciones y mortalidad de los pacientes con COVID 19, y ayudar a un control estricto de los casos de mayor riesgo. La presente investigación tiene como propósito desarrollar sistema informático para evaluar el riesgo complicaciones respiratorias o muerte a pacientes con COVID 19, de forma para que se contribuya al sistema de salud cubano, permita una mejor evaluación de pacientes con problemas respiratorios y ayude a un control más estricto de los casos de mayor complicación. El desarrollo de la solución está guiado por el uso de la metodología de desarrollo de software Proceso Unificado Ágil en su variación para la Universidad de Ciencias Informáticas, se usó Visual Paradigm v8.0 como herramienta para el modelado, PHP v7.4 como lenguaje de programación, NetBeans v8.0 como entorno de desarrollo, MySQL v5.0 como gestor de base de datos y como framework de desarrollo Symfony v5.2. El resultado de la investigación fue validado aplicando pruebas de software.

PALABRAS CLAVE: COVID 19, Centro de salud, sistema, Sistema de informático, riesgo de complicaciones.

INDICE

OPINIÓN DEL(OS) TUTOR(ES)

<Contenido de la opinión de los tutores>

INTRODUCCIÓN

El 31 de diciembre de 2019, la Comisión Municipal de Salud y Sanidad de Wuhan (provincia de Hubei, China) informó a la Organización Mundial de la Salud (OMS) sobre un grupo de 27 casos de neumonía de etiología desconocida**.** El 11 de marzo del 2020 fue declarado pandemia por la OMS, y ya para el 30 de junio 2021 se habían reportado más de 182 millones de casos confirmados por laboratorio, más de 4 millones de muertes asociadas a la enfermedad, en más de 220 países y territorios afectados.

La morbilidad y mortalidad por Covid-19 supone un desafío para la asistencia médica y los sistemas de salud. La mortalidad, si bien se había identificado desde sus inicios que era menor a la del Síndrome Respiratorio Agudo Grave (SARS) y el Síndrome Respiratorio de Oriente Medio (MERS), fue alta desde sus comienzos, Michelozzi y col. reportaron un exceso de mortalidad en las provincias del norte de Italia, que llegó a ser de la mitad de todo el exceso de mortalidad reportado desde principios del 2020, con una tasa de letalidad de 20.2%. Según la red nacional para la vigilancia epidemiológica en España RENAVE la tasa de mortalidad de España 2020 fue de 1.5 por cada 100 000 habitantes, pero para la población de 70-79 años fue de 5.0 y 15.2 para los mayores de 80 años.

América, poco a poco se convirtió en el epicentro de la pandemia y de la mortalidad por esta causa, según La Comisión Económica para América Latina y El Caribe CEPAL hasta el 31 de octubre 2020 los países con mayor número de defunciones del área cuya causa se clasifica como COVID-19 son el Perú, el Brasil, con rango que oscila entre 104,2 y 36,1 por 100.000 habitantes.

En Cuba, el 11 de marzo de 2020 se confirma el primer caso de coronavirus, y 7 días después; se anunciaba el primer fallecido por COVID-19 en el país, que, aunque mostró una baja tasa de mortalidad en sus inicios, la introducción de nuevas cepas más difusibles y agresivas la incrementó no solo en los pacientes ingresados sino también en los egresados por complicaciones.

La variante Delta, identificada por el Instituto de Medicina Tropical “Pedro Kourí” (IPK) en el mes de abril del 2021, se convirtió semanas después, en la variante predominante en el país,

y fue la responsable, entre otros factores, de la elevada incidencia; fueron meses críticos para el sistema de salud cubano, se registraron personas enfermas en todos los municipios del país, con trasmisión comunitaria e incrementos exponenciales del número de activos.
Al existir esta crisis sanitaria en el país a causa del virus mortal, los hospitales colapsan de tantos pacientes que diariamente contraen el virus, se hace más difícil, el proceso de evaluación de los pacientes y hay menos personal capacitado correctamente para la decisión de ingreso o egreso según la gravedad de estos, se hace aún más difícil tomar los datos de los pacientes que se ingresan y su evolución clínica. Todo esto representa un problema en diferentes centros de atención de salud del país.

Por lo anteriormente planteado, se define como **problema a resolver** ¿Cómo contribuir con la evaluación del riesgo de complicaciones y mortalidad al alta, de los pacientes con covid-19 y ayudar a un control estricto de los casos de mayor riesgo?
Dicho problema está enmarcado en el **objeto de estudio:** Aplicaciones web para la gestión de información. Con el **campo de acción** definido como: Aplicaciones web para la información de gestión casos con complicaciones al alta de COVID 19 el cual tiene como **objetivo general:** Desarrollar un sistema informático para la aplicación de una escala médica que permita evaluar el riesgo de complicaciones y mortalidad de los pacientes con COVID 19, y ayudar a un control estricto de los casos de mayor riesgo.
Objetivos específicos y tareas para el cumplimiento de los objetivos.
Elaborar el marco teórico de la investigación mediante un estudio de los referentes teóricos sobre el desarrollo de aplicaciones web para la gestión de casos de COVID 19.
Realizar la identificación de los requisitos, análisis, diseño e implementación del sistema propuesto.
Validar y verificar el correcto funcionamiento del componente propuesto aplicando métricas, criterios y pruebas de software.

Para dar cumplimiento al objetivo trazado anteriormente se plantean las siguientes **tareas de investigación**:

- Definición del marco teórico asociado a la evaluación de riesgo por complicaciones respiratorias a pacientes con COVID 19.

- Análisis de los sistemas existentes para medir la evaluación del riesgo de complicaciones respiratorias en pacientes con COVID 19.
- Selección de metodologías, tecnologías, lenguajes y herramientas para el desarrollo del sistema.
- Análisis y diseño de la propuesta de solución.
- Implementación de la propuesta solución.
- Diseño de pruebas de software y de validación de la solución implementada.

Para el desarrollo de este trabajo se utilizaron los siguientes **métodos de investigación:**

Métodos teóricos:

- **Análisis Histórico-Lógico:** se utilizó para el análisis de las soluciones existentes a nivel internacional y nacional, en el campo de la medicina, que manejen riesgo de complicaciones y mortalidad al alta, de los pacientes ingresados con covid-19.
- **Modelación:** utilizado para la modelación de la aplicación móvil Android.
- **Analítico sintético:** se utilizó para analizar la estructura general de las soluciones encontradas, logrando el entendimiento de cada una de las partes funcionando como un todo.

Métodos empíricos:

- **Encuesta:** se aplicó a profesionales de la salud, para obtener información necesaria para la creación de la aplicación móvil Android, para los diferentes criterios epidemiológicos, criterios clínicos, hallazgos positivos o no al examen físico, criterios radiológicos, criterios de riesgos de trombosis, los antecedentes de trombofilia o trombosis entre otros parámetros médicos.

Estructura de los capítulos:

CAPITULO 1. Fundamentos y Referentes Teóricos-Metodológicos sobre el objeto de estudio.

Se hace referencia a conceptos asociados al proceso de ingreso de un paciente con COVID-19, centrándose en el riesgo de complicaciones y muerte al definir el alta. Se definen las características a analizar en las soluciones halladas a nivel internacional y nacional. Según las características definidas y las soluciones encontradas, se hace un análisis de las mismas para hallar una posible solución al problema planteado o para derivar posibles funcionalida-

des para la propuesta de solución. También se definen y describen las tecnologías, herramientas y lenguaje a utilizar en el desarrollo de la propuesta de solución.

CAPÍTULO 2: DISEÑO DE LA SOLUCIÓN PROPUESTA AL PROBLEMA CIENTÍFICO
Está enfocado al diseño de la solución del problema propuesto. En el mismo se describe el negocio y los requisitos funcionales y no funcionales. Se diseñan las historias de usuario necesarias para la solución de la problemática y la arquitectura de software a utilizar. A partir de los requisitos identificados se modela el diagrama de clase del diseño el cual permite obtener una visión más clara de la implementación del sistema y el modelo de base de datos del módulo propuesto. Además, se muestran ejemplos de estándares de codificación presentes en la implementación y los patrones utilizados.

CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN PROPUESTA
En este capítulo se realiza una estrategia de prueba de software que proporciona una guía que describe los pasos que deben realizarse como parte de la prueba, cuando se planean y se llevan a cabo dichos pasos, y cuánto esfuerzo, tiempo y recursos se requieren. Por tanto, cualquier estrategia de prueba debe incorporar la planificación de la prueba, el diseño de casos de prueba, la ejecución de la prueba y la recolección y evaluación de los resultados. Una estrategia de prueba de software debe ser lo suficientemente flexible para promover un uso personalizado de la prueba. Al mismo tiempo, debe ser suficientemente rígida para alentar la planificación razonable y el seguimiento de la gestión conforme avanza el proyecto. (Pressman, 2010)

CAPITULO 1. FUNDAMENTOS Y REFERENTES TEÓRICOS-METODOLÓGICOS SOBRE EL OBJETO DE ESTUDIO

En el presente capítulo, se hace referencia a conceptos asociados al proceso de ingreso de un paciente con COVID-19, centrándose en el riesgo de complicaciones y muerte al definir el alta. Se definen las características a analizar en las soluciones halladas a nivel internacional y nacional. Según las características definidas y las soluciones encontradas, se hace un análisis de las mismas para hallar una posible solución al problema planteado o para derivar posibles funcionalidades para la propuesta de solución. También se definen y describen las tecnologías, herramientas y lenguaje a utilizar en el desarrollo de la propuesta de solución.

1.1 Conceptos fundamentales asociados al dominio del problema

Para una mejor comprensión del tema de investigación, es necesario dominar las siguientes definiciones:

Tecnologías de la Información y la Comunicación (TIC): Las Tecnologías de la Información y la Comunicación son un conjunto de recursos, herramientas, dispositivos, programas informáticos, aplicaciones, redes y medios que permiten la recopilación, procesamiento, almacenamiento y transmisión de información tales como: voz, datos, texto, vídeo e imágenes (Ibáñez, 2008).

Usabilidad: La usabilidad es la capacidad de un producto de software para ser entendido, aprendido, usado y atractivo para los usuarios cuando se usa bajo condiciones específicas. La usabilidad web es la facilidad con la que una página web es accesible para los visitantes que ingresan e interactúan con ella. Un sitio web con buena usabilidad es aquel que permite interacciones sencillas, intuitivas, agradables y seguras para los usuarios (ISO-25010).

COVID-19: Enfermedad respiratoria muy contagiosa causada por el virus SARS-CoV-2. Se piensa que este virus se transmite de una persona a otra en las gotitas que se dispersan cuando la persona infectada tose, estornuda o habla. (Carter C, Notter J. Cuidados Intensivos de Enfermería En Pacientes Con Covid-19. Elsevier Health Sciences; 2022.)

Riesgo de Complicaciones o muerte: La enfermedad puede cursar con manifestaciones clínicas graves, aumentando la tasa de complicaciones y tornando sombrío el pronóstico de estos individuos. La infección por SARS-CoV-2 por sí sola, no es responsable de la mala evolución de los pacientes; son las complicaciones las determinantes del desenlace. La neumonía grave fue la más frecuente, así también el tromboembolismo pulmonar, la muerte súbita y el paro cardiorrespiratorio. Los APP son de interés ya que en muchas series se han relacionado con la presencia de complicaciones, dígase la Hipertensión arterial, la diabetes mellitus, la enfermedad renal crónica y el asma bronquial son los más frecuentes incluyendo otras comorbilidades, a mayor número de complicaciones mayor probabilidad de fallecer. (Carter C, Notter J. Cuidados Intensivos de Enfermería En Pacientes Con Covid-19. Elsevier Health Sciences; 2022.)

Comorbilidad: Presencia de diferentes enfermedades que acompañan a modo de satélite a una enfermedad protagonista aguda o crónica que es el objeto principal de la atención. También conocida como "morbilidad asociada", es un utilizado para describir dos o más trastornos o enfermedades que ocurren en la misma persona. Pueden ocurrir al mismo tiempo o uno después del otro. La comorbilidad es un fenómeno que complejiza el cumplimiento de las funciones del médico generalista en la atención a pacientes. (Carter C, Notter J. Cuidados Intensivos de Enfermería En Pacientes Con Covid-19. Else-vier Health Sciences; 2022.)

1.2 Tendencias y tecnologías actuales

Con el propósito de obtener experiencia en cuanto a la organización y estructuración para la construcción del sistema propuesto se llevó a cabo un estudio de software de las distintas aplicaciones que existen tanto en el ámbito nacional como extranjero. Entre los softwares extranjeros se encuentran:

COVID Alert NY: Esta es la aplicación oficial del estado de Nueva York dirigida por el Departamento de Salud del Estado de Nueva York como parte de sus esfuerzos en pruebas integrales de COVID-19 y rastreo de contactos. La meta final es ayudar a reducir la propagación de la COVID-19 mediante:

- Alertarlo si una persona enferma pasa 10 minutos o más a menos de 6 pies (2 metros) de usted, debido a que esto lo pone en mayor riesgo de infección por SARS CoV-2, el virus que causa la COVID-19.

- Motivarlo a contribuir con la salud y seguridad de sus amigos, su familia y la comunidad al alertar a los demás si el resultado de su prueba es positivo

- Conseguirle recursos importantes y ayuda si se expone o tiene un resultado positivo en la prueba. Puede llamar a la línea directa de COVID Alert NY o encontrar enlaces útiles para recursos sobre los próximos pasos para proteger a sus seres queridos.

- Lleve un registro privado de sus síntomas, lo cual puede ayudar a su proveedor de atención médica y a representantes de salud pública a determinar los siguientes pasos.

COVID Symptom Study: Tómese 1 minuto diario y ayude a combatir la propagación del COVID-19 en su comunidad

- Informe de su salud a diario, incluso si se siente bien

- Vea los valores diarios de COVID en su zona

- Ayude a frenar el brote cerca de usted

- Qué tan rápido se propaga el virus en su zona

- Las zonas de alto riesgo en EE. UU.

- Quiénes están en mayor riesgo, mediante una mejor comprensión de los síntomas relacionados con afecciones subyacentes

Contribuirá al avance de la investigación sobre el COVID-19 en conjunto con los principales investigadores de salud a nivel mundial, como TwinsUK, uno de los estudios más detallados clínicamente en el mundo.

Esta aplicación está diseñada para todos quienes quieran informar de su estado, no solo los que están enfermos.

Está diseñado por los médicos y científicos de King's College London, Guys Hospital, St Thomas Hospital y Zoe Global Limited, una empresa de tecnología de salud.

En EE. UU., Nurses' Health Study usa la aplicación para identificar los síntomas en trabajadores de salud activos que tratan a personas con COVID en todo el país y arriesgan su salud para ayudarnos.

En respuesta a las recomendaciones de Stand Up To Cancer (SU2C), la aplicación también incluye preguntas para pacientes y sobrevivientes de cáncer, tales como si viven con cáncer, de qué tipo y qué tratamiento reciben.

Solución existente en Cuba

Pesquisador Virtual: solución informática desarrollada por la Universidad de las Ciencias Informáticas (UCI) en colaboración con el Ministerio de Salud Pública (MINSAP), que mediante encuestas permite captar información del estado de salud (síntomas y padecimientos) de la población, como complemento al proceso de pesquisa activa, en el marco del enfrentamiento epidemiológico a la pandemia COVID-19.

Para la realización de la encuesta se cuenta con una solución móvil y una solución Web. El resultado de la encuesta (información estadística y nominal) es monitorizada en tiempo real mediante gráficos y estadísticas por las diferentes instancias del MINSAP, tanto a nivel nacional como provincial y municipal, mostrando reportes personalizados según el nivel de acceso en dependencia del nivel del organismo. Con esta información, diferentes instancias del sistema de Salud actúan de manera inmediata, sobre todo la atención primaria de salud (APS), para evitar la propagación de la epidemia y la atención temprana de posibles contagiados.

Cuba COVID: Esta aplicación pretende mantener actualizada a la población acerca del estado de la COVID-19 en Cuba, a la vez que le proporciona información, aclara algunas de las preguntas frecuentes respecto a la enfermedad y facilita y explica las medidas para prevenir el contagio.

Dispone también de un cuestionario que brinda al usuario la posibilidad de conocer su estado de salud respecto a la COVID-19 y le ofrece recomendaciones para actuar en caso de posible contagio. La información recogida por este cuestionario es totalmente personal no se almacena en el teléfono ni es enviada a otros servidores o entidades.

La información sobre la enfermedad, así como las medidas de prevención se obtuvieron de la página oficial de la organización mundial de la salud.

Tabla 1. Tendencias Actuales [Fuente: Elaboración propia]

Sistemas estudiados	Usabilidad	Conectividad	Pago	Base de Datos
COVID Symptom Study	Regular	Necesario	No	No
COVID Alert NY	Regular	Necesario	No	No
Pesquisador Virtual	Buena	Necesario	No	Si
Cuba COVID	Buena	Necesario	No	No

Después del análisis anterior, se llegó a la conclusión que las soluciones internacionales difieren del esquema establecido en Cuba. Todas son aplicaciones de uso gratuito debido a que combaten una enfermedad tan seria como lo es el Covid-19. Todos estos proyectos no tienen o no está desarrollada en su máximo potencial una base de datos para la información y seguimiento. Cabe mencionar que el software ya existente en Cuba necesita todavía de una cantidad de trabajo para llegar a lo aspirado para el proyecto, cabe resaltar que todos han de mejorar las funcionabilidades en común con nuestro proyecto.

1.3 Metodologías de Desarrollo de Software

Una metodología de desarrollo de software consiste en múltiples herramientas, modelos y métodos para asistir en el proceso de desarrollo de software. En ella se definen con precisión los artefactos, roles y actividades involucradas, junto con prácticas y técnicas recomendadas, las guías de adaptación de la metodología al proyecto y las guías para uso de herramientas de apoyo [CITATION Ass19 \l 21514].

Las metodologías para el desarrollo del software imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente. Tiene como principal objetivo aumentar la calidad del software que se produce en todas y cada una de sus fases

de desarrollo. No existe una metodología de software universal, ya que toda metodología debe ser adaptada a las características de cada proyecto, exigiéndose así que el proceso sea configurable. Las metodologías de desarrollo se clasifican en dos clases: las metodologías tradicionales o robustas, centradas en el control del proceso, con un riguroso seguimiento de las actividades involucradas en ellas, y las ágiles o ligeras, centradas en el factor humano, en la colaboración y participación del cliente en el proceso de desarrollo y a un incesante incremento de software con iteraciones muy cortas [CITATION Pre10 \l 21514].

Metodología de desarrollo de software a utilizar

La metodología de desarrollo de software a utilizar en la presente investigación es la variación de AUP para la UCI (AUP-UCI) en su escenario 4 debido a que logra estandarizar el proceso de desarrollo de software.

Esta es la metodología a utilizar en la línea de productos elaborados en la Universidad de las Ciencias Informáticas.

De las 4 fases que propone AUP (Inicio, Elaboración, Construcción, Transición), AUP-UCI mantiene la fase de Inicio, pero modifica el objetivo de la misma, se unifican las restantes 3 fases de AUP en una sola, la fase de Ejecución y se agrega la fase de Cierre de la siguiente manera:

Inicio: Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización del cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.

Ejecución: En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.

Cierre: En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto. (UCI, 2018)

1.4 Herramientas y tecnologías utilizadas en la propuesta de solución

En la realización de un proyecto es imprescindible la etapa donde se definen las tecnologías y herramientas a utilizar, así como la versión de cada una de ellas que será empleada. Las tecnologías que se emplean para desarrollar la propuesta de solución se seleccionan por las ventajas aprovechables para el desarrollo del sistema de gestión de licencia del IACC.

1.4.1 Herramientas de Ingeniería del Software asistida por computadoras

Las herramientas de Ingeniería de Software Asistida por Computadoras (*CASE por sus siglas en inglés: Computer Aided Software Engineering*) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas ayudan al proceso de desarrollo del software en tareas como: realizar un diseño del proyecto, cálculo de costos, implementación de parte del código con el diseño dado, compilación automática, documentación o detección de errores entre otras[CITATION uni18 \l 23562].

Visual Paradigm v8.0

Constituye una herramienta CASE profesional, que utiliza UML (Unified Modeling Language) como lenguaje de modelado y que soporta el ciclo de vida completo del desarrollo de software, además de tener la ventaja de ser multiplataforma. A continuación, se listan algunas características de la misma [CITATION Vis17 \l 3082]:

- Ofrece amplias características de modelado de casos de uso incluyendo la función completa de UML, diagrama de casos de uso y editor de flujo de eventos.
- Permite diseñar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. El mismo proporciona abundantes tutoriales del Lenguaje de Modelado, demostraciones interactivas de UML y proyectos UML.
- Se integra con herramientas Java, como son: Eclipse/IBM, NetBeans DE, entre otras.
- Es apoyado por un conjunto de lenguajes tanto en la generación del código como en la Ingeniería Inversa.

Con lo expuesto anteriormente y teniendo en cuenta que la UCI posee la licencia académica para el uso de Visual Paradigm v 8.0, se considera la herramienta idónea para el desarrollo del proyecto. La herramienta posee un repertorio de funcionalidades muy completas para el desarrollo de software, yendo desde el análisis y diseño hasta la generación de código fuente y documentación.

1.4.2 Lenguaje de modelado UML v2.1

El UML fue creado para forjar un lenguaje de modelado visual común, semántica y sintácticamente rico para la arquitectura, el diseño y la implementación de sistemas de software complejos, tanto en estructura como en comportamiento. Consiste en diferentes tipos de diagramas que describen los límites, la estructura y el comportamiento del sistema y los objetos que contiene. El UML se perfecciona continuamente de forma tal que la facilidad de uso, la implementación y la adaptación se pueden realizar de manera sencilla. Algunas de las actualizaciones de los diagramas UML son: mayor integración entre modelos estructurales y de comportamiento, capacidad de definir jerarquía y desglosar un sistema de software en componentes y subcomponentes y UML 2.0 eleva el número de diagramas de 9 a 13[CITATION Luc18 \l 23562].

El análisis de este lenguaje se realiza debido a que UML es la notación utilizada por la herramienta CASE que se emplea para la creación de los componentes. Además, es el lenguaje estándar de modelado para software que emplea la metodología AUP. Teniendo en cuenta lo anterior, se decide utilizar el lenguaje de programación antes descrito.

1.4.3 Lenguajes de programación

Constituyen lenguajes artificiales que pueden ser usados para controlar el comportamiento de una máquina, especialmente una computadora. Estos se componen de un conjunto de reglas sintácticas y semánticas que permiten expresar instrucciones que luego serán interpretadas[CITATION ORA18 \l 3082].

Symfony 5.2

Symfony es un framework cuyo principal objetivo es simplificar el desarrollo de aplicaciones en PHP mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Dado la normalización y estructuración que introduce el framework en el de-

sarrollo, una de las ventajas que rápidamente se observan es un aumento en código más legible y más fácil de mantener. Por último, un framework facilita la programación de aplicaciones, ya que encapsula operaciones complejas en instrucciones sencillas.

Con este framework, es posible optimizar el desarrollo de aplicaciones web. Permite la separación de la lógica de negocio y la lógica del servidor, así como la capa de presentación de la aplicación. Symfony aporta diversas herramientas y clases con el objetivo de reducir el tiempo de desarrollo de una aplicación web compleja. Proporciona automatismos para las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación.

Symfony está desarrollado completamente con PHP 5. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Symfony es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y SQL Server de Microsoft. Se puede ejecutar tanto en plataformas nix (Unix, Linux, etc.) como en plataformas Windows.

PHP v7.4

PHP (acrónimo de *Hypertext Preprocessor)* es un lenguaje del lado del servidor (esto significa que PHP funciona en un servidor remoto que procesa la página web antes de que sea abierta por el navegador del usuario) especialmente creado para el desarrollo de páginas web dinámicas. Este puede ser incluido con facilidad dentro del código HTML (por la traducción de sus siglas al inglés *HyperText Markup Language*), y permite una serie de funcionalidades extraordinarias [CITATION Ion20 \l 21514].

Tres funciones principales distinguen en particular el PHP:

- Orientado al desarrollo de aplicaciones web dinámicas con acceso a información almacenada en una base de datos.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Es un lenguaje de secuencia de comandos de servidor. Dentro de una página web puede incluir código PHP que se ejecute cada vez que se visite una página. El código
- PHP es interpretado en el servidor web y genera código HTML y otro contenido que el visitante verá[CITATION ict19 \l 3082].

JavaScript v1.8

El JavaScript es un lenguaje de programación interpretado, lo que significa que no necesita ser compilado. Orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Proviene del Java y se utiliza principalmente para la creación de páginas web. El JavaScript es una mezcla entre el Java y el HTML, es un lenguaje que se incorpora dentro de la página web, formando parte del código HTML[CITATION Les201 \l 1033].

HTML v.5

Por sus siglas en inglés de (*HyperText Markup Language)* es el lenguaje básico de la web para crear documentos y aplicaciones para que los desarrolladores lo utilicen en cualquier lugar. HTML en su versión 5 establece una serie de nuevos elementos y atributos que reflejan el uso típico de los sitios web modernos. Es un lenguaje simple que sirve para definir otros lenguajes que tienen que ver con el formato de los documentos [CITATION w3s18 \l 2058].

CSS v.3

CSS (*Cascading Style Sheets*) es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML).

Está diseñado principalmente para marcar la separación del contenido del documento y la forma de presentación de este. Con esto se pretende mejorar la accesibilidad del documento, permitir que varios documentos HTML compartan un mismo estilo usando una sola hoja de estilos separada en un archivo .css, y reducir la complejidad y la repetición de código en la estructura del documento[CITATION w3s18 \l 2058].

1.4.4 Entorno de Desarrollo Integrado

Es una aplicación informática que proporciona servicios integrales para facilitarle al programador el desarrollo de software. Normalmente, un entorno de desarrollo integrado (IDE) consta de un editor de código fuente, herramientas de construcción automáticas y un depurador. La mayoría de estos tienen autocompletado inteligente de código (IntelliSense). Algunos IDE contienen un compilador, un intérprete, o ambos, tales como NetBeans y Eclipse[CITATION Jua20 \l 3082].

NetBeans v8.0

Es un entorno de desarrollo integrado libre, que permite programar en diversos lenguajes. Existe además un número importante de módulos para extenderlo. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.

1.4.5 Sistema Gestor de Bases de Datos

Un Sistema Gestor de Base de Datos (SGBD) es un sistema que permite la creación, gestión y administración de bases de datos, así como la elección y manejo de las estructuras necesarias para el almacenamiento y búsqueda de información del modo más eficiente posible[CITATION Raf19 \l 1033].

MySQL v5.0

Es el más común en la actualidad al estar basado en código abierto. Se trata de un sistema de gestión relacional, es decir, utiliza tablas múltiples que se conectan entre sí para organizar y almacenar la información de manera correcta. Tiene una arquitectura cliente/servidor, compatibilidad con SQL, soporte multiplataforma y ofrece la posibilidad de incrementar la eficacia de la implementación sin necesidad de procesar las tablas directamente. Se ha enfocado tradicionalmente en aplicaciones web de lectura, usualmente escritas en PHP.

Para la realización de una aplicación web se necesita una base de datos rápida, que permita abundantes consultas, con integridad y protección de los datos. Por cumplir con estos requisitos y además es muy fácil de usar, utiliza varias capas de seguridad entre otras ventajas, se selecciona el SGBD MySQL, ya que sus características apoyan al cumplimiento de los objetivos de este proyecto

Conclusiones del capítulo

En este capítulo se marcaron las pautas teóricos-metodológicas del objeto de estudio, lo cual ayuda a facilitar la comprensión del mismo. Para la propuesta de solución se determina el uso de AUP-UCI como metodología de desarrollo, Visual Paradigm v8.0 como herramienta para el modelado, PHP v7.4 como lenguaje de programación, NetBeans v8.0 como entorno de desarrollo, MySQL v5.0 como gestor de base de datos y como framework de desarrollo Symfony v5.2.

CAPÍTULO 2: DISEÑO DE LA SOLUCIÓN PROPUESTA AL PROBLEMA CIENTÍFICO

2.1 Introducción

Este capítulo está enfocado al diseño de la solución del problema propuesto. En el mismo se describe el negocio y los requisitos funcionales y no funcionales. Se diseñan las historias de usuario necesarias para la solución de la problemática y la arquitectura de software a utilizar. A partir de los requisitos identificados se modela el diagrama de clase del diseño el cual permite obtener una visión más clara de la implementación del sistema y el modelo de base de datos del módulo propuesto. Además, se muestran ejemplos de estándares de codificación presentes en la implementación y los patrones utilizados.

2.2 Descripción del negocio

Al llegar el paciente al área de salud correspondiente es atendido por el médico de urgencia, el cual registra todos sus datos, exámenes físicos y complementarios en su historia clínica y los añade a la hoja de cargo del especialista en turno, la cual se registra en estadística del centro una vez cumplido su horario laboral.

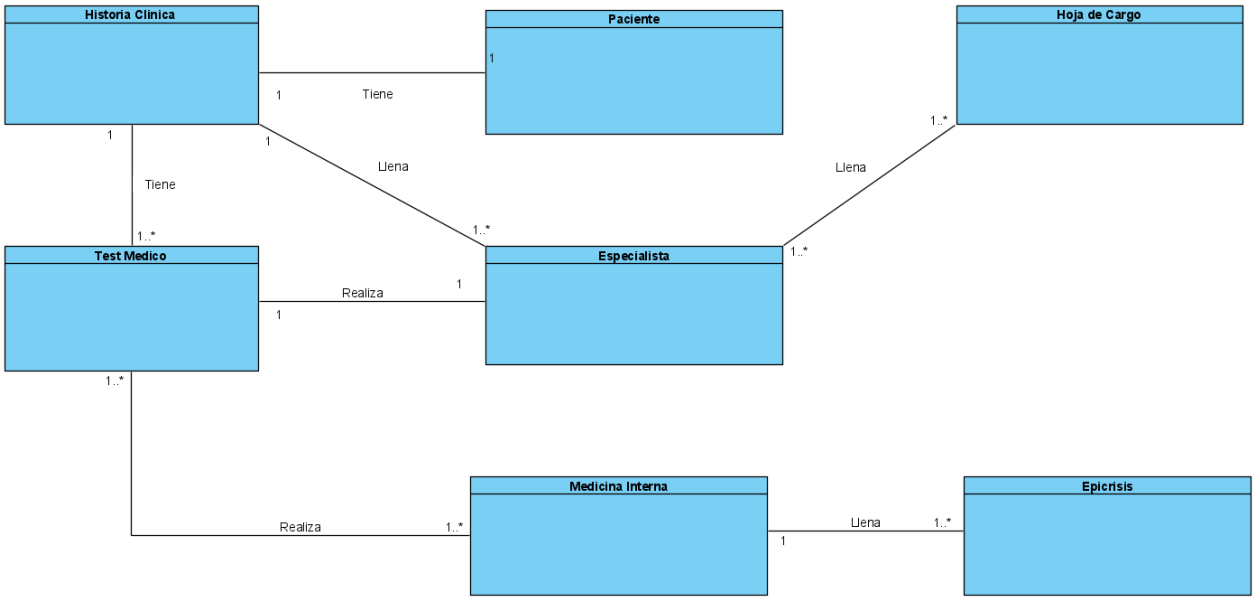
Según la severidad de la afectación en el paciente puede darse el alta y asignación de atención primaria o ingreso domiciliario, de ser un riesgo de muerte alto se decide el ingreso en el área de atención médica secundaria o sala de respiratorio.

A los pacientes ingresados se les realiza un diagnóstico de exámenes físicos y complementarios para valorar la evolución de la complicación en estos y así valorar el alta médica o mantener el seguimiento hasta su recuperación.

Si el paciente ingresado recibe el alta médica se crea el documento de epicrisis donde queda registrado el resumen de la evaluación del paciente desde el ingreso, dicho documento se registra en la base de datos de estadísticas del centro.

2.2.1 Diseño del Modelo de Dominio

El modelo del dominio es una representación de las clases conceptuales del mundo real que se utiliza como guía para el diseño de los objetos. En él se muestran las clases conceptuales más significativas del dominio del problema. Este se realiza cuando no se logra determinar el proceso del negocio con fronteras bien establecidas. Este modelo representa clases conceptuales del dominio del problema que son las ideas u objetos físicos y el enlace de unos objetos con otros. Para la realización del mismo se utiliza un lenguaje común facilitando la comunicación entre los desarrolladores y usuarios de la aplicación y un mayor entendimiento del contexto en que se desarrolla el sistema. El modelo de dominio se describe mediante diagramas UML (diagrama de clases). Estos diagramas muestran las clases del dominio y cómo se relacionan unas con otras mediante asociaciones (Larman, 2003).



2.3 Requisitos

Un requisito es la condición o capacidad que un usuario necesita para poder resolver un problema o lograr un objetivo. También se define como la condición o capacidad que debe exhibir o poseer un sistema para satisfacer un contrato, estándar, especificación, u otra documentación formalmente impuesta [CITATION IEE18 \l 2058].

2.3.1 Definición de requisitos funcionales

Los requisitos funcionales (RF) de un sistema, son aquellos que describen cualquier actividad que este deba realizar, en otras palabras, el comportamiento o función particular de un sistema cuando se cumplen ciertas condiciones. Por lo general, estos deben incluir funciones desempeñadas por pantallas específicas, descripciones de los flujos de trabajo a ser desempeñados por el sistema y otros requerimientos de negocio[CITATION Rog071 \l 2058].

A continuación, son definidos los requisitos funcionales del sistema a desarrollar:

Tabla 2.1 Requisitos funcionales del sistema informático para evaluar riesgo y complicaciones respiratorias a pacientes con COVID 19 [Fuente: Elaboración Propia]

No.	Nombre	Descripción
RF1	Listar usuarios	El sistema permite que los usuarios que tien en rol de administrador puedan listar los da tos de usuarios.
RF2	Insertar usuario	El sistema permite que los usuarios que tien en rol de administrador puedan insertar usu ario.
RF3	Modificar usuario	El sistema permite que los usuarios que tien en rol de administrador puedan modificar u suario.
RF4	Eliminar usuario	
RF5	Autenticar usuario.	El sistema permite la autenticación de usuarios.
RF6	Cerrar sesión	El sistema permite cerrar la sesión de un usuario que se encuentre registrado.
RF7	Cambiar contraseña	El sistema permite cambiar la contraseña de un usuario.

RF8	Listar trazas	El sistema permite que los usuarios que tien en rol de administrador puedan listar los da tos de las trazas.
RF9	Visualizar trazas por fecha	El sistema permite que los usuarios que tien en rol de administrador puedan visualizar lo s datos de trazas por fecha.
RF10	Visualizar trazas por usuario	El sistema permite que los usuarios que tien en rol de administrador puedan visualizar lo s datos de trazas por usuario.
RF11	Exportar trazas a .pdf	El sistema permite que los usuarios que tien en rol de administrador puedan exportar las trazas a Pdf.
RF12	Salvar BD	El sistema permite que los usuarios que tien en rol de administrador puedan hacer una salva de la BD.
RF13	Restaurar BD	El sistema permite que los usuarios que tien en rol de administrador puedan hacer una restauración de la BD.
RF14	Listar criterios de edad	El sistema permite que los usuarios que tien en rol de administrador puedan listar los da tos de criterios de edad.
RF15	Insertar criterio edad	El sistema permite que los usuarios que tien en rol de administrador puedan insertar criterio edad.
RF16	Modificar criterio edad	El sistema permite que los usuarios que tien en rol de administrador puedan modificar criterio edad.

RF17	Eliminar criterio edad	El sistema permite que los usuarios que tien en rol de administrador puedan eliminar criterio edad.
RF18	Listar índice de neutrófilos	El sistema permite que los usuarios que tien en rol de administrador puedan listar los da tos de índice de neutrófilos.
RF19	Insertar índice de neutrófilos	El sistema permite que los usuarios que tien en rol de administrador puedan insertar índice de neutrófilos.
RF20	Modificar índice de neutrófilos	El sistema permite que los usuarios que tien en rol de administrador puedan modificar índice de neutrófilos.
RF21	Eliminar índice de neutrófilos	El sistema permite que los usuarios que tien en rol de administrador puedan eliminar índice de neutrófilos.
RF22	Listar comorbilidades	El sistema permite que los usuarios que tien en rol de administrador puedan listar los da tos de comorbilidades.
RF23	Insertar comorbilidad	El sistema permite que los usuarios que tien en rol de administrador puedan insertar co morbilidad.
RF24	Modificar comorbilidad	El sistema permite que los usuarios que tien en rol de administrador puedan modificar c omorbilidad.
RF25	Eliminar comorbilidad	El sistema permite que los usuarios que tien en rol de administrador puedan eliminar co

		morbilidad.
RF26	Listar criterios APP (Antecedentes personales patológicos)	El sistema permite que los usuarios que tien en rol de administrador puedan listar escalas.
RF27	Insertar criterio APP	El sistema permite que los usuarios que tien en rol de administrador puedan insertar criterio de APP.
RF28	Modificar criterio APP	El sistema permite que los usuarios que tien en rol de administrador puedan modificar criterio de APP.
RF29	Eliminar criterio APP	El sistema permite que los usuarios que tien en rol de administrador puedan eliminar criterio de APP.
RF30	Listar radiologías	El sistema permite que los usuarios que tien en rol de administrador puedan listar las radiologías.
RF31	Insertar radiología	El sistema permite que los usuarios que tien en rol de administrador puedan insertar radiología.
RF32	Modificar radiología	El sistema permite que los usuarios que tien en rol de administrador puedan modificar radiología.
RF33	Eliminar radiología	El sistema permite que los usuarios que tien en rol de administrador puedan eliminar radiología.
RF34	Listar necesidades de oxígeno	El sistema permite que los usuarios que tien

		en rol de administrador puedan listar las ne cesidades de oxígeno.
RF35	Insertar necesidad de oxígeno	El sistema permite que los usuarios que tien en rol de administrador puedan insertar nec esidad de oxígeno.
RF36	Modificar necesidad de oxígeno	El sistema permite que los usuarios que tien en rol de administrador puedan modificar n ecesidad de oxígeno.
RF37	Eliminar necesidad de oxígeno	El sistema permite que los usuarios que tien en rol de administrador puedan eliminar ne cesidad de oxígeno.
RF38	Listar exámenes clínicos	El sistema permite que los usuarios que tien en rol de administrador puedan listar los ex ámenes clínicos.
RF39	Insertar examen clínico	El sistema permite que los usuarios que tien en rol de administrador puedan insertar examen clínico.
RF40	Modificar examen clínico	El sistema permite que los usuarios que tien en rol de administrador puedan modificar examen clínico.
RF41	Eliminar examen clínico	El sistema permite que los usuarios que tien en rol de administrador puedan eliminar examen clínico.
RF42	Listar criterios IMC (Índice de Masa Corporal)	El sistema permite que los usuarios que tien en rol de administrador puedan listar los cri terios de IMC.

RF43	Insertar criterio IMC	El sistema permite que los usuarios que tien en rol de administrador puedan insertar criterio IMC.
RF44	Modificar criterio IMC	El sistema permite que los usuarios que tien en rol de administrador puedan modificar criterio IMC.
RF45	Eliminar criterio IMC	El sistema permite que los usuarios que tien en rol de administrador puedan eliminar criterio IMC.
RF46	Listar inicio de síntomas	El sistema permite que los usuarios que tien en rol de administrador puedan listar los ini cios de síntomas.
RF47	Insertar inicio de síntomas	El sistema permite que los usuarios que tien en rol de administrador puedan insertar inicio de síntomas.
RF48	Modificar inicio de síntomas	El sistema permite que los usuarios que tien en rol de administrador puedan modificar inicio de síntomas.
RF49	Eliminar inicio de síntomas	El sistema permite que los usuarios que tien en rol de administrador puedan eliminar inicio de síntomas.
RF50	Listar diagnósticos	El sistema permite que los usuarios que tien en rol de recepcionista puedan listar los dat os de diagnósticos.
RF51	Registrar diagnóstico	El sistema permite que los usuarios que tien en rol de recepcionista puedan registrar dia gnósticos.
RF52	Modificar diagnóstico	El sistema permite que los usuarios que tien

		en rol de recepcionista puedan modificar di agnósticos.
RF53	Eliminar diagnóstico	El sistema permite que los usuarios que tien en rol de recepcionista puedan eliminar dia gnósticos.
RF54	Listar test médicos	El sistema permite que los usuarios que tien en rol de recepcionista puedan listar los dat os de tests médicos.
RF55	Insertar test médico	El sistema permite que los usuarios que tien en rol de recepcionista puedan insertar test médicos.
RF56	Modificar test médico	El sistema permite que los usuarios que tien en rol de recepcionista puedan modificar te st médicos.
RF57	Eliminar test médico	El sistema permite que los usuarios que tien en rol de recepcionista puedan eliminar test médicos.
RF58	Listar pacientes	El sistema permite que los usuarios que tien en rol de recepcionista puedan listar los dat os de pacientes.
RF59	Insertar paciente	El sistema permite que los usuarios que tien en rol de recepcionista puedan insertar paci ente.
RF60	Modificar paciente	El sistema permite que los usuarios que tien en rol de recepcionista puedan modificar pa ciente.

RF61	Eliminar paciente	El sistema permite que los usuarios que tien en rol de recepcionista puedan eliminar pac iente.
RF62	Visualizar pacientes por criterios según medidas	El sistema permite que los usuarios que tien en rol de administrador, o recepcionista pue dan visualizar a los pacientes según los crite rios de medidas que hayan previamente sel eccionado.
RF63	Visualizar pacientes mayores de 50 años que presentan alto ries- go de complicaciones.	El sistema permite que los usuarios que tien en rol de administrador, o recepcionista pue dan visualizar a los pacientes mayores de 50 años que presentan alto riesgo de complicaciones.
RF64	Graficar pacientes mayores de 50 años que presentan alto ries- go de complicaciones.	El sistema permite que los usuarios que tien en rol de administrador, o recepcionista vis ualizar a los pacientes de 50 años que presentan alto riesgo de complicaciones.
RF65	Visualizar pacientes con riesgo de trombosis alto.	El sistema permite que los usuarios que tien en rol de administrador, o recepcionista pue dan visualizar a los pacientes con riesgo de trombosis alto.
RF66	Graficar cantidad de pacientes con riesgo de trombosis alto dado un periodo de fecha.	El sistema permite que los usuarios que tien en rol de administrador recepcionista pued an visualizar gráficamente a los pacientes con riesgo de trombosis alto dado un período de fecha.

RF67	Visualizar diagnósticos por año	El sistema permite que los usuarios que tien en rol de administrador, recepcionista pued an visualizar los diagnósticos por año.
RF68	Visualizar test médicos por año	El sistema permite que los usuarios que tien en rol de administrador, recepcionista pued an visualizar los tests médicos por año.

2.3.2 Requisitos no funcionales

Los requisitos no funcionales son necesarios para establecer aquellos aspectos necesarios a tener en cuenta para el desarrollo del sistema, pero que no intervienen en las funcionalidades del mismo directamente. Para este proyecto en la captura de requisitos se enfatizó en la seguridad. También se tuvo en cuenta la apariencia, las condiciones del hardware, entre otros que se aprecian a continuación:

- ✓ **Apariencia o interfaz externa.**
  - Su interfaz deberá ser amigable y de fácil comprensión para su trabajo.
  - El sistema tendrá una interfaz simple, de acuerdo a la línea de diseño establecida, en el color que representa al órgano presente. La tipografía utilizada debe ser uniforme, de un tamaño adecuado.

- ✓ **Usabilidad.**
  - El sistema podrá ser utilizado por personas que tengan un conocimiento básico en el manejo de las computadoras.
  - Se debe garantizar que la curva de aprendizaje del sistema sea baja. El usuario debe alcanzar su máxima productividad en el menor tiempo posible desde la primera vez de uso. Para ello debe proveerse al usuario de un manual que detalle las funcionalidades del sistema.
  - En el sistema se deben visualizar todos los mensajes en idioma español.

- El sistema debe facilitar la interacción con el usuario. Se mostrarán mensajes al cancelar alguna operación relacionada con el registro o la modificación. Se alertará ante la posible ejecución de operaciones críticas.

✓ **Soporte.**

- El sistema permitirá la realización de salvas de la base de datos.
- El sistema permitirá la restauración de la base de datos.
- Garantizar un sistema con capacidad de mantenimiento, reparable y escalable.

✓ **Seguridad.**

- Cada usuario debe autenticarse para iniciar su trabajo en el sistema.
- Todos los usuarios tienen que estar registrados en el sistema y contener un rol con determinados permisos.
- Cada rol tendrá un nivel de acceso, el sistema será accedido por los usuarios con permisos y accesos requeridos.
- El usuario tendrá la opción de cambiar su contraseña.

2.4 Historias de usuario

Las Historias de Usuario (HU) son una forma rápida de administrar los requisitos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. Las historias de usuario permiten responder rápidamente a los requisitos cambiantes, por lo que una historia de usuario puede tener varios cambios a lo largo de un desarrollo sin afectarse el tiempo [CITATION Mar19 \l 21514].

El sistema informático para evaluar Riesgo complicaciones respiratorias o muerte a pacientes con Covid-19, posee 48 requisitos funcionales como bien se pudo observar anteriormente, los cuales se describen en historias de usuario, ya que estas muestran las especificaciones de los requisitos. A continuación, se exponen cuatro ejemplos de HU para los requisitos funcionales 1, 2, 3 y 4. La decisión de incluir estas cuatro HU en el documento se debe a que los demás requisitos parten de la misma funcionalidad de estos presentados.

Tabla 2.2 Historia de usuario Listar pacientes [Fuente: Elaboración Propia]

Número: 58	**Requisito: Listar pacientes**
Programador: Reynaldo Cesar González	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 2 hora
Riesgo en Desarrollo: N/A	**Tiempo Real:** 1 hora
Descripción: El sistema permite que los usuarios con rol de Recepcionista puedan listar los datos de los pacientes.	
Observaciones: Para listar pacientes tienen que existir registros de pacientes en la base de datos.	
Prototipo de interfaz gráfica de usuario:	

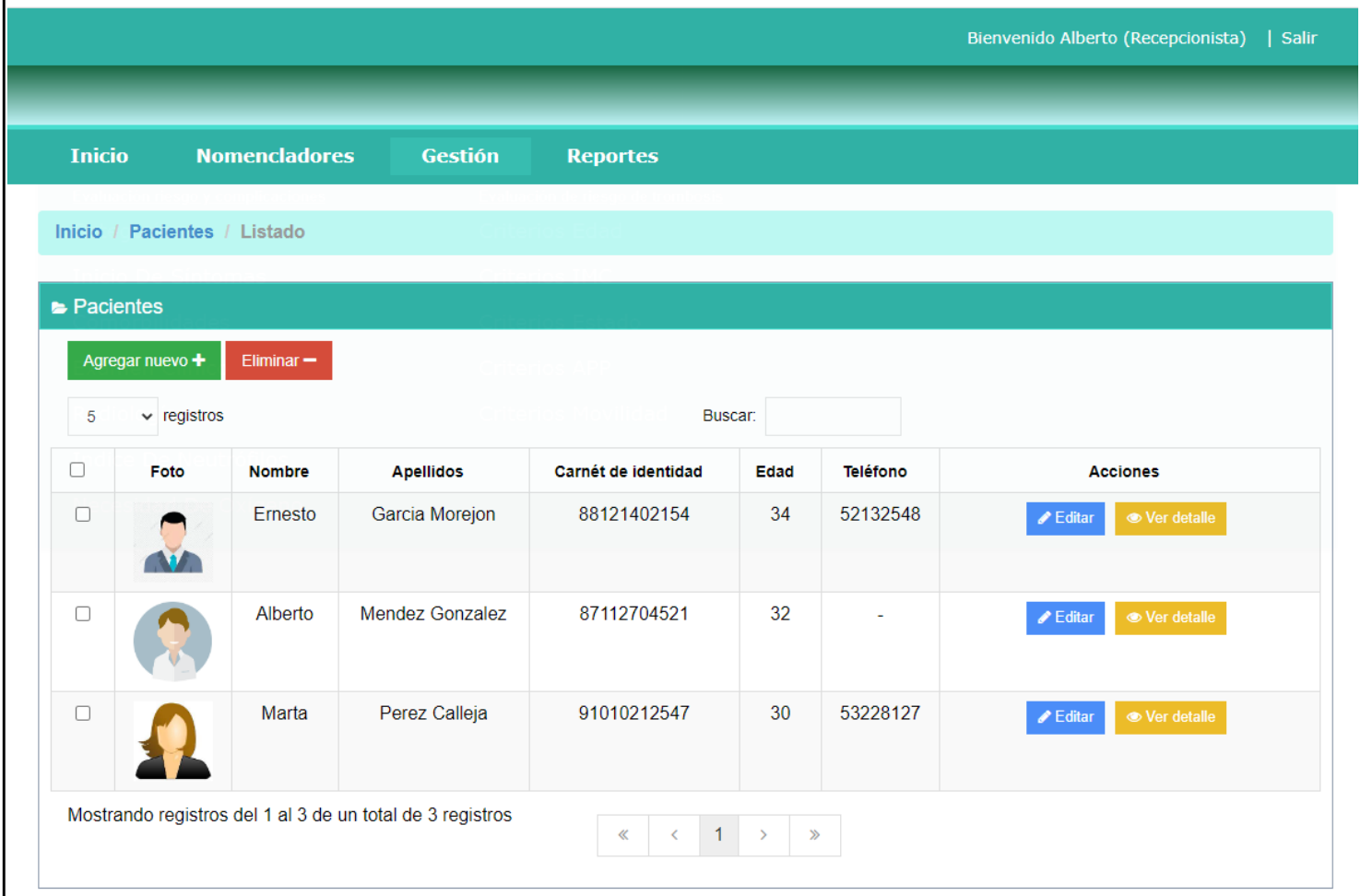


Tabla 2.3 Historia de usuario Insertar paciente [Fuente: Elaboración Propia]

Número: 59	**Requisito: Insertar paciente**
Programador: Reynaldo Cesar González	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 24 horas
Riesgo en Desarrollo: N/A	**Tiempo Real:** 22 horas
Descripción: El sistema permite que los usuarios con rol recepcionista añadan un nuevo paciente mediante el llenado de un formulario con los campos: **Nombre: (Obligatorio. Campo de texto)** **Primer apellido: (Obligatorio. Campo de texto)** **Segundo apellido: (Obligatorio. Campo de texto)** **Foto: (Opcional. Campo de imagen)** **Carnét de identidad: (Obligatorio. Campo de texto)** **Teléfono: (Opcional. Campo de texto)** **Edad: (Obligatorio. Campo de texto)**	
Observaciones: Para insertar un paciente tienen que existir registros de pacientes en la base de datos.	
Prototipo de interfaz gráfica de usuario:	

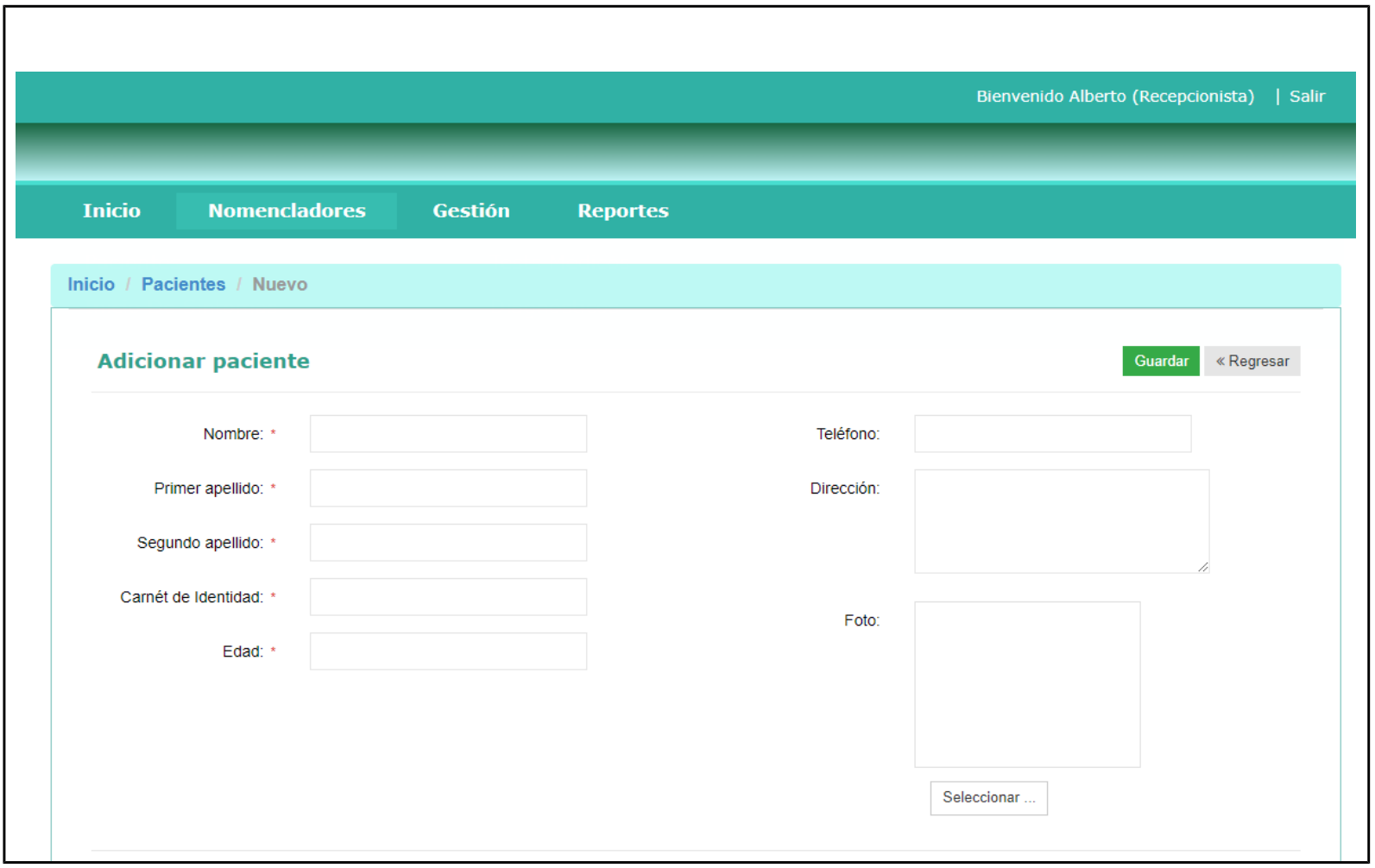


Tabla 2.4 Historia de usuario Modificar paciente [Fuente: Elaboración Propia]

Número: 60	**Requisito: Modificar paciente**
Programador: Reynaldo Cesar González	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 24 horas
Riesgo en Desarrollo: N/A	**Tiempo Real:** 22 horas
Descripción: El sistema permite que los usuarios con rol recepcionista puedan modificar un paciente mediante el llenado de un formulario con los campos: **Nombre: (Obligatorio. Campo de texto)** **Primer apellido: (Obligatorio. Campo de texto)**	

Segundo apellido: (Obligatorio. Campo de texto)

Foto: (Opcional. Campo de imagen)

Carnét de identidad: (Obligatorio. Campo de texto)

Teléfono: (Opcional. Campo de texto)

Edad: (Obligatorio. Campo de texto)

Observaciones: **Para modificar un paciente tienen que existir registros de pacientes en la base de datos.**

Prototipo de interfaz gráfica de usuario:

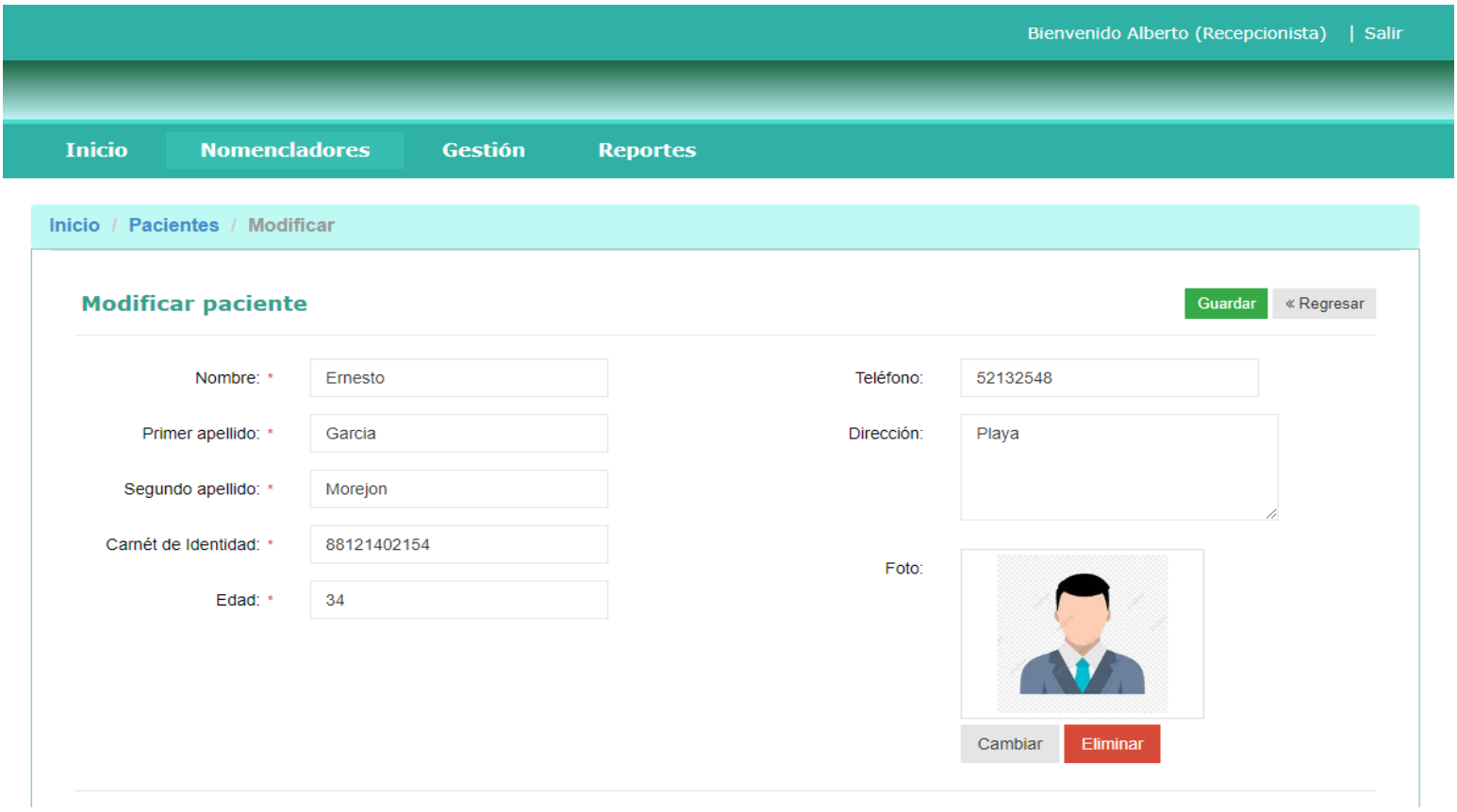
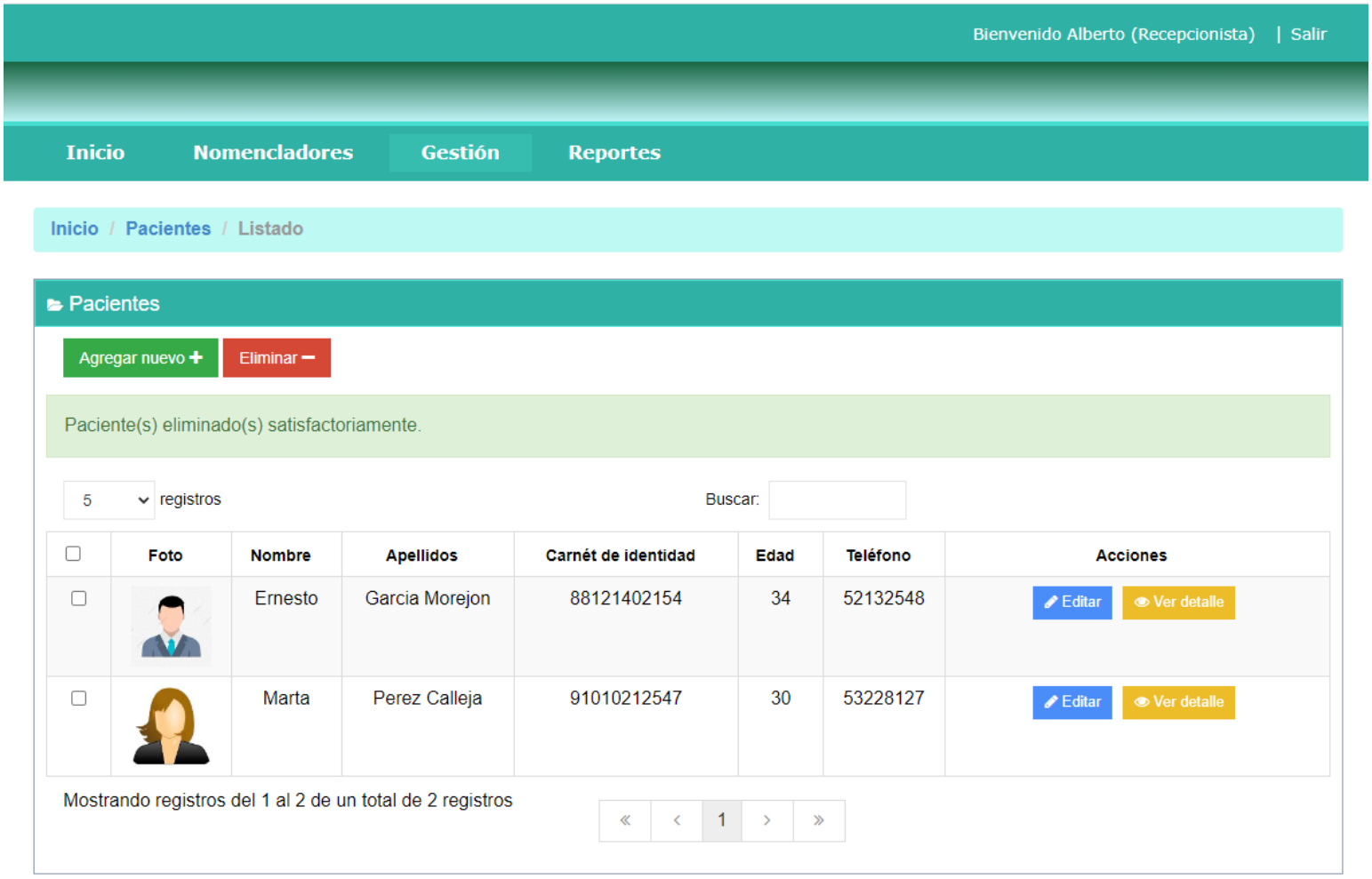


Tabla 2.5 Historia de usuario Eliminar paciente [Fuente: Elaboración Propia]

Número: 61	**Requisito: Eliminar paciente**
Programador: Reynaldo Cesar González	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 24 horas
Riesgo en Desarrollo: N/A	**Tiempo Real:** 22 horas
Descripción: El sistema permite que los usuarios con rol recepcionista puedan eliminar los datos de un paciente mediante una casilla de verificación de tipo checkbox.	
Observaciones: Para eliminar un paciente tienen que existir registros de pacientes en la base de datos.	
Prototipo de interfaz gráfica de usuario:	



2.5 Arquitectura de Software

El diseño arquitectónico garantiza cómo debe organizarse un sistema y cómo tiene que diseñarse la estructura global del mismo. En el modelo del proceso de desarrollo de software, el diseño arquitectónico es la primera etapa en el proceso de diseño del software. Es el enlace crucial entre el diseño y la ingeniería de requerimientos, ya que identifica los principales componentes estructurales en un sistema y la relación entre ellos. La salida del proceso de diseño arquitectónico consiste en un modelo que describe la forma en que se organiza el sistema como un conjunto de componentes en comunicación (Sommerville, 2016).

El patrón arquitectónico seleccionado en la presente investigación para el desarrollo de la propuesta de solución es Modelo-Vista-Controlador (MVC), ya que es el que utiliza el framework de desarrollo Symfony 5.2.10. Entre sus ventajas está la posibilidad de separar la lógica de negocio (el modelo) y la presentación (la vista), por lo que se consigue un mantenimiento más sencillo de las aplicaciones.

Si por ejemplo una misma aplicación debe ejecutarse tanto en un navegador estándar como en un navegador de un dispositivo móvil, solamente es necesario crear una vista nueva para cada dispositivo; manteniendo el controlador y el modelo original. El controlador se encarga de aislar al modelo y a la vista de los detalles del protocolo utilizado para las peticiones. El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes de, por ejemplo, el tipo de gestor de bases de datos utilizado por la aplicación.

Symfony 5.2.10 implementa el patrón arquitectónico MVC como se muestra en la siguiente figura:

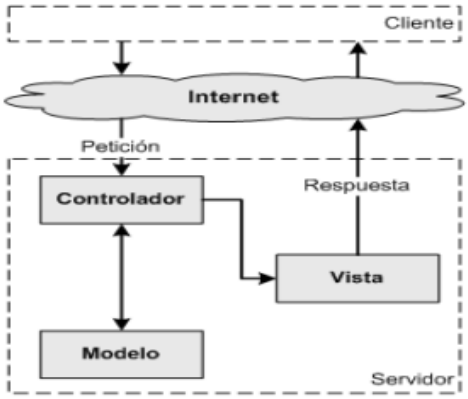


Fig. 2.1 Patrón arquitectónico MVC. (Fuente: (Uniwebsidad, 2018))

Symfony 5.2.10 simplifica el proceso de acceder a cada parte de la aplicación, obtiene lo mejor del patrón arquitectónico MVC y la implementa de forma que el desarrollo de aplicaciones sea rápido y sencillo. En primer lugar, el controlador frontal y el layout son comunes para todas las acciones de la aplicación. Se pueden tener varios controladores y varios layouts, pero solamente es obligatorio tener uno de cada uno. El controlador frontal es un componente que sólo tiene código relativo al MVC, por lo que no es necesario crear uno, ya que Symfony 5.2.10 lo genera de forma automática. (Uniwebsidad, 2018).

2.6 Diagrama de clases de la solución

Un diagrama de clases es una herramienta para comunicar el diseño de un programa que se creó para orientar objetos y que permite modelar relaciones entre diferentes entidades. Los diagramas de clase describen la vista estática del modelo o parte del modelo, describiendo que atributos y comportamientos tienen en lugar de detallar los métodos para realizar operaciones, son más útiles para ilustrar relaciones entre clases e interfaces. Las generalizaciones, agregaciones, y asociaciones son todas valiosas al reflejar herencias, composición o uso, y conexiones respectivamente (Gómez & Olive, 2003). En el Anexo 1 se muestra el diagrama de clases general para la solución propuesta.

A continuación, en la figura 2.2 se muestra un diagrama de clases de diseño con estereotipos web para la historia de usuario: Insertar paciente.

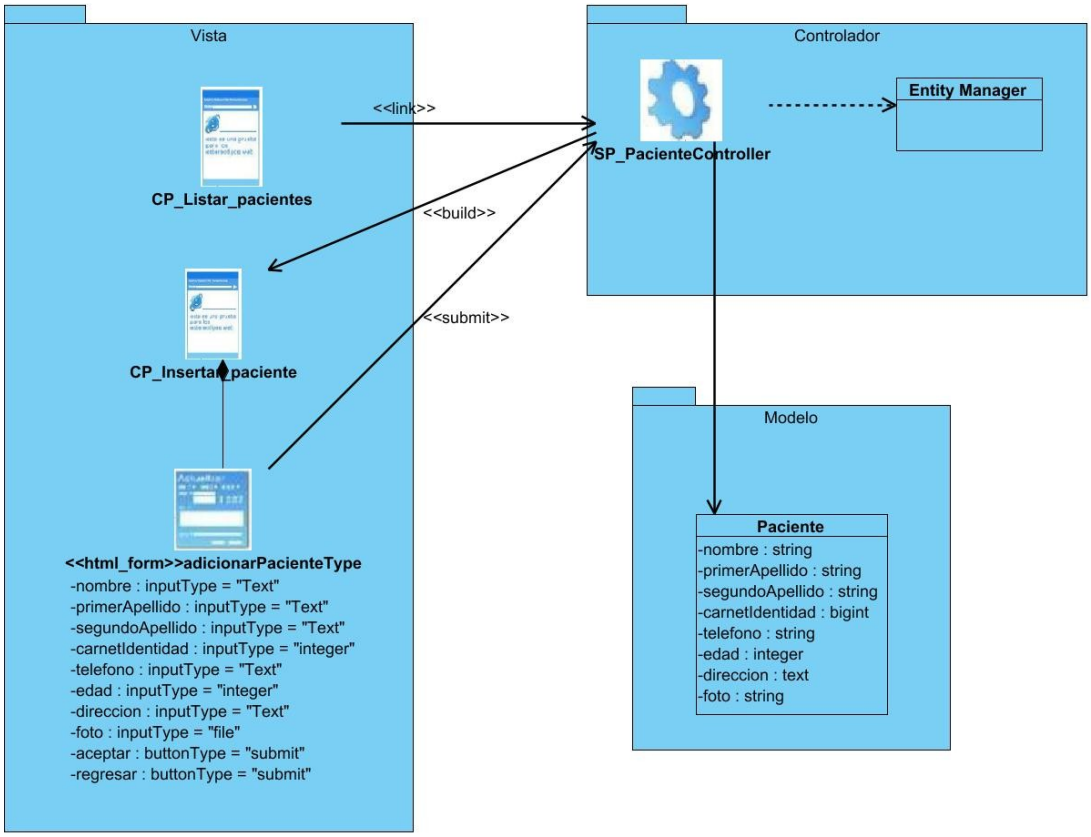


Fig. 2.2 Diagrama de clases del diseño de la HU Insertar paciente [Fuente: Elaboración propia]

2.7 Diagrama de componentes

El diagrama de componentes muestra las relaciones estructurales entre los componentes de un sistema. Los componentes se consideran unidades autónomas encapsuladas dentro de un sistema o subsistema que proporcionan una o más interfaces. Estos diagramas son generalmente dirigidos al personal de aplicación de un sistema, además presenta una comprensión temprana del sistema global que se está construyendo. (IBM, 2004)

En la Figura 2.2.3 a continuación se muestra el diagrama de componentes de la solución.

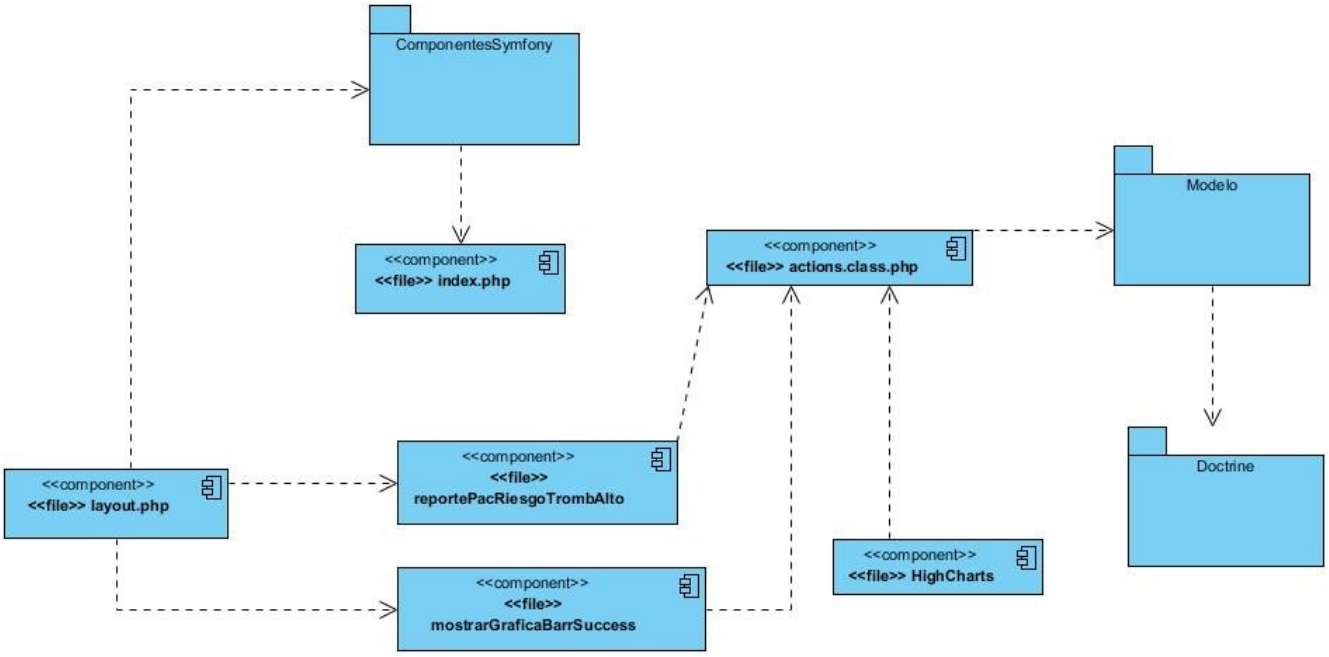


Fig.2.2.3 Diagrama de componentes. Generar reporte de paciente de riesgo Alto. [Fuente: Elaboración propia]

A continuación, se expone la descripción de los componentes y subsistemas de implementación que conforman el diagrama de componente.

index.php: Corresponde con el controlador frontal del sistema que es el único punto de entrada a la aplicación.

layout.php: Contiene los elementos que se muestran de forma idéntica a lo largo de toda la aplicación.

actions.class.php: Corresponde a la clase que almacena todas las acciones del módulo.

reportePacRiesgoTrombAlto: Se encarga de visualizar el Reporte por tipo de Riesgo de trombosis.

mostrarGraficaBarraSuccess.php: Es el encargado de mostrar el reporte seleccionado por el usuario en forma de barra.

Highcharts: El componente Highcharts es una biblioteca que incluye gráficos para aplicaciones web como: gráficos de barra, de pastel, entre otros.

2.8 Patrones de diseño

Los patrones de diseño son técnicas para resolver problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño resulta ser una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias. También son propuestas generales planteadas en término de una colaboración de objetos mediante las que se resuelven una gran cantidad de problemas que aparecen una y otra vez en el desarrollo de aplicaciones informáticas. Se consideran una serie de buenas prácticas de aplicación recomendable en el diseño de software. (Iturralde, 2016).

2.8.1 Patrones GRAPS

Los patrones GRASP son una ayuda en el aprendizaje que permiten al desarrollador entender lo esencial del diseño de objetos y aplicar el razonamiento de una forma metódica, racional y explicable [CITATION Rog071 \l 21514]. A continuación, se describen los utilizados en la propuesta de solución.

Patrón controlador: El patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control. Se recomienda dividir los eventos del sistema en el mayor número de controladores para poder aumentar la cohesión y disminuir el acoplamiento. (Iturralde, 2016).

En la figura 2.2 se muestra el uso de este patrón controlador en la solución propuesta.

```
class PacienteController extends AbstractController
{
    /**
     * @Route("/", name="paciente_index", methods={"GET"})
     */
    public function index(PacienteRepository $pacienteRepository): Response
    {
        return $this->render('paciente/index.html.twig', [
            'pacientes' => $pacienteRepository->pacientesOrdenadosDESC(),
        ]);
    }
```

Fig. 2.2

Patrón controlador. Clase PacienteController [Fuente: Elaboración propia]

Patrón experto: Es el principio básico de asignación de responsabilidades. Indica, por ejemplo, que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtendrá un diseño con mayor cohesión y así la información se mantiene encapsulada [CITATION Lar \l 21514]. Este patrón se pone en práctica en las clases entidades, expertas en la información y encargadas del manejo de la lógica del negocio. En la figura 2.3 se muestra la clase entidad *Paciente.php*, la cual es la experta en información para la gestión de los pacientes en el sistema:

```php
class Paciente
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     */
    private $nombre;

    /**
     * @ORM\Column(type="datetime")
     */
    private $fecha;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     */
    private $primerApellido;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     */
    private $segundoApellido;
```

Fig. 2.3 Patrón experto. Clase Paciente.php [Fuente: Elaboración propia]

Patrón creador: ayuda a identificar quien es el encargado (o quien debería tener la responsabilidad) de crear un determinado objeto [CITATION Lar \l 21514]. Este patrón se utiliza en la propuesta de solución en las clases gestoras a la hora de crear nuevos objetos. En la figura 2.4 se muestra como desde la clase *Paciente.php* se hace instancia al crear un objeto de tipo Paciente:

Fig. 2.4 Patrón creador. Clase Paciente.php [Fuente: Elaboración propia]

```
$paciente = new Paciente();
$form = $this->createForm(PacienteType::class, $paciente);
$form->add('Guardar', 'Symfony\Component\Form\Extension\Core\Type\SubmitType'

$form->handleRequest($request);

if ($form->isSubmitted() && $form->isValid()) {
    $entityManager = $this->getDoctrine()->getManager();

   $paciente->setFecha(new \DateTime('now'));

    $entityManager->persist($paciente);
    $entityManager->flush();
```

2.8.2 Patrones GOF (Gang of Four)

Los patrones GOF describen soluciones simples y elegantes a problemas específicos en el diseño de software. Se clasifican en tres grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento. También representan soluciones técnicas basadas en la programación orientada a objetos que favorecen la reutilización del código. (Iturralde, 2016)

Patrón decorador: Añade funcionalidad a una clase dinámicamente. Esto permite no tener que crear sucesivas clases que hereden de la primera incorporando la nueva funcionalidad, sino otras que la implementan y se asocian a la primera. Se aplica con la intención de proporcionar una forma flexible de introducir o eliminar funcionalidad de un componente sin modificar su apariencia externa o su función. (Iturralde, 2016)

Se evidencia el uso en la clase TestMedicoController en la funcionalidad editAction () a la hora de modificar los atributos de un objeto clase existente, sin cambiar la apariencia como se muestra en la figura 2.5

```
if ($form->isSubmitted() && $form->isValid()) {
    $entityManager = $this->getDoctrine()->getManager();

      $testMedico->setFecha(new \DateTime('now'));

    $entityManager->persist($testMedico);
    $entityManager->flush();
```

Fig. 2.5

Patrón decorador. [Fuente: Elaboración propia]

Patrón fachada: Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema. Es un tipo de patrón de diseño estructural. Viene motivado por la necesidad de estructurar un entorno de programación y reducir su complejidad con la división en subsistemas, minimizando las comunicaciones y dependencias entre estos. Este patrón se utiliza en la definición de las plantillas usando el motor de plantillas TWIG. Permite decorar la presentación de la información al usuario. (Iturralde, 2016)

A continuación, en la Figura 2.6 se muestra el uso del patrón fachada para la solución propuesta.

```
{{ form_start(form, {'attr': {'class': 'form-horizontal','novalidate':'novalidate' }}) }}
<div class="tab-pane active" id="tab_0">
    <div class="portlet light bg-inverse">
        <div class="portlet-title">
            <div class="caption">
                <i class="icon-equalizer font-red-sunglo"></i>
                <span class="caption-subject font-red-sunglo bold verda"><b>Adicionar paciente</b></span>
                <span class="caption-helper"></span>
            </div>
            <div class="actions btn-set"style="float: right;">
                <button class="btn btn-primary btn-sm" {{ form_widget(form.Guardar)}}</button>
                <button type="button" class="btn default btn-sm" id="volveratras"><i class="fa fa-angle-double-left"></i> Regresar</button>
            </div>
        </div>
        <div class="portlet-body form">
            <div class="form-body">
                <div class="row">

                    <div class="col-sm-6">
                        <div class="form-group {% if form.nombre.vars.valid==false %}has-error{% endif %}">
                            <label class="col-md-4 control-label">Nombre: <span class="required">*</span></label>
                            <div class="col-md-8">
                                {{ form_widget(form.nombre, { 'attr': {
                        'class' : 'form-control',
                        } }) }}
                                {% if form.nombre.vars.valid==false %}
                                    <span class="help-block">{{ form_errors(form.nombre)}}</span>
                                {% endif %}
                            </div>
                        </div>
                        <div class="form-group {% if form.primerApellido.vars.valid==false %}has-error{% endif %}">
                            <label class="col-md-4 control-label">Primer apellido: <span class="required">*</span></label>
                            <div class="col-md-8">
                                {{ form_widget(form.primerApellido, { 'attr': {
                        'class' : 'form-control',
                        } }) }}
                                {% if form.primerApellido.vars.valid==false %}
                                    <span class="help-block">{{ form_errors(form.primerApellido)}}</span>
                                {% endif %}
                            </div>
                        </div>
```

Fig. 2.6 Ilustración del patrón fachada. [Fuente: Elaboración propia]

2.9 Modelo de datos

Un modelo de datos es una parte esencial en el proceso de modelado de una base de datos. Este es una estructura abstracta que documenta y organiza la información de los datos y las relaciones entre ellos. El propósito de un modelo de datos es, por una parte, representar los datos y por otra, ser comprensible. En él se describen los datos, las relaciones de datos y la semántica de los datos. (Gómez & Olive, 2003).

En la figura 2.7 se definen cómo los datos se conectan entre sí y cómo se procesan y almacenan dentro del sistema.

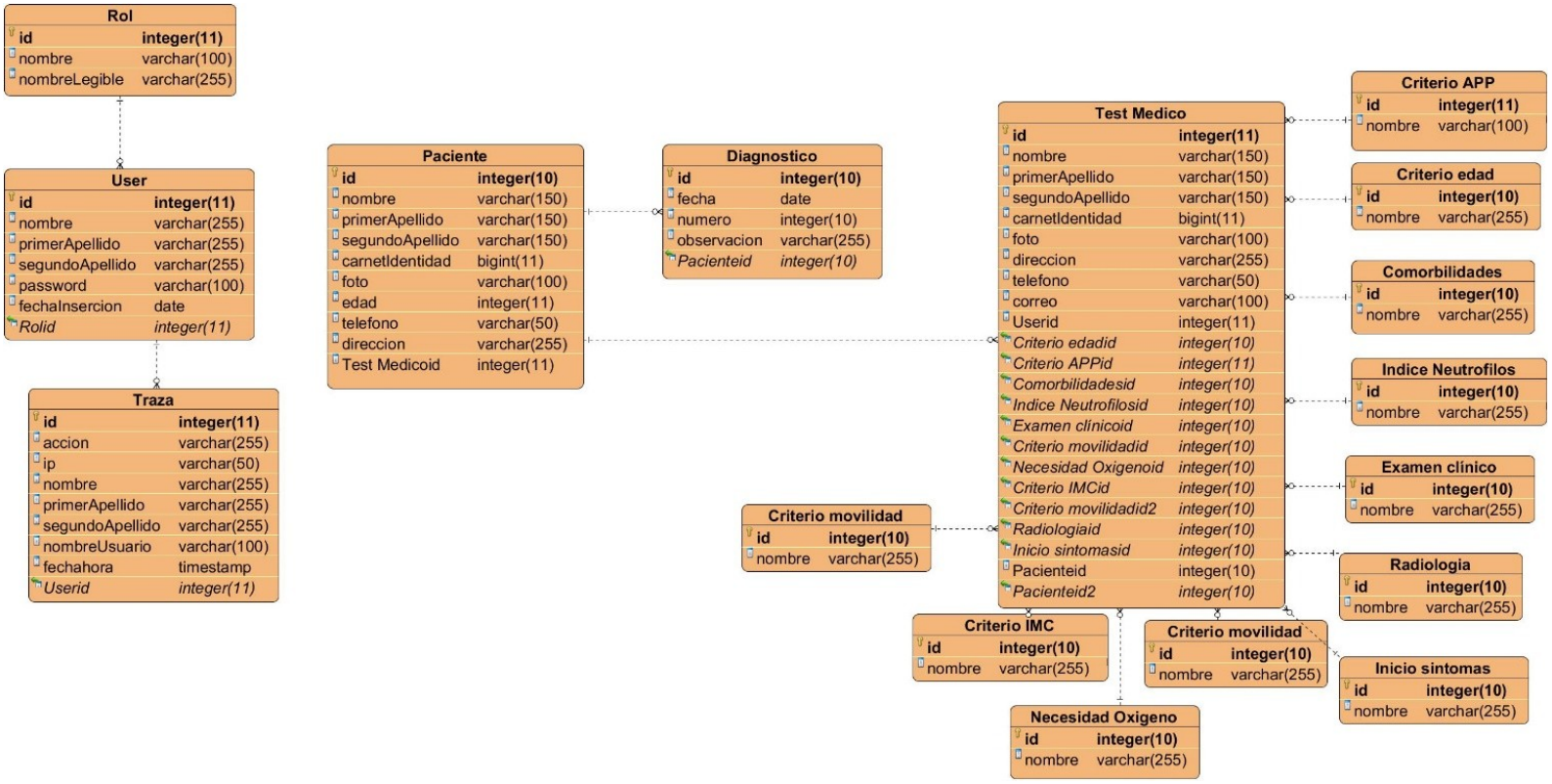


Fig. 2.7 Modelo de datos. [Fuente: Elaboración propia]

2.10 Diagrama de despliegue

El diagrama de despliegue muestra la distribución física del sistema. Permite conocer los requisitos necesarios de hardware y software para la puesta en marcha de la aplicación web. En la Figura 2.8 se muestra la representación de estos requisitos. En él se verá que el sistema trabaja a partir de una estación de trabajo conectado a la red a un servidor web, con conexión a un servidor de base de datos.

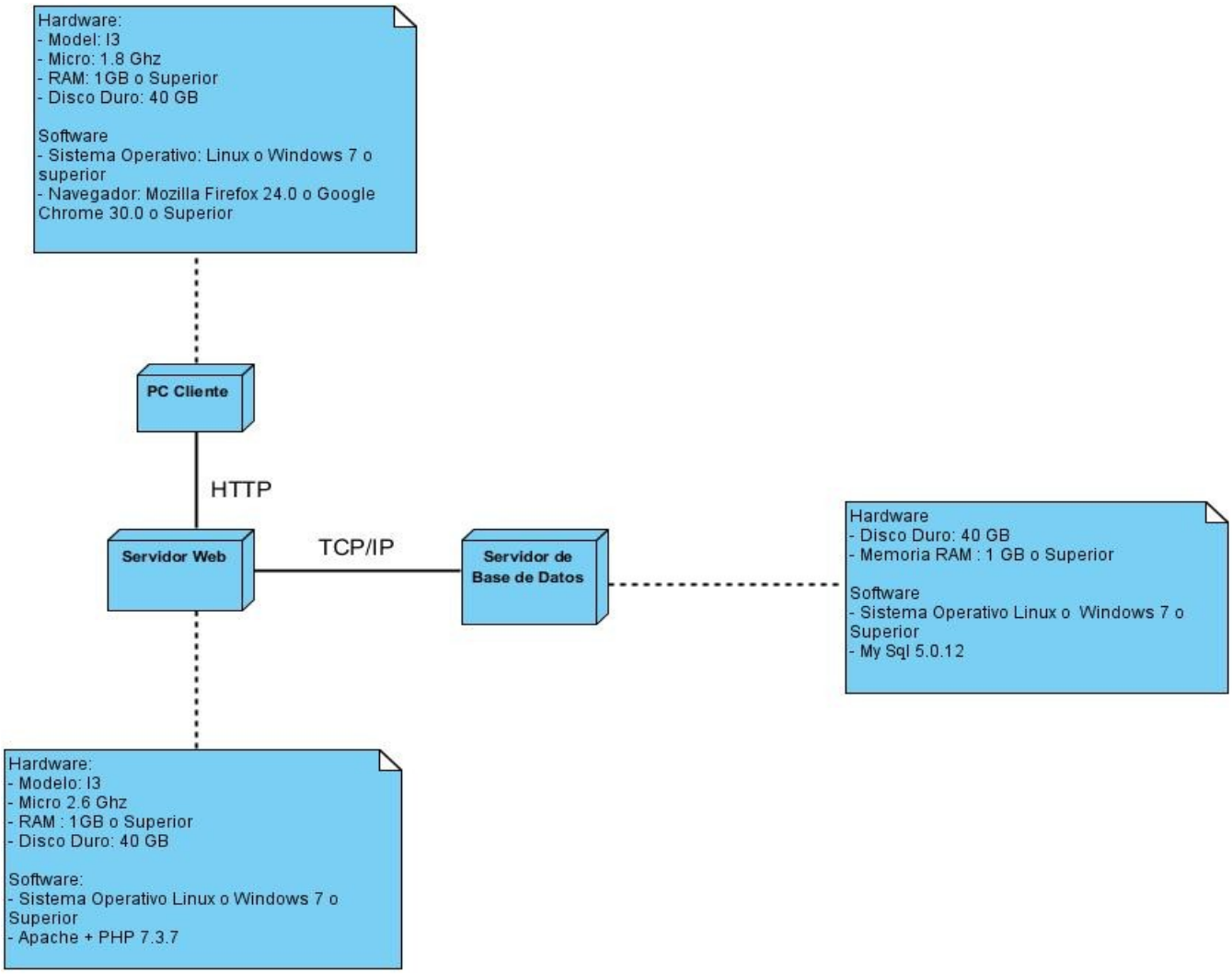


Fig. 2.8 Diagrama de despliegue. [Fuente: Elaboración propia]

A continuación, se explica el propósito de los nodos que se representaron en el diagrama de despliegue.

PC Cliente: El propósito es reflejar los puestos de trabajo conectados a la red de la institución médica que acceden a la aplicación.

Impresora: El propósito es una impresora conectada a la red de la institución médica, en la cual pueden imprimir todas las PC Cliente que la tengan configurada.

Servidor Web: El propósito es instalar la aplicación en el servidor web de la institución médica

para que todas las PC Clientes conectadas a la red accedan a las

funcionalidades de la misma.

Servidor de Base de Datos: El propósito es almacenar todos los registros que genera la aplicación en la base de datos.

2.11 Implementación y prueba de la solución

La fase de implementación y prueba es el período durante el cual se implementa la solución y se comprueba que es estable y utilizable. Como resultado en esta fase se obtendrá una aplicación que cumpla con las funciones necesarias para dar solución al problema existente en la institución médica.

2.11.1 Estructura interna de la solución

La estructura de directorios de Symfony varía según el tipo de proyecto que hallamos creado (framework-standard-edition, website-skeleton). Para el sistema informático para evaluar riesgo de complicaciones respiratorias o muerte a pacientes con Covid-19, la estructura interna esta complementada por el framework de desarrollo web Symfony en su versión 5.2.10 de la siguiente manera:

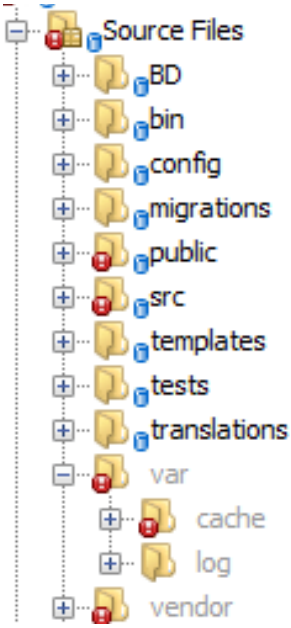


Fig. 2.9 Directorio de Symfony 5.2.10. [Fuente: Estructura interna de Symfony]

La figura 2.9 muestra:

- **Source Files/bin:** Los binarios del proyecto se ubican aquí, por ejemplo, la consola que se utiliza, posteriormente, para las llamadas a doctrine.
- **Source Files/config:** Contiene todos los ficheros de configuración del proyecto, por ejemplo, los archivos rounting.yml y security.yml.
- **Source Files/migrations:** Contiene las migraciones del proyecto
- **Source Files/public:** En el directorio se almacenan los controladores frontales y todas las hojas de estilo, ccs, JavaScript, imágenes entre otras.
- **Source Files/src:** Este directorio contiene el código relativo a los controladores, rutas, vistas y formularios. Es decir, el código específico de la lógica de negocio de la aplicación.
- **templates:** En esta ruta se tienen las plantillas TWIG y los ficheros de traducción necesarios para la presentación de la aplicación.
- **tests:** Contiene las aplicaciones de test al sistema.
- **var/cache:** Almacena todos los archivos de caché generados por la aplicación.
  var/logs: Almacena todos los archivos de logs generados por la aplicación.
- **vendor:** En este directorio se encuentran las dependencias de código de la aplicación.

2.12 Estándares de codificación

Se definen estándares de codificación porque un estilo de programación homogéneo en un proyecto permite que todos los participantes lo puedan entender en menos tiempo y que el código en consecuencia sea mantenible (Arias, 2019). Partiendo de lo antes expuesto se definen los siguientes estándares para la solución propuesta:

Nomenclatura de las clases: Los nombres de las clases comienzan con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se emplea notación PascalCasing, esta especifica que los identificadores y nombres de variables, métodos y funciones que están compuestos por múltiples palabras juntas, deben iniciar cada palabra con letra mayúscula, lo que posibilita que con sólo leerlo se reconozca el propósito de la misma.

Ejemplo: “TestMedico” en este caso el nombre de clase está compuesto por dos palabras iniciadas cada una con letra mayúscula.

Clases controladoras: Los nombres de las clases controladoras comienzan con la primera letra en mayúscula seguido por la palabra “Controller”. En caso de que sea un nombre compuesto comienza con mayúsculas los dos nombres acompañados de la palabra “Controller”. **Ejemplo:** “TestMedicoController “.

Clases repositorios: Las clases que se encuentran dentro de Repository comienzan con mayúscula y después del nombre llevan la palabra “Repository”. **Ejemplo**: “PacienteRepository”.

Clases entidades: Su nombre comienza con mayúsculas y están ubicadas dentro del directorio Entity del proyecto. Son las clases que crean el esquema de la base de datos según su contenido. **Ejemplo:** “User”.

Clases formularios: Su nombre comienza con mayúsculas y están ubicadas dentro del directorio Form del proyecto. **Ejemplo:** “PacienteType”.

Nomenclatura de las funcionalidades o métodos: El nombre de los métodos está descrito con la inicial del identificador en minúscula. **Ejemplo:** “index”.

Además de cumplir con el estándar anterior, los nombres de los métodos están seguidos de la palabra “Action” **Ejemplo:** “indexAction ()”.

A continuación, se explica cómo está estandarizada la estructura interna de las funcionalidades o métodos.

Importaciones: Las importaciones están en líneas separadas.

Codificaciones: Utilizar la codificación UTF-8.

Comentarios: Los comentarios deben ser lo suficientemente claros y concisos para que se entienda el propósito de lo que se está desarrollando. En caso de ser una función complicada se debe comentar su interior para lograr una mejor comprensión del código.

Operadores: Los operadores deben tener un espacio a la izquierda y un espacio a la derecha.

Variables:

- El nombre de las variables o constantes debe comenzar con minúsculas.
- Si el nombre de las variables está compuesto por dos palabras o más, se empleará el tipo **lowerCamelCase** de la notación **CamelCasing**.
- En las funciones que para su ejecución necesiten valores como parámetros, en caso de que sea más de un parámetro, estos deben ir separados por coma (,) y con un espacio después de la coma.

Conclusiones del capítulo

Con el empleo de la metodología elegida, AUP-UCI, se organizó de manera satisfactoria el desarrollo de la solución propuesta. Con la arquitectura del sistema MVC se facilitó la ampliación de las funcionalidades de la solución y la reutilización de los componentes de la presente solución en aplicaciones futuras. Los patrones de diseño y los estándares de codificación utilizados, permitieron obtener una solución flexible a la hora de introducir cambios, realizar mantenimientos y generar poca dependencia entre clases.

CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

En este capítulo se realiza una estrategia de prueba de software que proporciona una guía que describe los pasos que deben realizarse como parte de la prueba, cuando se planean y se llevan a cabo dichos pasos, y cuánto esfuerzo, tiempo y recursos se requieren. Por tanto, cualquier estrategia de prueba debe incorporar la planificación de la prueba, el diseño de casos de prueba, la ejecución de la prueba y la recolección y evaluación de los resultados. Una estrategia de prueba de software debe ser lo suficientemente flexible para promover un uso personalizado de la prueba. Al mismo tiempo, debe ser suficientemente rígida para alentar la planificación razonable y el seguimiento de la gestión conforme avanza el proyecto. (Pressman, 2010)

3.1 Pruebas de rendimiento

La prueba de rendimiento se diseña para poner a prueba el rendimiento del software en tiempo de corrida, dentro del contexto de un sistema integrado de rendimiento de software. Se realizan para medir la respuesta de la aplicación a distintos volúmenes de carga esperados, se centra en determinar la velocidad con la que el sistema bajo pruebas realiza una tarea en las condiciones particulares del escenario de pruebas. (Sommerville, 2005)

Carga: Este es el tipo más sencillo de pruebas de rendimiento. Una prueba de carga se realiza generalmente para observar el comportamiento de una aplicación bajo una cantidad de peticiones esperada. Esta carga puede ser el número esperado de usuarios concurrentes utilizando la aplicación y que realizan un número específico de transacciones durante el tiempo que dura la carga. Esta prueba puede mostrar los tiempos de respuesta de todas las transacciones importantes de la aplicación. Si la base de datos, el servidor de aplicaciones, etc. También se monitorizan, entonces esta prueba puede mostrar el cuello de botella en la aplicación. (Sommerville, 2005)

Estrés: Esta prueba se utiliza normalmente para romper la aplicación. Se va doblando el número de usuarios que se agregan a la aplicación y se ejecuta una prueba de carga hasta que se rompe. Este tipo de prueba se realiza para determinar la solidez de la aplicación en los momentos de carga extrema y ayuda a los administradores para determinar si la aplicación

rendirá lo suficiente en caso de que la carga real supere a la carga esperada. (Sommerville, 2005)

Las pruebas de carga y estrés se realizaron con la ayuda de la herramienta Apache JMeter v5.1.16, la cual permite medir el rendimiento de los recursos como servicios web, lenguajes dinámicos y bases de datos, entre otros.

Estas se llevaron a cabo en un ambiente seleccionado utilizando un ordenador con las siguientes características:

Hardware de prueba (PC cliente):

- Tipo de procesador: Intel(R) Core (TM) i3-4030U CPU @ 1.90GHz.
- RAM: 4 GB DDR3.
- Tipo de Red: Ethernet 10/100Mbps.

Hardware de prueba (PC servidor):

- Tipo de procesador: Intel(R) Core (TM) i3-4030U CPU @ 1.90GHz
- RAM: 4 GB DDR3.
- Tipo de Red: Ethernet 10/100Mbps.

Software instalado en ambas PC:

- Plataforma: SO, Windows 10 Pro (PC servidor) y SO Windows 10 Pro (PC cliente).
- Tipo de Sistema: Sistema operativo de 64 bits, procesador x64.
- Servidor de BD: MySQL 5.0.12.

A continuación, se muestra en la Figura 2.10 los resultados obtenidos por la herramienta Apache JMeter v5.1.1 con un total de doscientos usuarios conectados concurrentemente.

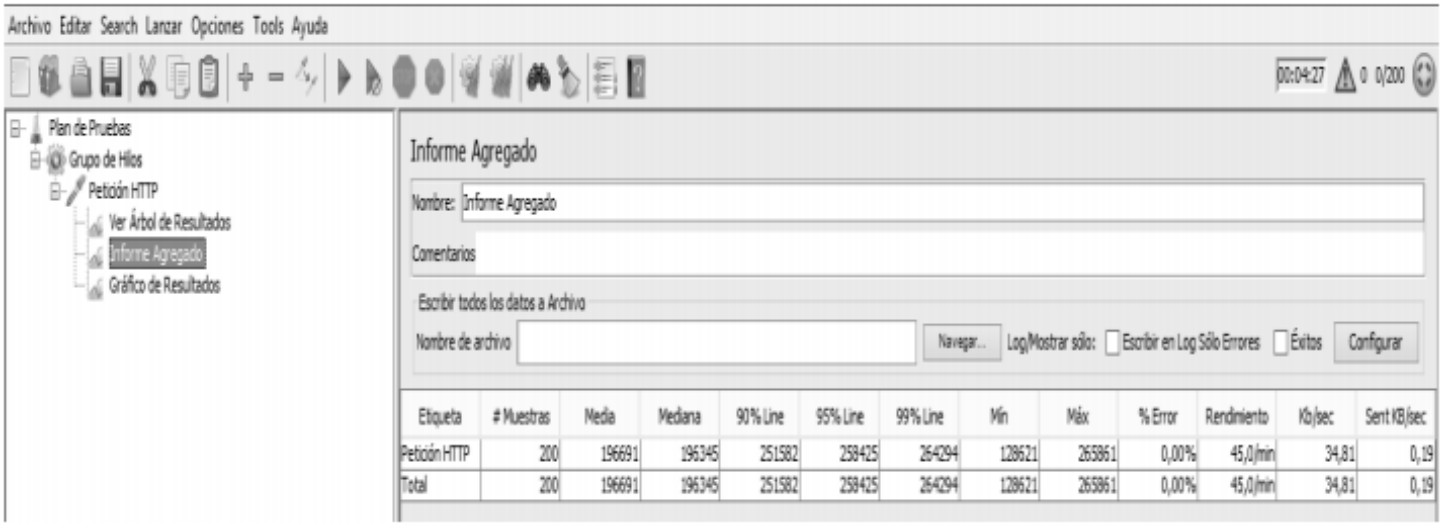


Fig. 3.1 Resultado de las pruebas de carga y estrés. [Fuente: Elaboración propia]

Las pruebas de rendimiento arrojaron como resultado que el tiempo total de las solicitudes es de 196345 milisegundos, realizándose con un total de 200 muestras. El tiempo total para los 200 hilos se calcula de la siguiente manera:

Tiempo Total= #Muestras * Media = 200 * 196345 = 39269000 milisegundos.

El tiempo promedio requerido por cada hilo se calcula de la siguiente manera:

Tiempo Promedio = (Media / 1000) / Cantidad de Hilos = (196345 / 1000) /200 = 0.981725 segundos.

3.2 Pruebas de integración

Las pruebas de integración son una técnica sistemática para construir la arquitectura del software mientras se llevan a cabo pruebas para descubrir errores asociados con la interfaz. El objetivo es tomar los componentes probados de manera individual y construir una estructura de programa que se haya dictado por diseño. (Pressman, 2010)

Las pruebas de integración se le realizaron al sistema para verificar la compatibilidad y el funcionamiento de las interfaces y módulos lo que arrojó como resultado que cada uno de los componentes y módulos del sistema se integran correctamente. También se aplicó para verificar que los elementos de software que interactúan entre si funcionan de manera correcta, en este caso se verifica que los nuevos reportes requeridos por el cliente se integran correctamente al sistema, además de cada una de sus funcionalidades por separado.

Las pruebas de integración arrojaron como resultado que existe una correcta integración entre los componentes internos del sistema.

3.3 Pruebas de aceptación

Se aplica este tipo de prueba con el objetivo de determinar, por parte del cliente, la aceptación o rechazo del sistema desarrollado. Este tipo de prueba es realizada por el usuario final en lugar de los ingenieros de software. Una prueba de aceptación puede variar desde una prueba de conducción informal hasta una serie de pruebas planificadas y ejecutadas sistemáticamente. La prueba de aceptación puede realizarse durante un periodo de semanas o meses, y mediante ella descubrir errores acumulados que con el tiempo puedan degradar el sistema. (Pressman, 2010)

Entre los tipos de pruebas de aceptación que existen en este caso en particular se realizan las pruebas alfa que son las que se desarrollan en la propia compañía desarrolladora del software con el equipo de pruebas del software. Las pruebas de aceptación se les realizan a todas las funcionalidades del sistema. A continuación, se muestra un ejemplo de un caso de prueba realizado, en la tabla 3, los campos de la tabla son los siguientes:

- **Nombre del probador:** nombre de la persona que realiza la prueba.
- **Descripción:** se describe la funcionalidad que se desea probar.
- **Condiciones de Ejecución:** muestra las condiciones que deben cumplirse para poder llevar a cabo el caso de prueba, estas condiciones deben ser satisfechas antes de la ejecución del caso de prueba para que se puedan obtener los resultados esperados.

- **Entradas/Pasos de Ejecución:** descripción de cada uno de los pasos seguidos durante el desarrollo de la prueba, se tiene en cuenta cada una de las entradas que hace el usuario con el objetivo de ver si se obtiene el resultado esperado.

- **Resultado esperado:** breve descripción del resultado que se espera obtener con la prueba realizada.

- **Evaluación de la prueba:** acorde al resultado de la prueba realizada se emite una evaluación sobre la misma. Esta evaluación tiene uno de los dos valores que a continuación se describen:

    1. Satisfactorio

    2. Insatisfactorio

Caso de prueba de aceptación	
Código de caso de prueba: HU_39	**Nombre de historia de usuario**: Insertar paciente
Nombre de la persona que realiza la prueba: Reynaldo Cesar González	
Descripción de la prueba: Prueba la funcionalidad insertar paciente	
Entrada/Pasos de ejecución: En el menú del sistema se entra a Gestión "Pacientes"	

Escenarios:	**Resultados Esperados:**	**Evaluación de la prueba:**
EC 1.1 Insertar paciente con los datos correctamente.	Se inserta el paciente, luego se muestra un cartel que dice que se ha insertado correctamente y muestra el listado de los pacientes del sistema incluyendo el nuevo insertado.	Satisfactoria
EC 1.2 Insertar paciente	Muestra un mensaje según el err	

introduciendo caracteres inválidos.	or encontrado: 1. Si es un campo de número: "Solo puede introducir números". 2. Si es un campo de letra: "Solo puede introducir letras".	Satisfactoria
EC 1.3 Adicionar paciente dejando campos vacíos.	Muestra un mensaje según el error encontrado: 1. Si es seleccionable: "Campo obligatorio. Debe seleccionar un elemento". 2. Si es un campo de texto o de número "Campo obligatorio".	Satisfactoria

Tabla. 3.1 Caso de prueba de aceptación del método: Insertar paciente [Fuente: Elaboración propia]

Las pruebas de aceptación arrojaron como resultado que no existía ninguna no conformidad por parte del cliente. Los resultados en todos los casos de prueba fueron satisfactorios por lo que se acepta el producto.

3.4 Pruebas funcionales

La prueba funcional se centra en comprobar que los sistemas desarrollados funcionan acorde a las especificaciones funcionales y requisitos del cliente, con el objetivo de validar que las funcionalidades implementadas funcionen correctamente y cumplan con los requisitos definidos con anterioridad. Estas pruebas se enfocan solamente en las entra-

das y salidas del sistema, sin preocuparse de tener conocimiento de la estructura interna del programa de software. Para obtener el detalle de cuáles deben ser esas entradas y salidas, se basan únicamente en los requisitos de software y especificaciones funcionales. Al definir una prueba de caja negra lo principal es identificar los datos de prueba (entradas) y el resultado esperado del sistema al ingresar esos datos, bien sean los datos de salida o algún comportamiento específico. (Pressman, 2010)

Este tipo de pruebas permite encontrar:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores de rendimiento.

Dentro de la prueba de **caja negra** se incluyen varias técnicas de pruebas tales como:

- **Partición de equivalencia:** divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
- **Grafos de causa-efecto:** permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.
- **Análisis de valor de frontera:** El análisis de valor de frontera conduce a una selección de casos de prueba que revisan los valores de frontera.

En la tabla 3.2 se muestran las variables para el caso de prueba adicionar tipo de licencia:

No	Nombre del campo	Clasificación	Valor nulo	Descripción
1	Número	Campo de texto	No	Se inserta una cadena que identifique al diagnóstico.
2	Fecha	Plugin para seleccionar una	No	Se inserta una fecha para el diagnóstico.

		fecha		
3	Paciente	Campo seleccionable	No	Se selecciona el paciente para el diagnóstico.
4	Observación	Campo de texto	No	Se inserta una descripción que identifique al diagnóstico.
5				

Tabla. 3.2 Variables para el caso de prueba insertar diagnóstico [Fuente: Elaboración propia]

Se aplicó el método de caja negra a la solución específicamente la técnica de partición de equivalencia con el siguiente caso de prueba y obteniendo como resultado la tabla 3.3

Nombre del requisito	Descripción	Escenarios de pruebas	Flujo del escenario
Insertar diagnóstico	El usuario con los permisos correspondientes adiciona un tipo de licencia proporcionando la información necesaria.	**EP 1.1** Insertar un diagnóstico insertando un número, una observación, una fecha y un paciente de forma correcta.	El usuario hace la petición al sistema para adicionar un nuevo diagnóstico. ➢ En la clase controladora se activa la función para adicionar un diagnóstico. ➢ Se muestra una ventana con el formulario para adicionar un nuevo diagnóstico. ➢ El usuario entra los datos requeridos para adicionar un nuevo diagnóstico de manera correcta (número, observación, paciente y

			fecha). ➢ El sistema valida que los datos introducidos son correctos. ➢ El sistema almacena en la base de datos los datos del nuevo diagnóstico creado. ➢ El sistema muestra un cartel indicando que se ha adicionado correctamente el diagnóstico y muestra el listado de todos los diagnósticos.
		EP 1.2 Adicionar un diagnóstico insertando algún dato incorrecto.	➢ El usuario hace la petición al sistema para adicionar un nuevo diagnóstico. ➢ En la clase controladora se activa la función para adicionar un nuevo diagnóstico. ➢ Se muestra una ventana con el formulario para adicionar un nuevo diagnóstico. ➢ El usuario entra los datos requeridos para

			adicionar un nuevo diagnóstico de manera incorrecta (número, observación, paciente y fecha). ➢ El sistema muestra en color rojo un cartel indicando el campo con valor inválido. ➢ El usuario debe corregir el campo. ➢ El sistema valida que los datos introducidos son correctos. ➢ El sistema almacena en la base de datos los datos del nuevo diagnóstico creado. ➢ El sistema muestra una notificación que se ha adicionado correctamente el diagnóstico y muestra el listado de todos los diagnósticos.
		EP 1.3 Adicionar un	➢ El usuario hace la petición al sistema para

		diagnóstico dejando uno o más campos vacíos.	adicionar un nuevo diagnóstico. ➢ En la clase controladora se activa la función para adicionar un nuevo diagnóstico ➢ Se muestra una ventana con el formulario para adicionar un nuevo diagnóstico. ➢ El usuario deja alguno de los campos vacíos (número, observación, fecha o paciente). ➢ El sistema muestra el campo vacío en rojo y muestra un cartel diciendo que el campo es obligatorio. ➢ El usuario debe llenar el campo vacío. ➢ El sistema valida que los datos introducidos son correctos. ➢ El sistema almacena en la base de datos los datos del nuevo diagnóstico creado. ➢ El sistema muestra una notificación que se ha

			adicionado correctamente el diagnóstico y muestra el listado de todos los diagnósticos.
		EP 1.4 Adicionar un diagnóstico introduciendo valores inválidos.	➢ El usuario hace la petición al sistema para adicionar un nuevo diagnóstico. ➢ En la clase controladora se activa la función para adicionar un nuevo diagnóstico. ➢ Se muestra una ventana con el formulario para adicionar un nuevo tipo de licencia. ➢ El usuario introduce valores inválidos (número, observación, fecha o paciente). ➢ El sistema muestra un cartel en rojo diciendo que el campo debe proporcionar un formato correcto. ➢ El usuario debe corregir el campo con valor incorrecto.

			➢ El sistema valida que los datos introducidos son correctos. ➢ El sistema almacena en la base de datos los datos del nuevo diagnóstico creado. ➢ El sistema muestra una notificación que se ha adicionado correctamente el diagnóstico y muestra el listado de todos los diagnósticos.

Tabla. 3.3 Caso de prueba Insertar diagnóstico [Fuente: Elaboración propia]

Después de aplicado el método de caja negra para validar las funcionalidades implementadas en la solución se concluye que los resultados obtenidos en cuanto a funcionalidad, fueron satisfactorios. Se les dio solución a las no conformidades detectadas durante la aplicación de las pruebas, obteniendo así un mejor funcionamiento de la aplicación.

En la Gráfico 1 se muestra el resultado de las pruebas con las no conformidades detectadas por iteración.

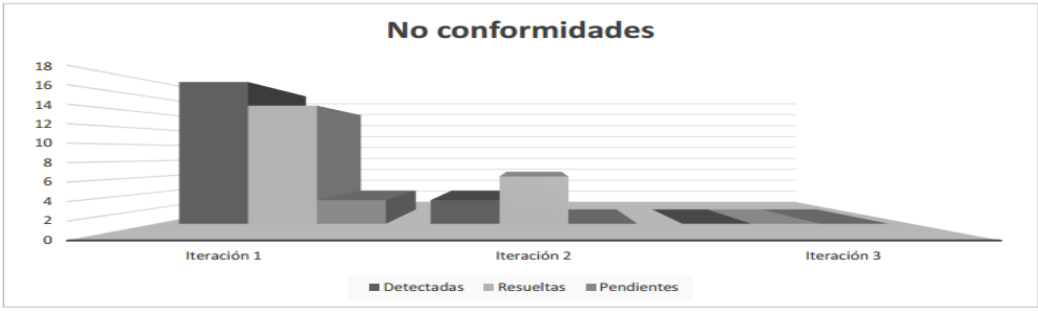


Gráfico 1. No conformidades de la prueba de caja negra [Fuente: Elaboración propia]

Para la validación de los requisitos funcionales se realizaron un total de 3 iteraciones donde se encontraron un total de 21 no conformidades, 18 en la primera iteración de las cuales 15 fueron resueltas quedando 3 pendientes. En la segunda iteración se detectaron 3 no conformidades que fueron resueltas y también las 3 pendientes de la primera iteración y en la tercera iteración no se encontraron no conformidades por lo que la aplicación de la prueba fue satisfactoria.

3.5 Pruebas unitarias

Una prueba unitaria es una forma de comprobar el correcto funcionamiento de una unidad de código. Por ejemplo, en diseño estructurado o en diseño funcional una función o un procedimiento, en diseño orientado a objetos una clase. Esto sirve para asegurar que cada unidad funcione correctamente y eficientemente por separado. Además de verificar que el código hace lo que tiene que hacer, se verifica que sea correcto el nombre, los nombres y tipos de los parámetros, el tipo de lo que se devuelve, que si el estado inicial es válido entonces el estado final es válido. (Barrientos, 2014)

En la aplicación de las pruebas unitarias se utiliza la técnica de caja blanca. Esta técnica se basa en el diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivarlos.

Permite al ingeniero del software obtener casos de prueba que:

1. Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.

2. Ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa.

3. Ejecuten todos los bucles en sus límites operacionales.

4. Ejerciten las estructuras internas de datos para asegurar su validez.

Se considera a la prueba de caja blanca como uno de los tipos de pruebas más importantes que se le aplican al software, logrando como resultado que disminuya en un gran porciento el

número de errores existentes en los sistemas y por ende una mayor calidad y confiabilidad. (Pressman, 2010)

Dentro del método se incluyen varias pruebas tales como:

- **La prueba del camino básico:** permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los pasos que se siguen para aplicar esta técnica son:

1. A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
2. Se calcula la complejidad ciclomática del grafo.
3. Se determina un conjunto básico de caminos independientes.
4. Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

- **La prueba de condición:** es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa.
- **La prueba de flujo de datos:** se selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.
- **La prueba de bucles:** es una técnica de prueba de caja blanca que se centra exclusivamente en la validez de las construcciones de bucles. (Pressman, 2010)

Dentro de la técnica caja blanca se emplea el método del camino básico para realizar las pruebas. La Figura 3.2 muestra el código del método editAction (), que permite editar una clase dado su identificador y la solicitud y en la Figura 3.3 se muestra el grafo de flujo asociado al método editAction ().

Fig.3.2 Método para editar una clase [Fuente: Elaboración propia]

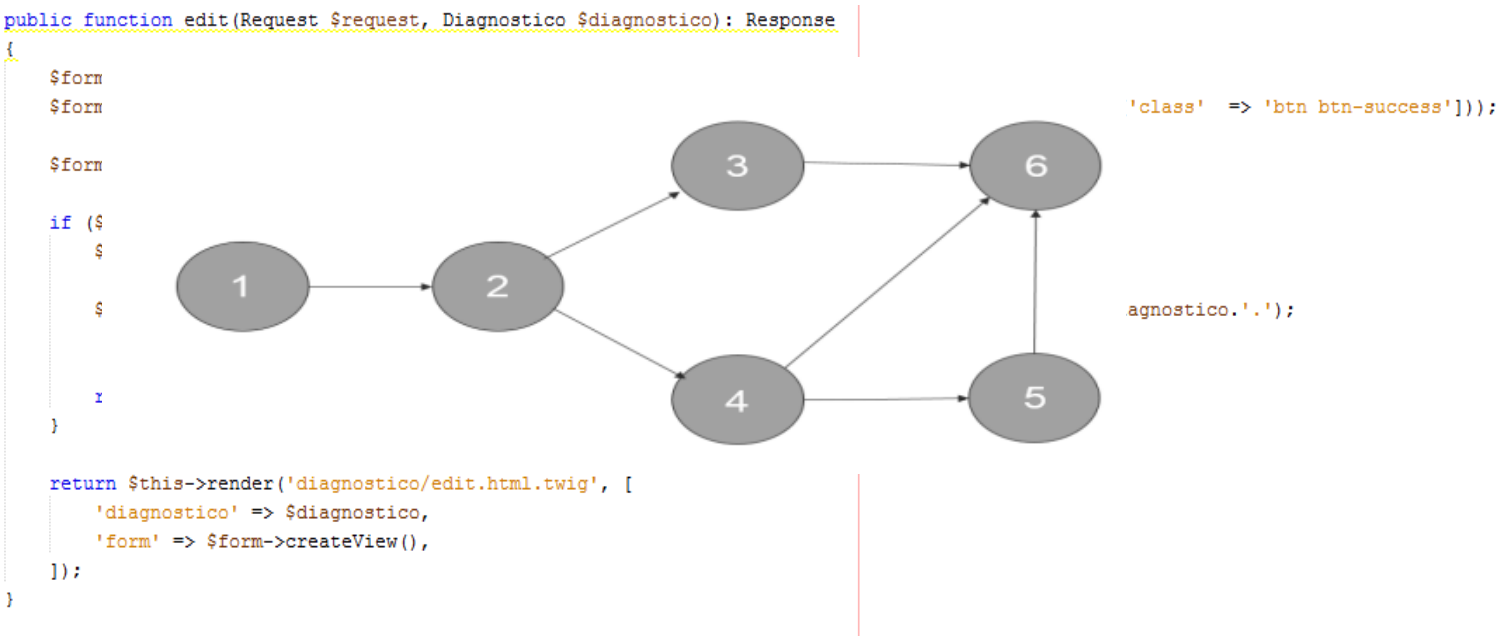


Fig.3.3 Grafo de flujo asociado [Fuente: Elaboración propia]

SonarQube

SonarQube es una plataforma de código abierto para la inspección continua de la calidad del código a través de diferentes herramientas de análisis estático de código fuente. Proporciona métricas que ayudan a mejorar la calidad del código de un programa permitiendo a los equipos de desarrollo hacer seguimiento y detectar errores y vulnerabilidades de seguridad para mantener el código limpio.

Es una herramienta esencial para la fase de testing y auditoría de código dentro del ciclo de desarrollo de una aplicación y se considera perfecta para guiar a los equipos de desarrollo durante las revisiones de código. Soporta una etapa de inspección continua.

Refleja la Complejidad Ciclomática calculada en base al número de caminos a través del código normalmente observado a nivel de métodos o funciones individuales. Nos indica el número de bloques de líneas duplicados. Ayuda a evitar resultados distintos en operaciones iguales.

La complejidad ciclomática es una medición de software que proporciona una evaluación cuantitativa de la complejidad lógica de un programa. Cuando se usa en el contexto del método de prueba de la ruta básica o camino básico, el valor calculado por la complejidad ciclomática define el número de rutas independientes del conjunto básico de un programa. Brinda una cota superior para el número de pruebas que debe realizar, a fin de asegurar que todos los enunciados se ejecutaron al menos una vez. La complejidad ciclomática tiene fundamentos en la teoría de gráficos y proporciona una medición de software extremadamente útil. La complejidad se calcula de tres formas:

1. El número de regiones del gráfico de flujo corresponde a la complejidad ciclomática.
2. La complejidad ciclomática **V(G)** para un gráfico de flujo **G** se define como:
    - **V(G) = E - N + 2** donde **E** es el número de aristas del gráfico de flujo y **N** el número de nodos del gráfico de flujo.
    - **V(G) = P + 1** donde P es el número de nodos predicado[3] contenidos en el gráfico de flujo **G**. (Pressman, 2010)

    En el gráfico de flujo anterior, la complejidad ciclomática es calculada usando cada uno de los algoritmos indicados:

    1. El gráfico de flujo tiene tres regiones.
    2. V(G) = 7 aristas - 6 nodos + 2 = 3.

V(G) = 2 nodos predicado + 1 = 3.

La complejidad ciclomática del gráfico de flujo asociado al método editAction () tiene valor 3. V(G)

proporciona la cota superior sobre el número de rutas linealmente independientes a través de la estructura de control del programa. Por tanto, las rutas o caminos posibles son las siguientes:

3 Un nodo predicado es el que representa una condicional if o case, es decir, que de él salen varios caminos.

Número de rutas	Rutas básicas
1	**1-2-3-6**
2	**1-2-4-6**
3	**1-2-4-5-6**

Tabla. 3.4 Rutas básicas del grafo de flujo [Fuente. Elaboración propia]

Una vez determinadas las rutas o caminos básicos de flujo, se pasa a ejecutar los casos de pruebas para cada camino resultante. Los datos deben elegirse de modo que las condiciones en los nodos predicados se establezcan de manera adecuada conforme se prueba cada ruta. Cada caso de prueba se ejecuta y compara con los resultados esperados. Una vez completados todos los casos de prueba, el examinador puede estar seguro de que todos los enunciados del programa se ejecutaron al menos una vez.

Ruta Básica 1: 1-2-3-6	
Descripción	Proceso para editar una clase, no es necesario introducir parámetros porque lo que se va a hacer es editar una clase

	existente.
Condición de ejecución	La entrada no sea una clase
Resultado esperado	El sistema muestra el mensaje “La clase no existe”.
Resultado	Satisfactorio

Tabla. 3.5 Caso de prueba de Ruta básica 1 [Fuente. Elaboración propia]

Ruta Básica 2: 1-2-4-6	
Descripción	Proceso para editar una clase.
Condición de ejecución	El formulario no ha sido enviado.
Resultado esperado	El sistema redirecciona nuevamente para editar la clase.
Resultado	Satisfactorio

Tabla. 3.6 Caso de prueba de Ruta básica 2 [Fuente. Elaboración propia]

Ruta Básica 3: 1-2-4-5-6	
Descripción	Proceso para editar una clase.
Condición de ejecución	Proceso para editar una clase, para ello es necesario editar los atributos de la clase existente y que el formulario sea enviado.
Resultado esperado	El sistema edita la clase y guarda en la base de

	datos los nuevos parámetros y muestra el siguiente mensaje “Se ha editado correctamente la clase”.
Resultado	Satisfactorio

Tabla. 3.7 Caso de prueba de Ruta básica 3 [Fuente. Elaboración propia]

Después de aplicado el método del camino básico, en el cual se realizaron 3 iteraciones de acuerdo a la cantidad de rutas básicas obtenidas, se concluye que, de los 3 casos de pruebas realizados, todos tuvieron resultados satisfactorios. Las 5 no conformidades detectadas fueron resueltas, por tanto, la aplicación de la prueba fue satisfactoria en su totalidad.

3.6 Pruebas de seguridad

Cualquier sistema basado en computadora que gestione información sensible o cause acciones que puedan dañar (o beneficiar) de manera inadecuada a individuos es un blanco de penetración inadecuada o ilegal.

La penetración abarca un amplio rango de actividades: hackers que intentan penetrar en los sistemas por deporte, empleados resentidos que intentan penetrar por venganza, individuos deshonestos que intentan penetrar para obtener ganancia personal ilícita. La prueba de seguridad intenta verificar que los mecanismos de protección que se construyen en un sistema en realidad lo protegerán de cualquier penetración impropia.

Durante la prueba de seguridad, quien realiza la prueba juega el papel del individuo que desea penetrar al sistema. Quien realice la prueba puede intentar adquirir contraseñas por medios administrativos externos; puede atacar el sistema con software a la medida diseñado para romper cualquier defensa que se haya construido; puede abrumar al sistema, y por tanto negar el servicio a los demás; puede causar a propósito errores del sistema con la esperanza de penetrar durante la recuperación; puede navegar a través de datos inseguros para encontrar la llave de la entrada al sistema. (Pressman, 2010)

Las pruebas de seguridad se aplicaron hasta el nivel 2, el nivel 1 está dirigido a todo tipo de software y el nivel 2 es para aplicaciones que contienen datos sensibles que requieran protección. En el nivel 1 arrojó un total de 4 elementos afectados para 1 no conformidad crítica 3 no conformidades no críticas quedando evaluada de insatisfactoria la prueba. Todas las no conformidades fueron resueltas. En la segunda iteración la prueba fue evaluada de satisfactoria, debido a que no se encontraron no conformidades. A continuación, se muestran los resultados arrojados por las pruebas de seguridad de nivel 1.

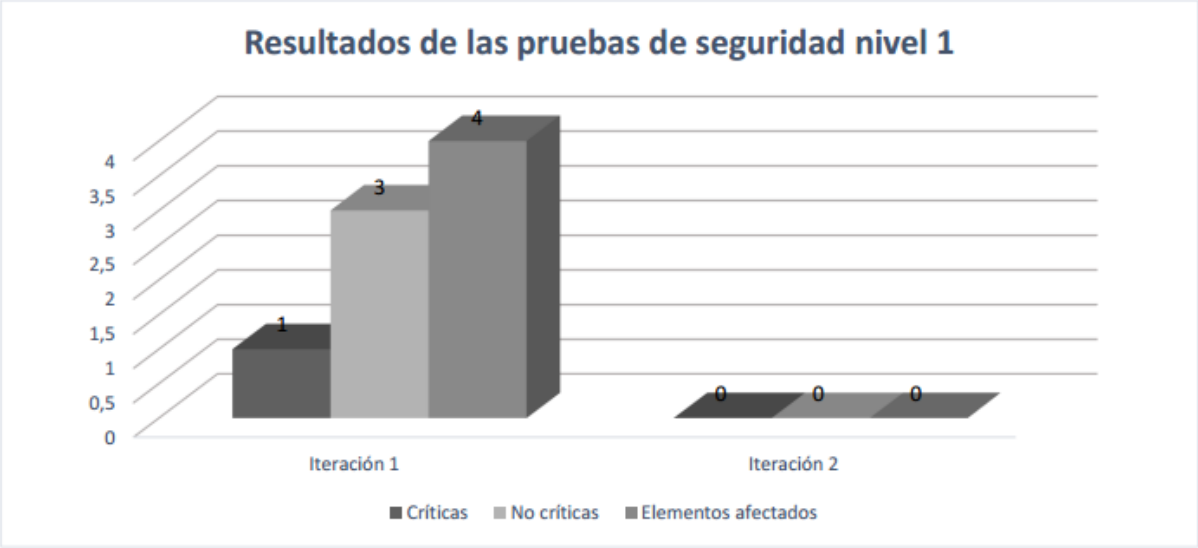


Gráfico 2. Resultados de las pruebas de seguridad de nivel 1 [Fuente. Elaboración propia]

Se desarrollaron las pruebas de nivel 2 con la ayuda de la herramienta Acunetix Web Vulnerabitily Scan[3], que establece alertas de tipo: alta, media, baja e informacional. En la primera iteración se detectaron 4 no conformidades medias, 1 no conformidad baja y 3 no conformidades informacionales para un total de 8 no conformidades, todas fueron solucionadas. Se consideran de mayor importancia las no conformidades altas y medias. Las no conformidades referentes al entorno de despliegue de la aplicación son descartadas por el equipo de desarrollo porque existen entidades encargadas de solucionarlas. A continuación, se muestran los resultados arrojados por las pruebas de seguridad de nivel 2.

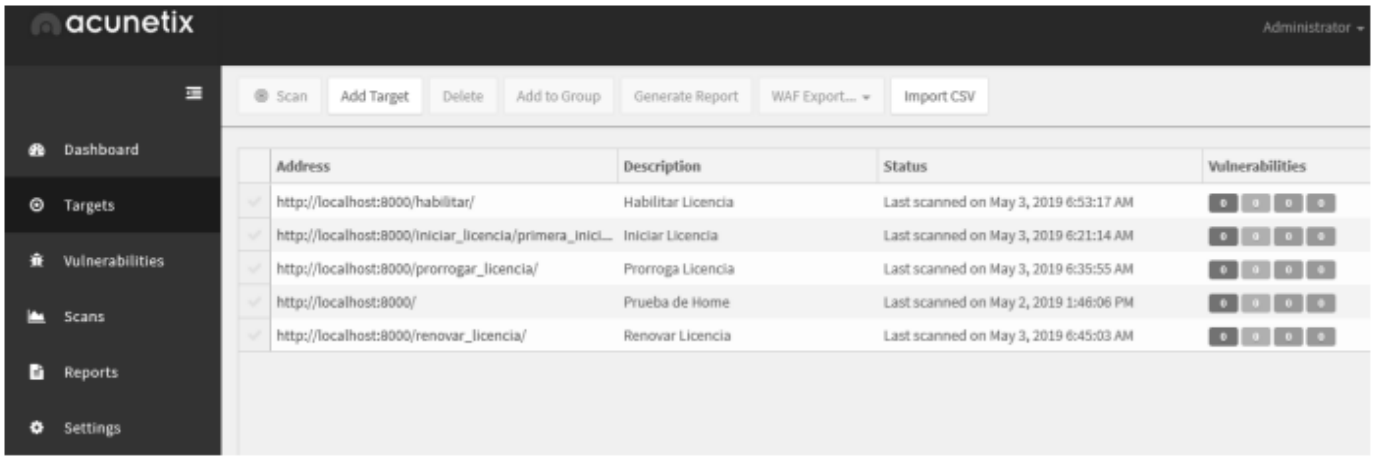


Fig.3.4 Resultados de las pruebas de seguridad iteración 2 en la herramienta [Fuente: Elaboración propia]

3 https://www.acunetix.com/vulnerability-scanner/

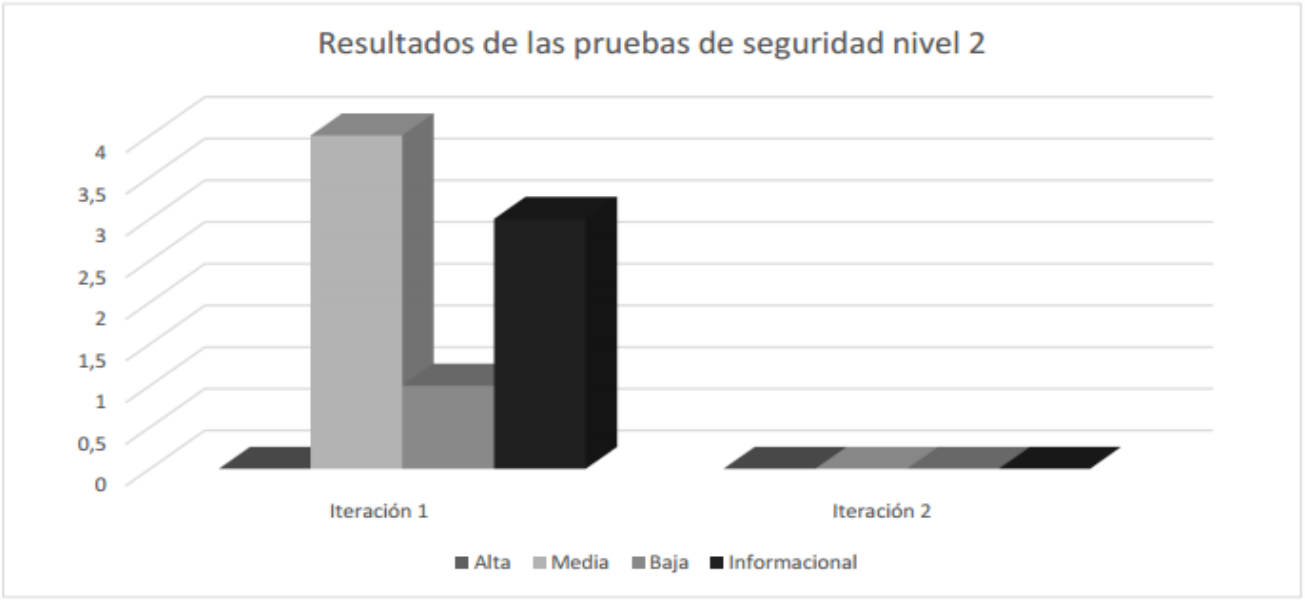


Gráfico 3. Resultados de las pruebas de seguridad de nivel 2 [Fuente. Elaboración propia]

Conclusiones del capítulo

En este capítulo se establecieron los estándares de codificación a tener en cuenta para la implementación del Sistema informático para evaluar riesgo complicaciones respiratorias o

muerte a pacientes con Covid-19, lo que permitió garantizar que el código posea alta calidad, reducción de errores y pueda ser mantenido fácilmente. De igual forma permite la reutilización por otros desarrolladores que lo necesiten. Se realizaron seis tipologías de pruebas a) rendimiento, b) integración, c) aceptación, d) funcionales, e) unitarias y f) seguridad. De forma general se detectaron 33 no conformidades, todas resueltas en las diferentes iteraciones ejecutadas.

Las pruebas de software sirvieron para comprobar que el flujo de trabajo de las funcionalidades es correcto y que el subsistema cumple con los requisitos definidos. La validación del resultado de la investigación, demuestra el cumplimiento de la relación causa efecto de la variable independiente.

CONCLUSIONES FINALES

- El estudio de los referentes teóricos vinculados con software que se encargan de la informatización de procesos relacionados con los sistemas de evaluación de riesgo de complicaciones respiratorias o muertes a pacientes con COVID 19, tanto en el ámbito nacional como extranjero, demuestra la necesidad de desarrollar la solución propuesta, teniendo en cuenta que las soluciones internacionales estudiadas difieren del esquema establecido en Cuba para la gestión de estos procesos y las soluciones nacionales no tienen o no está desarrollada en su máximo potencial una base de datos para la información y seguimiento de los datos.

- La metodología AUP-UCI escenario 4, permitió la obtención de un conjunto de métodos coherentes y relacionados por principios comunes en el proceso de desarrollo Sistema informático para evaluar riesgo complicaciones respiratorias o muerte a pacientes con Covid-19.
- El empleo de técnicas de captura de requisitos, patrones de diseño y arquitectónicos, así como el uso de estándares de codificación en la implementación de la solución propuesta, permitió obtener una solución, flexible al mantenimiento y a la introducción de cambios.
- La realización de las pruebas de software seleccionadas permitió la verificación de la propuesta de solución.

RECOMENDACIONES

- Continuar con el estudio de las nuevas tecnologías y tendencias para mejorar las potencialidades de la aplicación.

- Utilizar el sistema en otras unidades de salud similares a la institución médica.

REFERENCIAS BIBLIOGRÁFICAS

Association of Modern Technologies Professionals. 2019. It Knowledge portal. [En línea] 2019. http://www.itinfo.am/eng/software-development-methodologies/.

Bos, Bert. 2019. w3schools. W3C. [En línea] 2019. https://www.w3.org/Style/CSS/Overview.en.html.

Domínguez, Dorado M. 2018. *Todo Programación*. Nº 13. Págs. 32-34. s.l.: Editorial Iberprensa (Madrid), 2018.

Espinosa, Leslie Camila Legrá. 2020. *Componente para la gestión del turnado en el Expediente Judicial Electrónico*. La Habana: s.n., 2020.

Gaines, Jeff, Boyd, Geraldine y Copley, Della. 2017. Visual Paradigm Online. Visual Paradigm Online. [En línea] 2017. https://online.visual-paradigm.com/es/features/.

Gómez, C., & Olive, A. (2003). *Diseño de sistemas software en UML*. Barcelona, España: EDICIONS
UPC.
Iturralde, O. J. (2016). *Introducción a Los Patrones de Diseño: Un Enfoque Práctico*. Platform,
CreateSpace Independent Publishing.
Carter C, Notter J. Cuidados Intensivos de Enfermería En Pacientes Con Covid-19. Elsevier Health Sciences; 2022.)

Ionos. 2020. Ionos, Digital Guide. Tutorial de PHP: fundamentos básicos para principiantes. [En línea] 1&1, 16 de marzo de 2020. https://www.ionos.es/digitalguide/paginas-web/creacion-de-paginas-web/tutorial-de-php-fundamentos-basicos-para-principiantes/.

ICTEA. 2019. ictea.com. ictea.com. [En línea] 19 de 9 de 2019. http://www.ictea.com/cs/index.php?rp=/knowledgebase/8663/iQue-es-el-lenguaje-de-programacion-PHP.html.

Lucid. 2018. lucidchart.com. lucidchart.com. [En línea] 2 de 12 de 2018. https://www.lucidchart.com/pages/es/que-es-el-lenguaje-unificado-de-modelado-uml.

Marin, Rafael. 2019. revistadigital.inesem.es. revistadigital.inesem.es. [En línea] 16 de abril de 2019. https://revistadigital.inesem.es/informatica-y-tics/los-gestores-de-bases-de-datos-mas-usados/.

Mendez. 2018. uniciencista.gfrodriguez.online.[En línea] 10 de febrero de 2018. http://uniciencista.gfrodriguez.online/2018/02/herramientas-case-principales-usos.html.

Pressman, R. S. (2010). *INGENIERÍA DEL SOFTWARE. UN ENFOQUE PRÁCTICO*. México: McGRAW-HILL. Obtenido de www.FreeLibros.me

Pérez Valdés, Damián. 2007. maestrosdelweb. maestrosdelweb. [En línea] 3 de julio de 2007. http://www.maestrosdelweb.com/que-es-javascript/.

Sommerville, I. (2005). Madrid: Pearson Educación.

Uniwebsidad. (noviembre de 2018). 2.1. *El patrón MVC (Symfony 1.4, la guía definitiva)*. (uniwebsidad)
Obtenido de https://uniwebsidad.com/libros/symfony-1-4/capitulo-2/el-patron-mvc

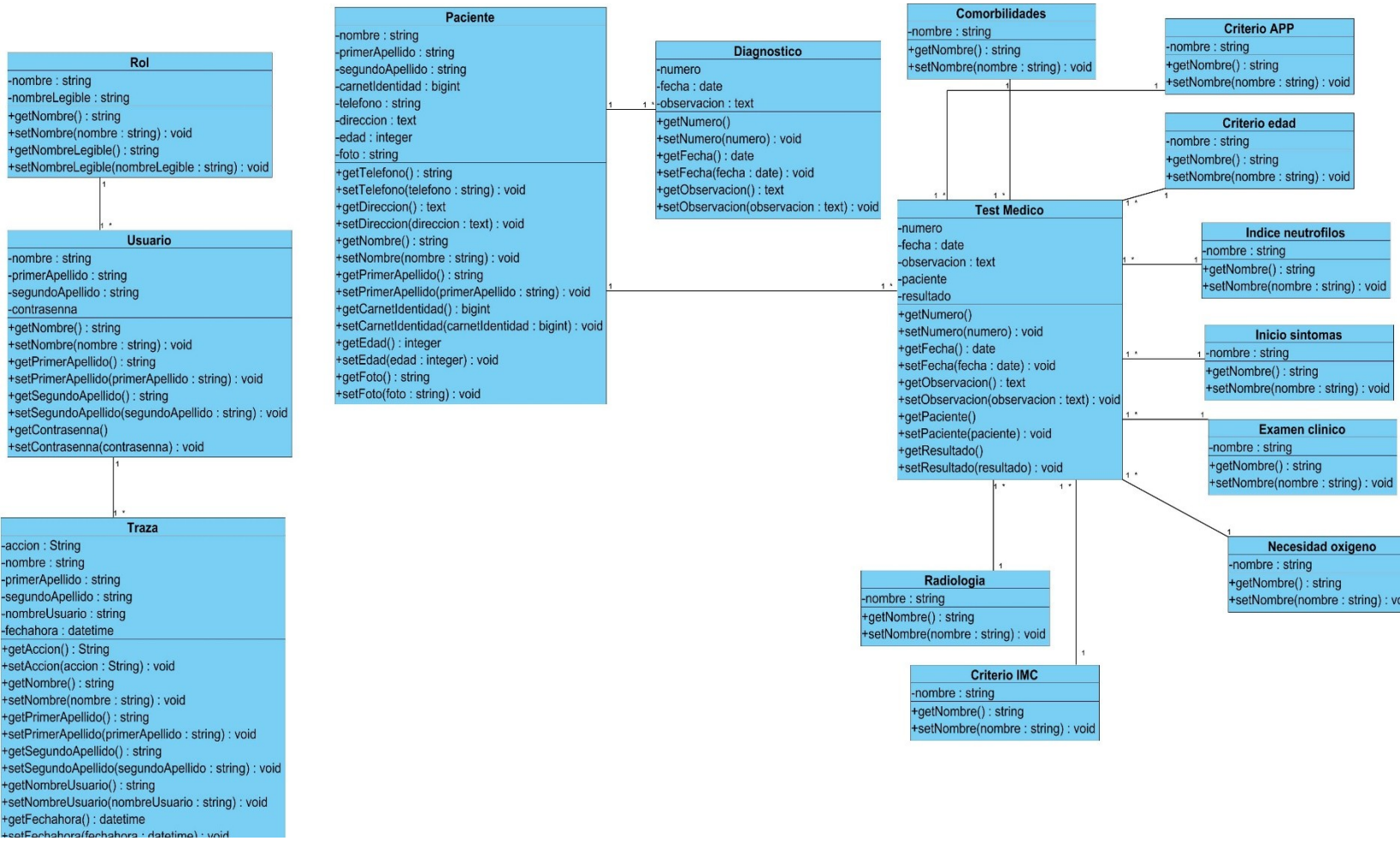
“COVID Alert NY” App Provides Power and Peace of Mind to Users. Accessed November 28, 2022. https://www.oswegocounty.com/news_detail_T17_R1089.php

COVID Symptom Study. Accessed November 28, 2022. https://health-study.joinzoe.com/

MINSAP R. Pesquisador Virtual para el enfrentamiento a la COVID-19: una autopesquisa responsable. Sitio oficial de gobierno del Ministerio de Salud Pública en Cuba. Published April 21, 2020. Accessed November 28, 2022. https://salud.msp.gob.cu/pesquisador-virtual-para-el-enfrentamiento-a-la-covid-19-una-autopesquisa-responsable/

MINSAP R. Cuba COVID-19. Sitio oficial de gobierno del Ministerio de Salud Pública en Cuba. Published February 27, 2020. Accessed November 28, 2022. https://salud.msp.gob.cu/infomed-lanza-aplicacion-movil-sobre-el-covid-19/

ANEXOS



Anexo1. Diagrama de clases