**2 - MARCO TEORICO**

Las agencias de turismo desempeñan un papel crucial en la industria turística al servir como intermediarias entre los proveedores de servicios (aerolíneas, hoteles, tours, etc.) y los viajeros. Ofrecen una variedad de servicios, como la gestión de reservas, itinerarios de viaje y asesoramiento a los clientes.

* Desafíos en la Operación de Agencias de Turismo:
  + **Gestión de reservas:** La eficiente gestión de reservas es esencial para evitar errores en fechas y garantizar la disponibilidad de servicios.
  + **Procesos de pagos:** La gestión de pagos es crítica para mantener un flujo de efectivo adecuado y garantizar una experiencia positiva para los clientes.
  + **Marketing y promoción:** Las estrategias de marketing son esenciales para atraer y retener clientes en un mercado competitivo.
  + **Capacitación de guías:** La capacitación y retención del personal, incluyendo guías turísticos, influye en la calidad del servicio al cliente.
  + **Cumplimiento normativo:** El cumplimiento de las regulaciones y requisitos legales en la industria turística es vital para evitar sanciones y multas.
  + **Tecnología obsoleta:** El uso de sistemas obsoletos puede limitar la automatización de procesos y dificultar la adaptación a las tendencias tecnológicas actuales.
* Importancia de la Optimización y Modernización:
  + La optimización y modernización de las operaciones son esenciales para superar los desafíos mencionados y para mantener la competitividad en el mercado actual.
  + La eficiencia operativa resulta en una mejor gestión de recursos, costos reducidos y una experiencia de cliente mejorada.
* Sistemas de Información en Agencias de Turismo:
  + Los sistemas de información desempeñan un papel fundamental en la modernización de las agencias de turismo, ya que pueden abordar los desafíos mencionados.
  + Estos sistemas permiten la gestión eficiente de reservas, automatización de procesos, seguimiento de pagos, análisis de datos para estrategias de marketing y cumplimiento normativo.
  + La implementación de un sistema de administración y gestión puede ser la solución para mejorar la eficiencia operativa y la toma de decisiones adecuadas en las agencias de turismo.
* Investigación y desarrollo de soluciones:
  + La investigación y el desarrollo de soluciones específicas para agencias de turismo, como sistemas de información personalizados, es crucial para abordar los desafíos mencionados.
  + La tecnología y la innovación desempeñan un papel fundamental en la modernización y optimización de las operaciones turísticas.

Este marco teórico proporciona un contexto sólido para investigar y desarrollar soluciones que aborden los desafíos en la operación de la agencia de turismo (JIWAKY) y promuevan la eficiencia operativa y la calidad del servicio.

**2-1** **SISTEMA DE INFORMACION**

Un sistema de información representa la sinergia de diversos elementos que colaboran de manera coordinada para respaldar las operaciones de una empresa o negocio. En su definición más amplia, este sistema no se limita estrictamente a la presencia de componentes electrónicos (hardware), aunque en la práctica, con frecuencia se hace referencia a un "sistema de información computarizado" por la prominencia de la tecnología.

Según “Cohen y Asin (2000)”, esta perspectiva se centra en la dimensión tecnológica, mientras que otros expertos como “Rodríguez Rodríguez y Daureo Campillo (2003)” ofrecen una definición más abarcadora.

Ellos describen un Sistema de Información (S.I.) como un conjunto de procedimientos, tanto manuales como automatizados, y funciones diseñadas para abarcar la recopilación, procesamiento, evaluación, almacenamiento, recuperación, síntesis y distribución de información dentro de una organización. Este sistema tiene la misión de facilitar el flujo eficaz de información desde su origen hasta su destinatario final.

**2-1.1** **ELEMENTOS DE UN SISTEMA DE INFORMACIÓN**

La función de un sistema de información se puede dividir en tres actividades fundamentales que generan los datos necesarios para que las organizaciones tomen decisiones, supervisen operaciones, resuelvan problemas y desarrollen nuevos productos o servicios. Estas actividades son las siguientes: entrada, procesamiento y salida, como se ilustra en la FIGURA 1.

La actividad de entrada se encarga de recopilar datos en su estado original desde dentro de la organización o desde su entorno externo. Posteriormente, el procesamiento se encarga de transformar estos datos en un formato significativo y útil. Finalmente, la actividad de salida se encarga de transmitir la información procesada a las personas que la utilizarán o a las operaciones para las que está destinada. Además de estas tres actividades principales, los sistemas de información requieren una función de retroalimentación. Esta función implica que la información procesada se devuelve a los miembros apropiados de la organización, lo que les ayuda a evaluar o corregir la etapa de entrada según sea necesario (Laudon & Laudon, 2012).

* Entrada: En esta fase, se capturan o recopilan datos desde fuentes internas o externas. Estos datos pueden ser de naturaleza diversa, como transacciones, informes, formularios o datos recopilados de sensores. La entrada es el punto de inicio del flujo de información en el sistema.
* Proceso: En el proceso, los datos recopilados se transforman, organizan, analizan y se convierten en información significativa. Esta etapa puede involucrar cálculos, comparaciones, clasificaciones, y otras operaciones para generar resultados útiles para la organización.
* Salida: La etapa de salida implica la presentación de la información procesada de manera comprensible y accesible para los usuarios finales. Esto puede incluir informes, gráficos, alertas o cualquier otro formato que facilite la toma de decisiones o la realización de tareas. La salida comunica los resultados del procesamiento a las personas o actividades que la utilizarán.

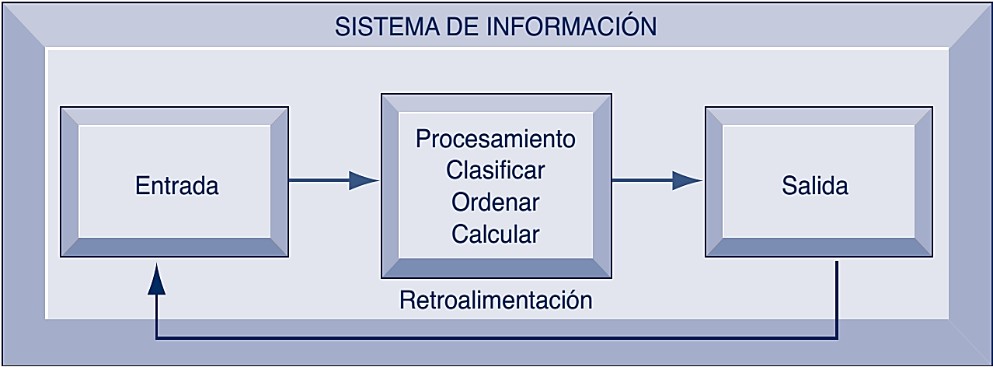


FIGURA 1. ACTIVIDADES DE UN SISTEMA DE INFORMACIÓN  
FUENTE: (Laudon & Laudon, 2012)

**2-1.2 TIPOS DE SISTEMAS DE INFORMACION**

Los sistemas de información se diseñan para cumplir una variedad de objetivos, adaptándose a las necesidades específicas de una empresa. Los sistemas de procesamiento de transacciones (TPS, Transaction Processing Systems) operan en el nivel más básico y operativo de una organización. Por otro lado, los sistemas de automatización de oficinas (OAS, Office Automation Systems) y los sistemas de trabajo del conocimiento (KWS, Knowledge Work Systems) respaldan las tareas relacionadas con la gestión del conocimiento.

En un nivel superior, encontramos los sistemas de información gerencial (MIS, Management Information Systems) y los sistemas de apoyo a la toma de decisiones (DSS, Decision Support Systems), diseñados para abordar cuestiones de gestión y decisiones más complejas. Estos sistemas facilitan la toma de decisiones a nivel ejecutivo y estratégico.

Adicionalmente, los sistemas de apoyo a la toma de decisiones en grupo (GDSS, Group Decision Support Systems) y los sistemas de trabajo colaborativo apoyados por computadora (CSCWS, Computer-Supported Collaborative Work Systems) se orientan a respaldar la toma de decisiones en grupo y ayudar a enfrentar situaciones de toma de decisiones semiestructuradas o no estructuradas a nivel colectivo.

En conjunto, esta variedad de sistemas de información se adapta a las necesidades específicas de una organización y aporta apoyo en diversos niveles de operación, gestión y toma de decisiones (Kendall & Kendall, 2005).

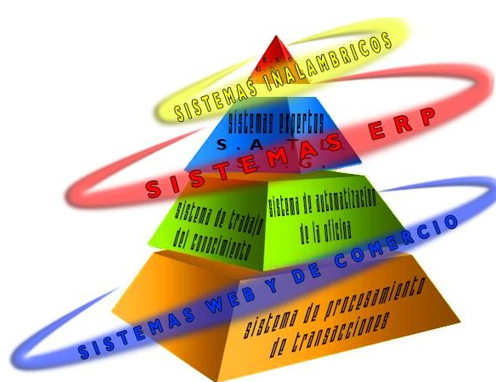


FIGURA 2. TIPOS DE SISTEMAS DE INFORMACION   
FUENTE. (Kendall & Kendall, 2005)

**2-1.2.1** **SISTEMA DE PROCESAMIENTO DE TRANSACCIONES**

Los sistemas de procesamiento de transacciones a nivel empresarial se refieren a aquellos sistemas diseñados para gestionar y registrar las actividades comerciales y financieras de manera rutinaria y cotidiana. Estas transacciones pueden abarcar una amplia gama de operaciones, como el procesamiento de nóminas, la facturación a clientes, el control de inventario y la gestión de depósitos bancarios. La naturaleza específica de estas transacciones puede variar significativamente en función del tipo de empresa y su sector de actividad.

**2-1.2.2** **SISTEMA DE APOYO A LA TOMA DE DECISIONES**

Un Sistema de Apoyo a la Toma de Decisiones, en un sentido general, es un sistema informático diseñado para facilitar y mejorar el proceso de toma de decisiones. En un contexto más específico, se trata de un sistema de información interactivo, flexible y adaptable, especialmente creado para respaldar la resolución de problemas de gestión que no siguen una estructura predefinida, con el objetivo de mejorar la calidad de las decisiones tomadas. Estos sistemas se basan en el uso de datos, proporcionan una interfaz de usuario amigable y permiten a los usuarios tomar decisiones basadas en un análisis integral de la situación.

**2-2 METODOLOGIA SCRUM**

**2-2.1 VISION GENERAL DE SCRUM**

Scrum es un marco de trabajo que permite a las personas abordar desafíos complejos y adaptativos al tiempo que entregan productos con el máximo valor posible, de manera productiva y creativa. En sus fundamentos, Scrum se caracteriza por ser:

* Ligero: Es de naturaleza ágil y no está cargado de procesos complejos.
* Fácil de entender: Su concepto central es accesible y comprensible para todos los involucrados.
* Extremadamente difícil de dominar: Aunque es sencillo de entender, su maestría requiere tiempo y experiencia.

Scrum ha sido un marco de trabajo ampliamente utilizado desde principios de la década de 1990 para gestionar el desarrollo de productos complejos. No es en sí un proceso o una técnica para construir productos, sino un marco dentro del cual se pueden aplicar diversas técnicas y procesos.

“En un nivel más específico, según "SCRUM PRIMER" de Deemer, Benefield, Larman y Vodde (2012), Scrum se define como un enfoque en el cual equipos multifuncionales colaboran en la creación iterativa e incremental de productos o proyectos. El trabajo se estructura en ciclos llamados Sprints o iteraciones, con una duración típica de dos semanas. Cada Sprint se enfoca en un objetivo claro y tangible acordado por el equipo, y durante su transcurso no se pueden agregar elementos nuevos. Los cambios se adaptan en el siguiente Sprint.”

Al final de cada Sprint, el equipo realiza una revisión con las partes interesadas y demuestra lo que han logrado. El feedback recibido se incorpora en el siguiente Sprint. Scrum pone un fuerte énfasis en entregar un producto completamente funcional al final de cada iteración, lo que significa que cada incremento es una versión "terminada" del producto.

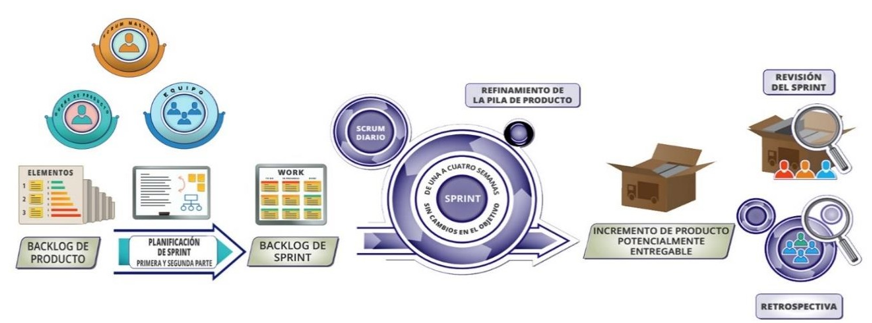


FIGURA 3. VISION GENERAL DE SCRUM   
FUENTE. (Deemer, Benefield, Larman, & Vodde, 2012)

**2-2.2** **EL EQUIPO SCRUM**

Según lo descrito por Schwaber y Sutherland en 2013, un Equipo Scrum se compone de tres roles fundamentales: el Dueño de Producto (Product Owner), el Equipo de Desarrollo (Development Team) y el Scrum Master. Estos equipos son característicamente autoorganizados y multifuncionales, lo que significa que tienen la capacidad de tomar decisiones internas y contar con todas las habilidades necesarias para llevar a cabo sus tareas sin depender de fuentes externas.

La autonomía de los equipos autoorganizados les permite determinar la mejor manera de abordar sus responsabilidades sin que fuerzas externas intervengan en su toma de decisiones. Además, la naturaleza multifuncional de estos equipos garantiza que cuenten con todas las competencias requeridas para completar su trabajo de manera independiente, sin requerir la intervención de personas ajenas al equipo. En última instancia, el diseño de los equipos Scrum está orientado a maximizar la flexibilidad, la creatividad y la productividad en el entorno de trabajo.

**2-2.2.1 DUEÑO DE PRODUCTO (PRODUCT OWNER)**

El propietario del producto (product owner) es quien toma las decisiones del cliente. Su responsabilidad es el valor del producto. Para simplificar la comunicación y toma de decisiones es necesario que este rol recaiga en una única persona. Si el cliente es una organización grande, o con varios departamentos, puede adoptar la forma de comunicación interna que consideren oportuna, pero en el equipo de desarrollo sólo se integra una persona en representación del cliente, y ésta debe tener el conocimiento suficiente del producto y las atribuciones, necesarias para tomar las decisiones que le corresponden. (Palacio, 2015)

**2-2.2.2 MAESTRO SCRUM (SCRUM MASTER)**

El Scrum Master cumple un rol esencial en la implementación exitosa de la metodología Scrum. Su responsabilidad principal radica en educar y guiar a cada miembro del equipo, asegurándose de que comprendan y apliquen efectivamente los principios y prácticas de Scrum. En esencia, el Scrum Master actúa como un mentor, facilitador y defensor de la metodología.

Además de ser un educador, el Scrum Master también es un líder que opera en función del equipo. Su enfoque se centra en servir al Equipo Scrum, brindando apoyo constante para que puedan alcanzar su máximo potencial. El Scrum Master desempeña un papel clave en la eliminación de obstáculos y en la creación de un entorno propicio para el equipo.

Adicionalmente, el Scrum Master actúa como un intermediario entre el Equipo Scrum y las partes externas interesadas, ayudándoles a comprender la dinámica del equipo y facilitando interacciones beneficiosas. Su objetivo es optimizar la colaboración y la comunicación para maximizar el valor generado por el Equipo Scrum. En resumen, el Scrum Master es un guía, un defensor y un facilitador que desempeña un papel fundamental en la adopción y éxito de Scrum.

**2-2.2.3 EL EQUIPO DE DESARROLLO (DEVELOPMENT TEAM)**

El Equipo de Desarrollo está compuesto por profesionales que desempeñan un papel fundamental en la entrega de un Incremento de producto que se considera "Terminado" al final de cada Sprint, con la capacidad de ser implementado en producción. Estos miembros son los únicos responsables de llevar a cabo el proceso de creación del Incremento. En este contexto, se les concede la autonomía y la responsabilidad de organizar y gestionar su propio trabajo.

La organización otorga al Equipo de Desarrollo la flexibilidad y la autoridad necesarias para tomar decisiones relacionadas con su labor. Esta autonomía permite que el equipo colabore de manera sinérgica, lo que resulta en una mejora significativa de la eficiencia y la efectividad en la ejecución de su trabajo. En resumen, el Equipo de Desarrollo está capacitado y facultado para producir resultados de alta calidad de manera independiente y coordinada.

**2-2.3 PROCESOS DE SCRUM**

El desarrollo se lleva a cabo de manera iterativa e incremental, organizado en ciclos definidos como "Sprints," que tienen una duración preestablecida, generalmente de 2 a 4 semanas. Al concluir cada Sprint, se obtiene una versión del software con nuevas funcionalidades listas para su uso. Durante cada nuevo Sprint, el equipo ajusta y mejora la funcionalidad existente, al tiempo que agrega nuevas características, priorizando aquellas que aportan un mayor valor de negocio.

El proceso de desarrollo se rige por los siguientes componentes clave:

* Pila de Producto (Product Backlog): Un conjunto de requisitos descritos en un lenguaje no técnico y priorizados en función del valor de negocio. Estos requisitos se revisan y ajustan de forma regular a lo largo del proyecto.
* Planificación del Sprint (Sprint Planning): En esta reunión, el Product Owner presenta las historias del Backlog en orden de prioridad. El equipo determina la cantidad de historias que se comprometerá a completar durante el Sprint y organiza cómo llevará a cabo esta tarea.
* Sprint: Una iteración de duración fija en la que el equipo trabaja para convertir las historias del Product Backlog, a las que se ha comprometido, en una nueva versión plenamente funcional del software.
* Pila del Sprint (Sprint Backlog): Una lista de las tareas necesarias para llevar a cabo las historias del Sprint.
* Reunión de Sprint Diario (Daily Sprint Meeting): Una reunión diaria de no más de 15 minutos, en la que el equipo sincroniza sus actividades y discute lo que hicieron el día anterior, lo que planean hacer ese día y si enfrentan algún impedimento.
* Demostración y Retrospectiva (Demo y Retrospectiva): Una reunión que se celebra al final de cada Sprint. En la demostración, el equipo presenta las historias completadas a través de una demostración del producto. Luego, en la retrospectiva, el equipo reflexiona sobre lo que se hizo bien, evalúa los procesos que podrían mejorarse y discute estrategias para su perfeccionamiento.

**2-3** **LENGUAJE UNIFICADO DE MODELADO (UML)**

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual de propósito general que se emplea para especificar, visualizar, construir y documentar los componentes de un sistema de software, según lo propuesto por Booch, Rambaugh y Jacobson en 2004. Este enfoque conceptualiza un sistema como una colección de objetos independientes que colaboran para cumplir una función que en última instancia beneficia a un usuario externo.

Los autores tenían cuatro objetivos principales en mente al desarrollar el UML:

* Representar sistemas en su totalidad, superando la mera representación de un solo programa.
* Establecer una conexión explícita entre los conceptos modelados y los componentes ejecutables del sistema.
* Tener en cuenta la complejidad inherente a sistemas críticos y de gran escala.
* Crear un lenguaje de modelado que resulte utilizable tanto para seres humanos como para máquinas.

Para comprender más a fondo esta metodología, es crucial considerar la creación de un modelo conceptual del lenguaje, el cual requiere tres elementos clave: la construcción de bloques básicos que representan los conceptos, las reglas que definen cómo estos bloques pueden interactuar entre sí y algunos mecanismos comunes que se aplican en todo el lenguaje UML. Este enfoque proporciona una base sólida para representar y comunicar de manera efectiva la estructura y el comportamiento de sistemas de software complejos.

**2-3.1 MODELOS**

Un modelo es una representación, generalmente en un medio distinto al objeto o sistema que se desea representar. Esta representación busca capturar los aspectos esenciales y significativos del objeto o sistema desde un punto de vista específico, a menudo simplificando o omitiendo detalles menos relevantes.

* Modelo de Casos de Uso: Este modelo permite la comunicación efectiva entre desarrolladores y clientes al definir actores, casos de uso y sus relaciones, estableciendo así los requisitos funcionales del sistema.
* Modelo de Análisis: Representa el nivel más alto del análisis y se organiza en paquetes que abstraen subsistemas y capas de diseño, ayudando a dividir y gestionar eficazmente un sistema complejo.
* Modelo de Diseño: Se enfoca en la realización física de los casos de uso, considerando tanto los requisitos funcionales como no funcionales y las restricciones del entorno de implementación. Este modelo sirve como base para las actividades de implementación y como una abstracción de la estructura del sistema.
* Modelo de Despliegue: Describe la distribución física del sistema en términos de cómo se distribuye su funcionalidad entre los nodos de cómputo. Este modelo influye en el diseño y es fundamental para las actividades de implementación.
* Modelo de Implementación: Detalla cómo los elementos del modelo de diseño, como las clases, se convierten en componentes de implementación, como archivos de código fuente o ejecutables. Además, organiza estos componentes según estructuras y modularización disponibles.
* Modelo de Prueba: Se enfoca en las pruebas de componentes ejecutables, incluyendo pruebas de integración y de sistema, para asegurar que el sistema funcione correctamente. Puede también abordar aspectos específicos del sistema que requieran pruebas particulares.

Cada uno de estos modelos cumple un papel crucial en el ciclo de desarrollo de software, abordando aspectos específicos y contribuyendo a la creación exitosa del sistema en su conjunto.

**2-3.2 DIAGRAMAS**

Según la Real Academia de la Lengua Española, un diagrama es un dibujo geométrico utilizado para ilustrar una proposición, resolver un problema o representar gráficamente la ley de variación de un fenómeno. A través de él, se presentan visualmente las relaciones entre las diversas partes de un conjunto o sistema.

**2-3.2.1 DIAGRAMA DE CASOS DE USOS**

Un diagrama de casos de uso es una representación gráfica que muestra un conjunto de casos de uso, actores y sus relaciones en un sistema. Estos diagramas capturan el comportamiento del sistema desde la perspectiva de los usuarios que interactúan con él. Esencialmente, un diagrama de casos de uso ilustra las interacciones entre el sistema, sus actores externos y los usuarios. Proporciona una visión gráfica de quiénes utilizan el sistema y cómo esperan interactuar con él.



FIGURA 4. COMPONENTES DEL DIAGRAMA DE CASOS DE USO  
FUENTE. (Booch, Rambaugh, & Jacobson, 2004)

En detalle, los elementos de un diagrama de casos de uso incluyen:

* Casos de Uso: Representan requisitos funcionales del sistema y describen las interacciones típicas entre los usuarios y el sistema. Definen la interfaz externa del sistema y especifican su comportamiento funcional.
* Actores: Representan roles coherentes desempeñados por los usuarios en sus interacciones con los casos de uso. Estos actores pueden ser personas, dispositivos, objetos o incluso otros sistemas que interactúan con el sistema en cuestión. Los actores se suelen representar con un rectángulo con el estereotipo "actor," que se asemeja a un ícono humano dibujado con líneas. En el diagrama, se establecen relaciones estándar, como asociación, generalización y uso, entre los actores y los casos de uso.
* Relación de Dependencia: Esta conexión de uso indica que un cambio en un elemento puede afectar a otro elemento que lo utiliza, pero no necesariamente a la inversa. Se representa con una flecha de línea discontinua orientada hacia el elemento que depende.
* Relación de Generalización: Representa la relación entre un elemento general y un caso específico de ese mismo elemento. La generalización significa que los objetos hijos pueden utilizarse en cualquier lugar donde aparezca el elemento padre, pero no al revés. Se representa con una flecha continua con punta vacía orientada hacia el elemento padre.
* Relación de Asociación: Esta relación entre un actor y un caso de uso denota la interacción entre ellos. Se llama asociación y se representa visualmente mediante una línea sólida que conecta al actor con el caso de uso correspondiente. Para diferenciar al actor que inicia el caso de uso de otros actores involucrados, la línea asociativa tiene una flecha en el extremo del caso de uso. Para los demás actores que participan, pero no lo inician, la asociación se representa con una línea sin flecha.

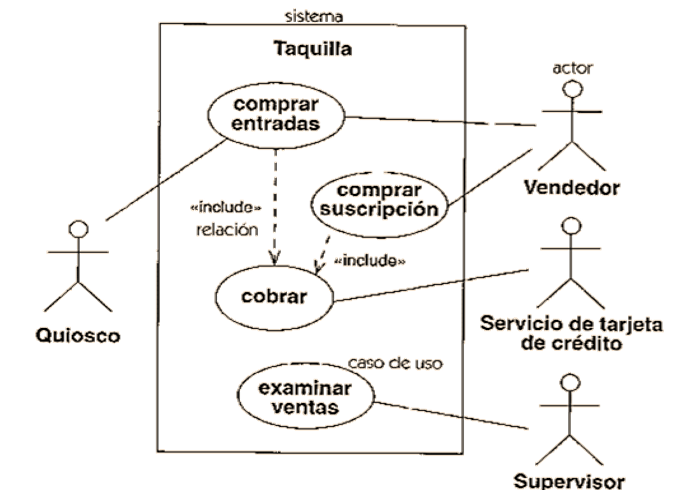
****

FIGURA 5. DIAGRAMA DE CASOS DE USO   
FUENTE. (Booch, Rambaugh, & Jacobson, 2004)

**2-3.2.2 DIAGRAMAS DE CLASES**

El diagrama de clases es una herramienta que se utiliza para representar la estructura estática de un sistema. Es importante destacar que se puede presentar a diferentes niveles de detalle, no siendo siempre necesario mostrarlo en su máxima complejidad. Este tipo de diagrama incluye la estructura de clases, que comprende sus atributos y métodos, y las relaciones que existen entre ellas, con detalles como cardinalidad, roles, herencia y pertenencia. La representación de una clase puede variar dependiendo de la profundidad de información requerida.

La simbología en un diagrama de clases se detalla de la siguiente manera:

* Clases: Representan conjuntos de objetos con estructuras similares, incluyendo su comportamiento y relaciones con otros elementos. Una clase se visualiza como un rectángulo sólido con tres secciones separadas por líneas horizontales. La primera sección contiene el nombre de la clase y propiedades generales. La segunda sección incluye los atributos de la clase, y la tercera lista las operaciones de la clase.
* Interfaces: Describen el comportamiento visible de una clase, componente o paquete y son un estereotipo de tipo. Se representan mediante un rectángulo que lleva la etiqueta "interface" y se dividen en compartimentos que contienen la lista de operaciones soportadas.
* Relación de Dependencia: Se establece entre clases cuando un cambio en un elemento independiente del modelo podría requerir cambios en un elemento dependiente.
* Relación de Asociación: Representan relaciones estructurales entre clases y objetos. Estas asociaciones indican una conexión entre elementos sin especificar la duración de vida relevante en relación con la dinámica general de los objetos instanciados.
* Relación de Generalización: Describe una relación jerárquica entre clases, donde una se considera superclase y la otra subclase. Puede haber varias clases que funcionen como subclases de una superclase.

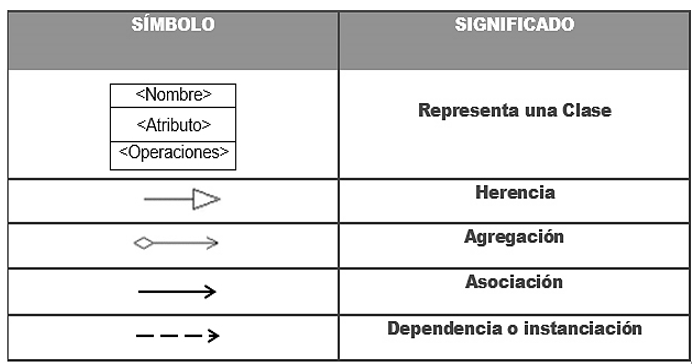


FIGURA 6. SIMBOLOGÍA DEL DIAGRAMA DE CLASES   
FUENTE. Elaboración Propia

**2-3.2.3 DIAGRAMAS DE SECUENCIA**

Según Booch, Rambaugh y Jacobson (2004), un diagrama de secuencia representa una interacción de manera gráfica y bidimensional. La dimensión vertical corresponde al eje del tiempo, donde el tiempo transcurre a través de la página. La dimensión horizontal muestra los roles que representan objetos individuales en la colaboración. Cada uno de estos roles se representa mediante una columna vertical que contiene un símbolo de inicio y una línea vertical, a menudo llamada "lifeline."

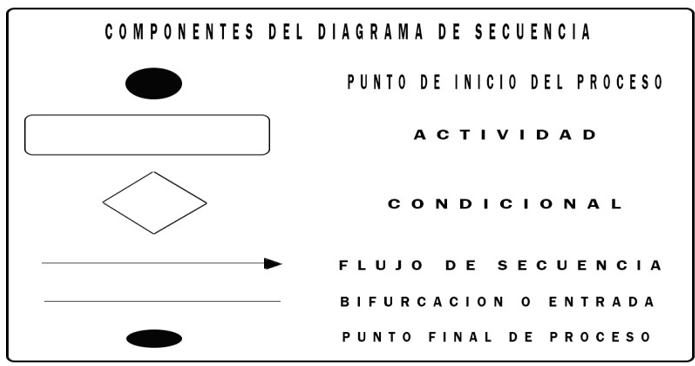


FIGURA 7. COMPONENTES DEL DIAGRAMA DE SECUENCIA   
FUENTE. ELABORACIÓN PROPIA

A continuación, se detallan los componentes clave de un diagrama de secuencia:

* Participantes: Representan los roles u objetos que interactúan en la secuencia. Cada participante se muestra como una línea vertical que refleja su actividad en el tiempo.
* Mensajes: Son las interacciones o comunicaciones entre los participantes. Estos mensajes se muestran mediante flechas que indican la dirección de la comunicación y su contenido.
* Activaciones: Indican el período durante el cual un participante está activo o involucrado en la interacción. Se representan mediante cajas verticales adyacentes a la línea del participante.
* Fragmentos de Interacción: Estos fragmentos ayudan a mostrar la lógica condicional o bucles en la secuencia de interacción. Pueden incluir condiciones como "if" o "else."
* Notas y Comentarios: Se utilizan para proporcionar información adicional o aclaraciones en el diagrama.

**2-3.2.4 DIAGRAMA DE ESTADOS**

Los diagramas de estados se emplean para crear una representación visual de las máquinas de estados finitos. Una máquina de estados finitos es un modelo de comportamiento que detalla las diferentes secuencias de estados que un objeto puede experimentar a lo largo de su vida en respuesta a eventos específicos, así como las acciones o respuestas asociadas a dichos eventos. Este enfoque permite una comprensión visual y estructurada de cómo un objeto o sistema se comporta en diferentes situaciones, lo que es fundamental para el diseño y la descripción de sistemas que implican transiciones de estado y eventos.

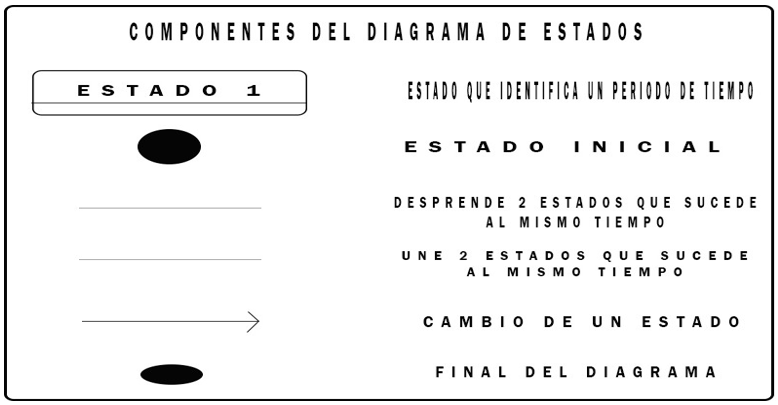
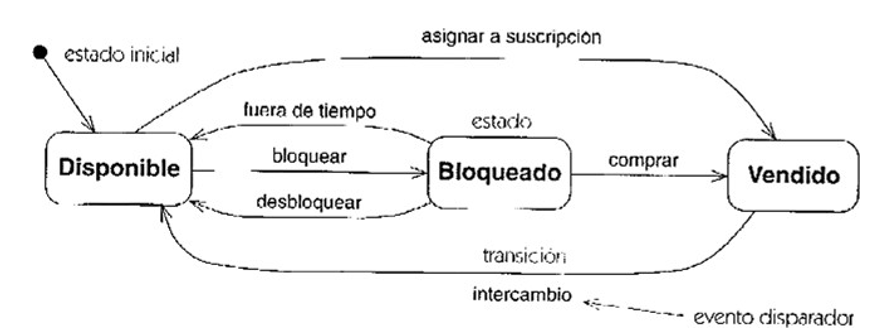


FIGURA 8. COMPONENTES DEL DIAGRAMA DE ESTADOS   
FUENTE. ELABORACION PROPIA

****

*FIGURA 9. DIAGRAMA DE ESTADOS   
FUENTE. (Booch, Rambaugh, & Jacobson, 2004)*

**2-3.2.5 DIAGRAMA DE ACTIVIDADES**

De acuerdo con la definición de Fowler (1999), un diagrama de actividades es una herramienta que se utiliza para representar un procedimiento lógico, un proceso de negocio o un flujo de trabajo. La interpretación de estos diagramas varía según su contexto en el proyecto. En las fases conceptuales, las actividades pueden considerarse como tareas que deben ser realizadas por una computadora o un ser humano.

**2-4** **TECNICA DE MODELADO DE OBJETOS (OMT)**

OMT (Object Modeling Technique) fue desarrollada por James Rumbaugh y Michael Blaha en 1991, mientras James lideraba un equipo de investigación en los laboratorios General Electric (Medina, 2005). La esencia del análisis y diseño orientado a objetos radica en la identificación y organización de conceptos dentro del dominio de la aplicación, independientemente de su posterior implementación en un lenguaje de programación orientado a objetos (Rumbaugh & Blaha, 1991).

Para construir sistemas complejos, los desarrolladores deben abstraer distintas perspectivas del sistema, crear modelos utilizando notaciones precisas, asegurarse de que estos modelos cumplan con los requisitos del sistema y, de manera gradual, agregar detalles para transformar los modelos en una implementación concreta (Rumbaugh & Blaha, 1991).

OMT divide el ciclo de vida del software en cuatro fases consecutivas:

* Análisis de Objetos: En esta etapa, se enfoca en comprender y modelar el problema dentro del dominio de la aplicación.
* Diseño del Sistema: Aquí se determina la arquitectura del sistema en términos de subsistemas.
* Diseño de Objetos: Esta fase refina y optimiza el análisis de objetos para prepararlo para la implementación.
* Implementación: En esta etapa, se procede a la codificación y pruebas de lo diseñado previamente.

OMT, como técnica, proporciona un enfoque estructurado y sistemático para el análisis y diseño de sistemas basados en objetos, lo que contribuye a la construcción eficiente de sistemas de software robustos y de alta calidad.

**2-4.1 ANALISIS DE OBJETOS**

En primer lugar, se inicia con la descripción del problema, enfocándose en la obtención de requisitos sin ambigüedades. En segundo lugar, se procede a la creación de los diagramas de objetos, donde se define la estructura de objetos y clases, así como las relaciones que los conectan.

Posteriormente, se desarrolla un modelo dinámico para describir los aspectos de control y evolución del sistema. Este modelo comprende un diagrama de estado para cada clase que presenta un comportamiento dinámico, además de un diagrama de flujo global de eventos, también conocido como diagrama de actividades.

A continuación, se elabora un modelo funcional que describe las funciones del sistema, especificando las entradas y salidas. En este punto, se suelen utilizar diagramas de flujo de datos orientados a objetos.

Por último, se realiza una verificación exhaustiva de todos los modelos creados, iterando para lograr un refinamiento en los tres aspectos mencionados.

Para clarificar, los modelos se dividen en tres categorías principales:

* Modelo de Objetos: Este es el modelo fundamental, ya que aquí se identifican las clases dentro del sistema junto con sus relaciones, atributos y operaciones, lo que representa la estructura estática del sistema.
* Modelo Dinámico: Este modelo se enfoca en los aspectos temporales del comportamiento y el control del sistema, detallando la secuencia de operaciones en el tiempo.
* Modelo Funcional: Aquí se representan los aspectos de transformación y función del sistema, describiendo cómo los datos se transforman a través de un diagrama de flujo.

Este enfoque estructurado garantiza que todos los aspectos clave del sistema se aborden de manera metódica y efectiva, lo que contribuye a un proceso de desarrollo de software más completo y sólido.

**2-4.1.1 MODELO DE OBJETOS**

Los pasos para construir el modelo de objetos son los siguientes:

* Identificación de Objetos y/o Clases: En esta etapa se identifican los objetos y clases que formarán parte del sistema, considerando sus roles y responsabilidades.
* Creación de un Diccionario de Datos: Se elabora un diccionario que contiene información detallada sobre los objetos, clases y conceptos del sistema, incluyendo sus atributos y relaciones.
* Identificación de Atributos y Conexiones para la Creación del Diagrama de Clases u Objetos: Aquí se definen los atributos de los objetos y se establecen las conexiones entre ellos para representar las relaciones entre clases. Con esta información, se puede crear el diagrama de clases u objetos que muestra la estructura estática del sistema.
* Realización de Iteraciones para el Refinamiento del Modelo: Es importante realizar iteraciones en todo el proceso para garantizar que el modelo sea preciso y completo. Esto implica revisar y ajustar el modelo según sea necesario.

**2-4.1.2 MODELO DINAMICO**

Los pasos para construir el modelo dinámico son los siguientes:

* Preparación de Escenarios de Secuencias Típicas de Iteración: En esta etapa inicial, se preparan escenarios que representan las secuencias típicas de interacción entre los objetos dentro del sistema. Estos escenarios ayudan a comprender cómo los objetos interactúan en situaciones específicas.
* Identificación de Sucesos que Actúan entre Objetos: Se identifican los eventos o sucesos que ocurren entre los objetos en cada escenario. Estos sucesos representan las interacciones y acciones que los objetos realizan en respuesta a eventos.
* Construcción de un Diagrama de Estado para Cada Objeto: Para cada objeto que tiene un comportamiento dinámico, se crea un diagrama de estado que representa sus posibles estados y las transiciones entre ellos. Estos diagramas de estado muestran cómo un objeto responde a eventos y cambia de estado en función de su comportamiento.
* Construcción del Diagrama Global de Flujo de Sucesos (Diagrama de Secuencia): Finalmente, se construye un diagrama de secuencia que representa la secuencia global de sucesos en el sistema. Este diagrama muestra cómo los objetos interactúan a lo largo del tiempo y cómo se comunican entre sí en función de los sucesos identificados en los escenarios.

En conjunto, estos pasos permiten modelar de manera efectiva el comportamiento dinámico del sistema, lo que incluye cómo los objetos interactúan, responden a eventos y evolucionan a lo largo del tiempo. La construcción de diagramas de estado y diagramas de secuencia es fundamental para representar estos aspectos del sistema de manera clara y comprensible.

**2-4.1.3 MODELO FUNCIONAL**

Los pasos para construir el modelo funcional son los siguientes:

* Identificación de los Valores de Entrada y de Salida: En esta etapa, se identifican claramente los datos y valores que ingresan al sistema (valores de entrada) y los resultados que el sistema produce (valores de salida). Esto ayuda a comprender la interacción del sistema con su entorno.
* Construcción de Diagramas de Flujo de Datos que Muestren las Dependencias Funcionales: Se crean diagramas de flujo de datos que representan gráficamente cómo fluyen los datos a través del sistema. Estos diagramas muestran las dependencias funcionales entre los elementos del sistema y cómo se transforman los datos a medida que atraviesan el sistema.
* Descripción de las Funciones: Se describe en detalle cómo se llevan a cabo las funciones o procesos dentro del sistema. Esto implica definir las operaciones y acciones que se realizan en respuesta a los datos de entrada, lo que incluye cálculos, transformaciones y otras tareas funcionales.
* Identificación de Restricciones: Se identifican y documentan cualquier restricción o limitación que afecte las funciones del sistema. Esto puede incluir restricciones de tiempo, recursos, regulaciones o cualquier otro factor que influya en el funcionamiento del sistema.

Estos pasos son fundamentales para comprender y representar de manera efectiva cómo el sistema realiza sus funciones y procesa los datos. El modelo funcional proporciona una visión clara de la lógica interna del sistema y cómo se logran sus objetivos funcionales.

**2-5 ARQUITECTURA DEL SOFTWARE**

La Arquitectura del Software, como lo define Clements en su trabajo en 1996, es un concepto fundamental en el diseño y desarrollo de sistemas informáticos. Se puede desglosar de la siguiente manera:

* Representación de alto nivel: Esto implica que la arquitectura del software ofrece una visión general y simplificada del sistema. En lugar de examinar cada detalle técnico, se enfoca en los componentes clave y cómo se relacionan entre sí para lograr los objetivos del sistema.
* Componentes fundamentales: La arquitectura del software identifica los elementos esenciales del sistema. Estos pueden ser módulos de software, componentes de hardware o cualquier otra parte vital del sistema. Estos componentes son como los bloques de construcción del sistema.
* Comportamiento desde la perspectiva del sistema en su conjunto: La arquitectura del software no solo se centra en las partes individuales del sistema, sino también en cómo interactúan y se comportan en conjunto. Considera cómo los componentes trabajan en armonía para lograr los objetivos del sistema.
* Interacciones y coordinación entre componentes: Esta parte es esencial. La arquitectura del software describe cómo los componentes se comunican y cooperan entre sí. Esto incluye la definición de interfaces, protocolos y flujos de datos que permiten que los componentes colaboren de manera efectiva.
* Cumplir la misión del sistema: En última instancia, la arquitectura del software se crea con un propósito en mente, que es asegurar que el sistema cumpla con su función o misión. Esto podría ser el procesamiento de datos, la gestión de una base de datos, la realización de operaciones comerciales o cualquier otro objetivo específico.

**2-5.1 PATRON MODELO VISTA CONTROLADOR**

El modelo-vista-controlador (MVC) es un patrón de arquitectura de software que separa claramente los datos y la lógica de la aplicación de la interfaz de usuario y la gestión de eventos. El patrón MVC propone la división de la aplicación en tres componentes fundamentales: el modelo, que se encarga de gestionar los datos y la lógica de negocio; la vista, que representa la interfaz de usuario, incluyendo las páginas HTML y la presentación de datos dinámicos; y el controlador, que recibe y gestiona los eventos de entrada provenientes de la vista. De esta manera, el MVC promueve una organización estructurada y modular de las aplicaciones, lo que facilita el mantenimiento y la escalabilidad.

**2-5.1.1 EL MODELO**

El modelo cumple diversas responsabilidades, que incluyen:

* Acceder a la capa de almacenamiento de datos, idealmente de manera independiente del sistema de almacenamiento subyacente.
* Definir las reglas de negocio que gobiernan la funcionalidad del sistema. Un ejemplo de regla podría ser: "Si el producto solicitado no se encuentra en el almacén, consultar el tiempo de entrega estándar del proveedor". Cabe destacar que la ubicación de estas reglas de negocio puede ser opcional, ya que también pueden residir en los controladores, directamente en las acciones correspondientes.
* Notificar a las vistas los cambios en los datos que puedan ser causados por agentes externos, en el caso de un modelo activo. Esto podría incluir situaciones como la actualización de datos a través de un archivo por lotes o la inserción desencadenada por un temporizador, entre otros. (Fernández Romero & Díaz González, 2012)

**2-5.1.2 LA VISTA**

Las vistas tienen las siguientes responsabilidades:

* Recibir los datos procesados por el controlador o el modelo y presentarlos al usuario.
* Mantener un registro de su controlador asociado.
* Proporcionar el servicio de "Actualización ()", que puede ser invocado por el controlador o el modelo cuando se trata de un modelo pasivo que no notifica cambios en los datos. Esto permite que la vista se actualice cuando se produzcan cambios en los datos, iniciados por otros agentes.

Un ejemplo de uso del patrón MVC con un modelo pasivo (que no notifica cambios en los datos) es la navegación web, donde las vistas responden a las interacciones del usuario, pero no detectan cambios en los datos del servidor. El Diagrama de Secuencia que se muestra en la FIGURA 15 ilustra la interrelación de los elementos del patrón. (Fernández Romero & Díaz Gonzáles, 2012)

**2-5.1.3 CONTROLADOR**

El controlador tiene las siguientes responsabilidades:

* Recibir los eventos de entrada del usuario, como clics de ratón o cambios en campos de texto.
* Contener reglas de gestión de eventos, en forma de instrucciones del tipo "Si ocurre el Evento Z, entonces ejecutar la Acción W". Estas acciones pueden implicar solicitudes al modelo o a las vistas. Por ejemplo, una de estas solicitudes a las vistas puede ser la llamada al método "Actualizar ()". O una solicitud al modelo podría ser "Obtener\_tiempo\_de\_entrega (nueva\_orden\_de\_venta)". (Fernández Romero & Díaz Gonzáles, 2012)

**2-5.2 ARQUITECTURA DE 3 CAPAS**

El Patrón de arquitectura por capas es una técnica ampliamente utilizada por arquitectos de software para estructurar sistemas de software complejos. Este enfoque conceptualiza un sistema como una serie de capas o niveles, similar a las capas de un pastel, donde cada capa se basa en la que está debajo de ella.

Los beneficios de trabajar con un enfoque por capas son:

* Permite comprender y trabajar en cada capa de manera aislada, sin necesidad de considerar las otras capas en ese momento.
* Las capas se pueden sustituir con implementaciones alternativas que ofrecen los mismos servicios básicos, lo que facilita la flexibilidad y adaptabilidad del sistema.
* Minimiza las dependencias entre las capas, lo que hace que el sistema sea más modular y fácil de mantener.
* Promueve la estandarización de servicios dentro de cada capa, lo que mejora la coherencia y la consistencia en todo el sistema.
* Una vez que se ha construido una capa, puede ser utilizada por múltiples servicios de nivel superior, lo que fomenta la reutilización de componentes.

**2-5.2.1 CAPA DE PRESENTACIÓN**

La interfaz de usuario se refiere a la forma en que los usuarios interactúan con el software. Puede ser tan básica como un menú de línea de comandos o tan sofisticada como una aplicación basada en formularios. Su función principal es mostrar información al usuario, interpretar los comandos que este introduce y llevar a cabo algunas validaciones sencillas de los datos ingresados.

**2-5.2.2 CAPA DE REGLAS DE NEGOCIO (EMPRESARIAL)**

También conocida como Lógica de Dominio, esta capa contiene la funcionalidad que implementa la aplicación. Engloba cálculos basados en la información proporcionada por el usuario y los datos almacenados, así como las validaciones necesarias. Controla la ejecución de la capa de acceso a datos y servicios externos. La lógica de la capa de negocios puede diseñarse para su uso directo por parte de los componentes de presentación o encapsularse como un servicio que se llama a través de una interfaz de servicios, coordinando así la comunicación con los clientes del servicio o invocando cualquier flujo o componente de negocio.

**2-5.2.3 CAPA DE DATOS**

Esta capa contiene la lógica de comunicación con otros sistemas encargados de llevar a cabo tareas dentro de la aplicación. Estos sistemas pueden ser monitores transaccionales, otras aplicaciones, sistemas de mensajería, entre otros. En el contexto de aplicaciones empresariales, esta capa suele estar representada por una base de datos, que se encarga del almacenamiento persistente de la información. Su función principal es abstraer por completo a las capas superiores, como la capa de negocio, del dialecto utilizado para comunicarse con los repositorios de datos, como PL/SQL, Transact-SQL, u otros.

**2-6 SEGURIDAD**

La Norma ISO 27001 es un estándar internacional de seguridad de la información que proporciona un marco para establecer, implementar, mantener y mejorar un sistema de gestión de seguridad de la información (SGSI). Algunos conceptos clave de la norma ISO 27001 que podrían estar aplicando incluyen:

* Política de Seguridad de la Información: Establecer una política que defina los objetivos de seguridad de la información y el compromiso de la alta dirección con la seguridad.
* Análisis de Riesgos: Identificar y evaluar los riesgos de seguridad de la información para determinar las medidas de seguridad adecuadas.
* Tratamiento de Riesgos: Implementar controles y medidas para mitigar los riesgos identificados.
* Documentación: Crear y mantener documentos que describan el SGSI, incluyendo la política, los procedimientos y los registros.
* Ciclo de Mejora Continua: Seguir un enfoque de mejora continua, incluyendo la revisión periódica de la eficacia de los controles y la política de seguridad de la información.
* Auditoría Interna: Realizar auditorías internas regulares para garantizar el cumplimiento de los controles y procedimientos de seguridad.
* Certificación ISO 27001: Buscar la certificación ISO 27001, que demuestra el compromiso de la organización con la seguridad de la información.
* Gestión de Incidentes de Seguridad: Implementar un proceso para la detección, gestión y respuesta a incidentes de seguridad de la información.

**2-6.1 PROPIEDADES DE UN SISTEMA DE INFORMACION SEGURO**

**2-6.1.1 INTEGRIDAD**

Este principio de seguridad informática garantiza la autenticidad y precisión de la información en cualquier momento que se solicite, asegurando que los datos no han sido alterados ni destruidos de manera no autorizada. Para prevenir y detectar posibles fallos en la integridad de los datos, el sistema debe estar equipado con mecanismos que permitan identificar y abordar los errores que puedan surgir.

**2-6.1.2 CONFIDENCIALIDAD**

La OCDE (Organización para la Cooperación y el Desarrollo Económico) define la confidencialidad, en sus Directrices para la Seguridad de los Sistemas de Información, como el hecho de que los datos o información estén disponibles únicamente para las personas, entidades o mecanismos autorizados, en los momentos autorizados y de una manera autorizada. Para garantizar la confidencialidad y prevenir posibles fallos, es fundamental diseñar un control de acceso al sistema que determine quién puede acceder, a qué parte del sistema, en qué momento y para qué tipo de operaciones.

**2-6.1.3 DISPONIBILIDAD**

La disponibilidad de la información garantiza que esta esté accesible para los usuarios autorizados cuando la necesiten, lo que implica que se pueda acceder al sistema de información en un periodo de tiempo considerado aceptable. La disponibilidad está estrechamente relacionada con la fiabilidad técnica de los componentes del sistema de información. Para asegurar la disponibilidad de la información, se deben implementar medidas de protección de los datos, así como establecer sistemas de copias de seguridad y mecanismos para la restauración de datos que puedan haber sido dañados o destruidos, ya sea de manera accidental o intencionada.

**2-6.2 MÉTODOS**

**2-6.2.1 CORTAFUEGOS (FIREWALL)**

Un cortafuegos o firewall es la parte de un sistema informático que se utiliza para impedir un acceso a través de la red no autorizado, pero permitiendo sin problemas todos los accesos autorizados. Los cortafuegos pueden ser software o hardware. (Gárcia-Cervigón Hurtado & Alegre Ramos, 2011)

**2-6.2.2 AUTENTICACIÓN Y AUTORIZACIÓN**

La autenticación y la autorización están estrechamente relacionadas debido a que la autenticación es el establecimiento de confirmación para dar paso a la autorización de acceso a las zonas restringidas, realizar acciones permitidas con su correspondiente nivel de privilegio, controlar el acceso a recursos protegidos, para prevenir ataques de escalada de privilegios.

**2-6.2.3 CIFRADO DE DATOS**

El cifrado de datos es uno de los métodos de protección más fiables en seguridad informática. Este proceso implica la transformación de los datos de tal manera que resulten incomprensibles para cualquier persona no autorizada a acceder a ellos. El cifrado puede llevarse a cabo mediante componentes lógicos o físicos, y es esencial para garantizar la confidencialidad de la información. Sin embargo, es importante tener en cuenta que el cifrado de datos puede ser intensivo en recursos computacionales, lo que significa que puede requerir un considerable poder de procesamiento y tiempo, dependiendo de la complejidad del cifrado empleado.

**2-6.2.4 SESIÓN**

Una sesión es una secuencia de interacciones entre un cliente y un servidor en la que se produce un intercambio de información. A través de una sesión, es posible realizar un seguimiento de la actividad de un usuario dentro de una aplicación. El inicio de una sesión ocurre cuando un usuario se conecta inicialmente a un sitio web, y su finalización puede estar vinculada a tres circunstancias diferentes.

En el contexto de las solicitudes RESTful (Representational State Transfer), una "sesión" puede ser una abstracción de estado que se mantiene a lo largo de varias solicitudes. A diferencia de las sesiones tradicionales en aplicaciones web más antiguas, que a menudo involucran el uso de cookies y tokens de sesión, las aplicaciones RESTful tienden a ser sin estado. Sin embargo, todavía es posible mantener una especie de sesión en una aplicación RESTful utilizando diferentes enfoques.

Una forma común de gestionar la "sesión" en una API RESTful es utilizar tokens de acceso, como tokens JWT (JSON Web Tokens). Un token JWT es un objeto JSON que se firma digitalmente y puede contener información sobre el usuario, sus permisos y cualquier otro dato necesario. El servidor emite un token al cliente después de una autenticación exitosa. Luego, el cliente incluye ese token en cada solicitud subsiguiente como una forma de autorización. El servidor puede verificar la firma del token y autorizar o denegar la solicitud en función de la información contenida en el token.

Esto permite que el servidor y el cliente mantengan un tipo de "sesión" o estado a lo largo de múltiples solicitudes, pero sin mantener una conexión continua o una información de estado en el servidor. La información de "sesión" se transporta con cada solicitud en el token JWT, lo que hace que la aplicación RESTful sea más escalable y sin estado.

En resumen, en una API RESTful, una "sesión" puede estar representada por un token de acceso, como un JWT, que se utiliza para mantener información de estado a lo largo de varias solicitudes sin la necesidad de mantener una sesión continua en el servidor.

**2-7** **MODELO DE ESTIMACION DE COSTOS COCOMO**

El Modelo de Estimación de Costos COCOMO (Constructive Cost Model) es un método ampliamente utilizado para estimar los costos, el tiempo y los recursos necesarios en el desarrollo de proyectos de software. Este modelo fue desarrollado por Barry W. Boehm en la década de 1980 y ha evolucionado en varias versiones a lo largo del tiempo. COCOMO se basa en una serie de fórmulas y parámetros que ayudan a los equipos de desarrollo de software a calcular los recursos requeridos y las estimaciones de costos con base en las características y el alcance del proyecto.

El modelo COCOMO se compone de tres niveles o modos de estimación, cada uno adaptado a diferentes niveles de detalle en la planificación del proyecto:

* COCOMO Básico: Este nivel de estimación se utiliza en las primeras etapas del proyecto, cuando se tienen pocos detalles específicos sobre el sistema. Se utiliza para realizar estimaciones de alto nivel y se basa principalmente en el tamaño estimado del software (en líneas de código o puntos de función).
* COCOMO Intermedio: En este nivel, se consideran factores adicionales, como la complejidad del proyecto, la experiencia del equipo y otras características específicas del proyecto. Estas estimaciones son más detalladas que las del COCOMO Básico y se utilizan cuando se cuenta con más información sobre el proyecto.
* COCOMO Detallado: Este es el nivel más detallado de estimación y se utiliza en etapas avanzadas del proyecto, cuando se cuenta con información detallada sobre los requerimientos, la arquitectura y otros aspectos del sistema. Permite realizar estimaciones muy precisas de costos y recursos.

Para aplicar el Modelo COCOMO, se utilizan ecuaciones específicas que toman en cuenta factores como el tamaño del software, la experiencia del equipo, la complejidad del proyecto y otros elementos clave. Además, COCOMO proporciona una serie de tablas de factores de ajuste que ayudan a adaptar las estimaciones a las condiciones particulares de cada proyecto.

**2-8 METRICAS DE CALIDAD DEL SOFTWARE**

Los desarrolladores de software más experimentados reconocen que la búsqueda de software de alta calidad es una de las metas más cruciales en la industria. La calidad del software es un concepto multifacético que abarca una amplia gama de factores, los cuales pueden variar según la naturaleza de las aplicaciones y las preferencias del cliente que las solicita (Pressman, 2010).

La calidad del software es una preocupación fundamental en la ingeniería de software y se refiere a la capacidad de un programa o sistema para cumplir con los requisitos y expectativas de los usuarios de manera confiable y eficaz. A medida que las aplicaciones informáticas se vuelven cada vez más integrales en nuestras vidas, la importancia de entregar software de alta calidad se vuelve aún más evidente.

La búsqueda de la calidad del software implica considerar aspectos como la funcionalidad, la confiabilidad, el rendimiento, la seguridad, la usabilidad y la mantenibilidad. Además, es esencial adaptar la calidad del software a las necesidades y prioridades específicas de cada proyecto y cliente, lo que agrega un nivel adicional de complejidad a esta búsqueda.

En resumen, la calidad del software es un objetivo central en el desarrollo de aplicaciones informáticas, y su logro exitoso requiere una cuidadosa consideración de diversos factores que pueden variar según el contexto y los intereses de los stakeholders involucrados en el proyecto.

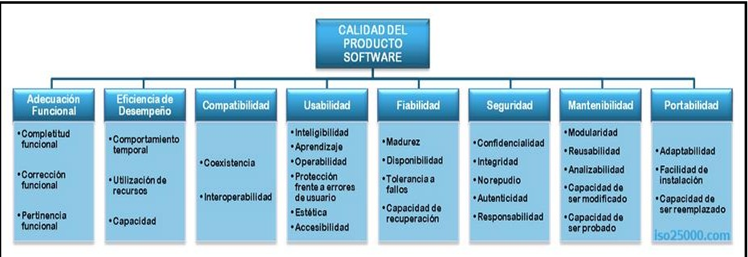
En resumen, el Modelo de Estimación de Costos COCOMO es una herramienta valiosa para los gerentes de proyectos y equipos de desarrollo de software, ya que les ayuda a planificar y presupuestar proyectos con mayor precisión. Ofrece una metodología sólida para estimar los recursos necesarios y los costos asociados a lo largo de las diferentes etapas del ciclo de vida del desarrollo de software.

**2-8.1** **EL ESTANDAR ISO/IEC 25010**

La ISO (International Organization for Standardization) y la IEC (International Electrotechnical Commission) constituyen un sistema especializado de estandarización a nivel mundial. En el ámbito de las tecnologías de la información, ISO e IEC han colaborado en la creación de un comité técnico conjunto conocido como ISO/IEC JTC, cuya responsabilidad principal es desarrollar estándares internacionales (Pérez Medina & Sánchez, 2012).

El modelo de calidad del producto definido por la ISO/IEC 25010 está compuesto por las ocho características de calidad que se presentan en la siguiente figura:

Este modelo se ha convertido en un marco fundamental para evaluar y garantizar la calidad de productos y sistemas de software, brindando una base sólida para la estandarización y la mejora continua en el desarrollo de software y tecnologías relacionadas.



**2-8.1.1 ADECUACIÓN FUNCIONAL**

La característica de "Funcionalidad" representa la capacidad del producto de software para ofrecer funciones que satisfacen las necesidades explícitas e implícitas, cuando el producto se utiliza en las condiciones especificadas. Esta característica se desglosa en las siguientes subcaracterísticas:

* Completitud Funcional: Esta subcaracterística evalúa en qué medida el conjunto de funcionalidades del producto abarca todas las tareas y objetivos especificados por el usuario.
* Corrección Funcional: La corrección funcional se refiere a la capacidad del producto o sistema para proporcionar resultados correctos con el nivel de precisión requerido.
* Pertinencia Funcional: Evalúa la capacidad del producto de software para ofrecer un conjunto adecuado de funciones que se ajusten a las tareas y objetivos específicos del usuario.

**2-8.1.2 EFICIENCIA DE DESEMPEÑO**

La característica de "Eficiencia del Desempeño" se refiere al rendimiento del software en relación a la cantidad de recursos utilizados en condiciones específicas. Esta característica se divide en las siguientes subcaracterísticas:

* Comportamiento Temporal: Evalúa los tiempos de respuesta y procesamiento, así como las tasas de transferencia de datos de un sistema cuando realiza sus funciones en condiciones predefinidas en comparación con un conjunto de pruebas de referencia (benchmark).
* Utilización de Recursos: Mide las cantidades y tipos de recursos que el software utiliza al llevar a cabo sus funciones en condiciones determinadas.
* Capacidad: Se refiere al grado en que los límites máximos de los parámetros de un producto o sistema de software cumplen con los requisitos especificados.

**2-8.1.3 COMPATIBILIDAD**

La característica de "Compatibilidad" se refiere a la capacidad de dos o más sistemas o componentes para intercambiar información y realizar sus funciones requeridas cuando comparten el mismo entorno hardware o software. Esta característica se divide en las siguientes subcaracterísticas:

* Coexistencia: Evalúa la capacidad del producto para operar en conjunto con otro software independiente en un entorno compartido, compartiendo recursos comunes sin afectar negativamente su funcionamiento.
* Interoperabilidad: Mide la capacidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada de manera efectiva.

**2-8.1.4 USABILIDAD**

Capacidad del producto software para ser entendido, aprendido, usado y resultar atractivo para el usuario, cuando se usa bajo determinadas condiciones. Esta característica se subdivide a su vez en las siguientes subcaracterísticas:

* Capacidad para reconocer su adecuación: capacidad del producto que permite al usuario entender si el software es adecuado para sus necesidades.
* Capacidad de aprendizaje: capacidad del producto que permite al usuario aprender su aplicación.
* Capacidad para ser usado: capacidad del producto que permite al usuario operarlo y controlarlo con facilidad.
* Protección contra errores de usuario: capacidad del sistema para proteger a los usuarios de hacer errores.
* Estética de la interfaz de usuario: capacidad de la interfaz de usuario de agradar y satisfacer la interacción con el usuario.
* Accesibilidad: capacidad del producto que permite que sea utilizado por usuarios con determinadas características y discapacidades.

**2-8.1.5 FIABILIDAD**

Capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados. Esta característica se subdivide a su vez en las siguientes subcaracterísticas:

* Madurez: capacidad del sistema para satisfacer las necesidades de fiabilidad en condiciones normales.
* Disponibilidad: capacidad del sistema o componente de estar operativo y accesible para su uso cuando se requiere.
* Tolerancia a fallos: capacidad del sistema o componente para operar según lo previsto en presencia de fallos hardware o software.
* Capacidad de recuperación: capacidad del producto software para recuperar los datos directamente afectados y reestablecer el estado deseado del sistema en caso de interrupción o fallo. (iso 25000, 2015)

**2-8.1.6 SEGURIDAD**

La característica de "Seguridad" se refiere a la capacidad del software para proteger la información y los datos de manera que personas o sistemas no autorizados no puedan acceder a ellos o modificarlos. Esta característica se desglosa en las siguientes subcaracterísticas:

* Confidencialidad: Capacidad del sistema para proteger los datos e información contra el acceso no autorizado, ya sea de manera accidental o intencionada.
* Integridad: Capacidad del sistema o componente para prevenir accesos o modificaciones no autorizadas en los datos o programas de computadora.
* No repudio: Capacidad del sistema para demostrar las acciones o eventos que han tenido lugar, de modo que dichas acciones o eventos no puedan ser negadas posteriormente.
* Responsabilidad: Capacidad del sistema para rastrear de manera inequívoca las acciones de una entidad.
* Autenticidad: Capacidad del sistema para demostrar la identidad de un sujeto o un recurso.

**2-8.1.7 MANTENIBILIDAD**

La característica de "Mantenibilidad" se refiere a la capacidad del producto software para ser modificado de manera efectiva y eficiente, ya sea para abordar necesidades evolutivas, correcciones de errores o mejoras. Esta característica se desglosa en las siguientes subcaracterísticas:

* Modularidad: Capacidad de un sistema o programa de computadora, compuesto por componentes discretos, que permite que un cambio en un componente tenga un impacto mínimo en los demás.
* Reusabilidad: Capacidad de un activo que permite que sea utilizado en más de un sistema de software o en la construcción de otros activos.
* Analizabilidad: Facilidad con la que se puede evaluar el impacto de un cambio específico en el software en el resto del sistema, diagnosticar deficiencias o causas de fallos en el software, o identificar las partes que requieren modificaciones.
* Capacidad para ser modificado: Capacidad del producto que permite que sea modificado de manera efectiva y eficiente sin introducir defectos o degradar el rendimiento.
* Capacidad para ser probado: Facilidad con la que se pueden establecer criterios de prueba para un sistema o componente y llevar a cabo pruebas para determinar si se cumplen dichos criterios.

**2-8.1.8 PORTABILIDAD**

La característica de "Portabilidad" se refiere a la capacidad del producto o componente de software para ser transferido de manera efectiva y eficiente entre diferentes entornos de hardware, software, operacionales o de uso. Esta característica se desglosa en las siguientes subcaracterísticas:

* Adaptabilidad: Capacidad del producto que le permite ser adaptado de manera efectiva y eficiente a diferentes entornos específicos de hardware, software, operaciones o uso.
* Capacidad para ser instalado: Facilidad con la que el producto se puede instalar y/o desinstalar con éxito en un entorno determinado.
* Capacidad para ser reemplazado: Capacidad del producto para ser utilizado en lugar de otro producto de software específico con el mismo propósito y en el mismo entorno.

**2-9 HERRAMIENTAS DE DESARROLLO WEB**

**2-9.1 SISTEMA GESTOR DE BASE DE DATOS MYSQL**

MySQL es un sistema de gestión de bases de datos (DBMS) ampliamente utilizado y uno de los más populares en el mundo. A continuación, se proporciona una descripción general de MySQL:

* Arquitectura:
  + Modelo Cliente-Servidor: MySQL utiliza una arquitectura cliente-servidor en la que múltiples clientes pueden conectarse al servidor MySQL para acceder y gestionar bases de datos.
* Características Principales:
  + Lenguaje SQL: MySQL es compatible con el lenguaje SQL (Structured Query Language), que permite realizar consultas y operaciones en la base de datos.
  + Soporte Transaccional: Ofrece compatibilidad con transacciones, lo que significa que puede realizar operaciones atómicas, consistentes, aisladas y duraderas (conocidas como propiedades ACID).Escalabilidad: MySQL es escalable y puede manejar bases de datos desde pequeñas hasta muy grandes.
  + Multiplataforma: Está disponible para varias plataformas, incluyendo Windows, Linux y macOS.
  + Almacenamiento de Datos: MySQL admite varios motores de almacenamiento, incluido el motor InnoDB, que se utiliza comúnmente para garantizar la integridad de los datos y proporcionar características de rendimiento.
  + Replicación: Permite la replicación de bases de datos, lo que facilita la creación de copias de bases de datos para fines de respaldo y redundancia.
  + Seguridad: Ofrece características de seguridad, como autenticación basada en contraseñas y autorización de usuarios.
* Usos Comunes:
  + MySQL se utiliza en una amplia gama de aplicaciones, desde sitios web dinámicos hasta sistemas de gestión de contenido (CMS), aplicaciones empresariales, sistemas de comercio electrónico y mucho más.
* Compatibilidad y Ecosistema:
  + MySQL es compatible con varios lenguajes de programación, incluyendo PHP, Python, Java, y más. También es compatible con herramientas de administración de bases de datos y marcos de desarrollo.
* Ediciones de MySQL:
  + MySQL ofrece varias ediciones, incluyendo la edición gratuita MySQL Community Edition y las ediciones comerciales de MySQL Enterprise para entornos empresariales.
* Comunidad y Soporte:
  + MySQL cuenta con una activa comunidad de usuarios y desarrolladores, lo que facilita la búsqueda de ayuda y recursos en línea.

**2-9.1.1 FUNCIONAMIENTO**

El funcionamiento de una base de datos MySQL implica varios componentes y procesos que trabajan juntos para gestionar y acceder a los datos almacenados. Aquí se describe de manera general cómo funciona una base de datos MySQL:

* Servidor MySQL (MySQL Server): El servidor MySQL es el núcleo de una base de datos MySQL. Es un programa que se ejecuta en un servidor y gestiona las solicitudes de los clientes.
* Clientes MySQL: Los clientes MySQL son aplicaciones que se conectan al servidor MySQL para enviar comandos y realizar operaciones en la base de datos. Pueden ser aplicaciones web, aplicaciones de escritorio, herramientas de línea de comandos, etc.
* Sistema de Almacenamiento: MySQL es modular y admite varios motores de almacenamiento, como InnoDB, MyISAM, y otros. Cada motor de almacenamiento tiene características y propiedades diferentes. El motor InnoDB es ampliamente utilizado por su soporte para transacciones y capacidades ACID.
* Conexiones: Los clientes se conectan al servidor MySQL a través de conexiones de red utilizando el protocolo MySQL, generalmente en el puerto 3306 por defecto.
* SQL (Structured Query Language): Los clientes envían comandos SQL al servidor MySQL. Estos comandos pueden incluir consultas para recuperar datos, instrucciones para actualizar datos, comandos de administración y más.
* Plan de Ejecución: Cuando se envía una consulta SQL al servidor, el servidor MySQL crea un plan de ejecución. Este plan determina cómo se debe acceder a los datos y cómo se deben realizar las operaciones solicitadas. El servidor optimiza el plan para mejorar el rendimiento.
* Acceso a Datos: El servidor MySQL accede a los datos almacenados en los archivos de la base de datos utilizando el motor de almacenamiento correspondiente. Los datos se leen, escriben y modifican según sea necesario.
* Buffer de Resultados: Cuando se recuperan datos, el servidor MySQL almacena temporalmente los resultados en un búfer de resultados antes de enviarlos al cliente. Esto permite manejar grandes conjuntos de resultados de manera eficiente.
* Envío de Resultados: Los resultados se envían al cliente MySQL, donde se pueden procesar y mostrar al usuario final. Los resultados pueden ser tablas de datos, filas de datos, números, etc.
* Gestión de Transacciones: MySQL admite transacciones para garantizar la integridad de los datos. Puede comenzar, confirmar o deshacer transacciones según sea necesario.
* Seguridad: MySQL proporciona mecanismos de seguridad, como autenticación y autorización, para proteger los datos y asegurarse de que solo usuarios autorizados puedan acceder a la base de datos.
* Gestión de la Base de Datos: Los administradores de bases de datos pueden realizar tareas de administración, como la creación y modificación de tablas, índices y copias de seguridad a través de comandos SQL o herramientas de administración.
* Optimización y Mantenimiento: Para garantizar un rendimiento óptimo, los administradores pueden realizar tareas de optimización, como la indexación de tablas y la monitorización del rendimiento.

**2-9.1.2 BASE DE DATOS RELACIONAL**

Una base de datos relacional, desde la perspectiva del usuario, puede describirse como un conjunto de tablas interconectadas que permiten el almacenamiento de información con el propósito de acceder y utilizar estos datos en el futuro. Estas bases de datos se fundamentan en el modelo de datos relacional para organizar y gestionar las tablas, el cual, a su vez, se apoya en conceptos derivados de la teoría de conjuntos para establecer relaciones entre los datos. Este enfoque estructurado y basado en tablas proporciona un medio efectivo para almacenar y recuperar información de manera lógica y coherente, siguiendo las formas normales de base de datos, como la Primera Forma Normal (1NF), la Segunda Forma Normal (2NF) y la Tercera Forma Normal (3NF). Estas formas normales son esenciales para garantizar la integridad y eficiencia de los datos, evitando la redundancia y asegurando una gestión óptima de la información.

**2-9.2 LENGUAJE DE PROGRAMACION PHP ("PREPROCESADOR DE HIPERTEXTO")**

PHP es un lenguaje de programación de alto nivel e interpretado con código abierto, que se incrusta directamente en páginas HTML y se ejecuta en el servidor. A diferencia de los lenguajes compilados, PHP es interpretado en tiempo real y se encuentra inmerso en el código HTML, lo que le otorga un alto rendimiento y versatilidad.

En contraste con lenguajes de script como JavaScript, PHP es un lenguaje de servidor web, lo que significa que se ejecuta en el servidor en lugar de en el navegador del cliente. Como resultado, solo se envía al navegador el resultado de su ejecución, lo que implica que el código fuente en PHP no es visible en el código fuente de la página web que se muestra en el navegador.

Desde la perspectiva del programador, PHP tiene una sintaxis similar a la del lenguaje de programación C. Se utiliza en tres contextos principales: primero, en el desarrollo de scripts del lado del servidor, que es su aplicación más tradicional; segundo, en la ejecución de scripts en la línea de comandos de sistemas operativos como Linux o Windows; y tercero, en el desarrollo de aplicaciones de interfaz gráfica utilizando PHP-GTK. Esto hace de PHP un lenguaje sumamente versátil y ampliamente utilizado en el desarrollo web y más allá.

**2-9.3 MARCO DE DESARROLLO YIIFRAMEWORK II**

Yii Framework 2: Yii Framework es un framework de desarrollo web de código abierto escrito en PHP. La versión 2 de Yii Framework (Yii2) es la segunda iteración de este popular framework y ha sido desarrollada para facilitar la creación de aplicaciones web de alta calidad de manera eficiente. Algunas de las características clave de Yii Framework 2 incluyen:

* Seguridad: Yii2 proporciona herramientas y prácticas de seguridad integradas, como protección contra ataques de inyección, autenticación y autorización.
* Eficiencia: Yii2 está diseñado para ser rápido y eficiente, lo que lo convierte en una excelente opción para aplicaciones web de alto rendimiento. También ofrece soporte para caché y optimización de consultas de bases de datos.
* Facilidad de uso: Yii2 utiliza un patrón de diseño MVC (Model-View-Controller) que facilita la organización y estructuración de aplicaciones web. También es altamente extensible y admite complementos y módulos.
* Compatibilidad con AJAX: Yii2 simplifica la integración de AJAX en aplicaciones web, lo que permite la creación de interfaces de usuario dinámicas y altamente interactivas.
* Generación de código automático: Yii2 ofrece herramientas de generación de código automático que aceleran el proceso de desarrollo al crear código repetitivo, como modelos y controladores.
* Comunidad activa: Yii Framework cuenta con una comunidad de desarrollo activa y una amplia base de usuarios, lo que significa que puedes encontrar una gran cantidad de recursos, extensiones y documentación en línea.

**2-9.4 CARACTERISTICAS GENERALES DE ANGULAR**

Angular es un popular framework de desarrollo de aplicaciones web y móviles que ofrece una amplia gama de características y funcionalidades. Aquí tienes algunas de las características generales de Angular:

* Arquitectura basada en componentes: Angular utiliza una arquitectura basada en componentes que permite dividir una aplicación en módulos reutilizables y fáciles de mantener. Cada componente encapsula su propia lógica y vista.
* Lenguaje TypeScript: Angular se basa en TypeScript, un superconjunto tipado de JavaScript. TypeScript agrega tipos estáticos y otras características de programación que ayudan a detectar errores en tiempo de compilación y a mejorar la calidad del código.
* Rutas y navegación: Angular proporciona un enrutador incorporado que facilita la creación de aplicaciones de una sola página (SPA). Esto permite la navegación sin recargar la página y una experiencia de usuario más fluida.
* Inyección de dependencias: Angular tiene un sistema de inyección de dependencias incorporado que facilita la gestión de componentes y la reutilización de código. Esto también promueve la modularidad y la testabilidad.
* Directivas personalizadas: Angular permite crear directivas personalizadas que extienden el comportamiento de las etiquetas HTML. Estas directivas son reutilizables y facilitan la creación de componentes personalizados.
* Manejo de formularios: Angular ofrece una amplia gama de opciones para trabajar con formularios, incluyendo formularios reactivos, formularios basados en plantillas y validación de formularios.
* Comunicación con el servidor: Angular facilita la comunicación con servidores a través de módulos como HttpClient. Esto permite realizar solicitudes HTTP, lo que es esencial para interactuar con servicios web.
* Pruebas unitarias y E2E: Angular incluye herramientas para realizar pruebas unitarias y pruebas end-to-end (E2E) de tus aplicaciones, lo que ayuda a garantizar la calidad y la robustez del código.
* Optimización de rendimiento: Angular incluye características de optimización de rendimiento, como la detección de cambios y la estrategia de detección de cambios OnPush, que ayudan a reducir el consumo de recursos y a mejorar la velocidad de las aplicaciones.
* Soporte de comunidad activa: Angular cuenta con una comunidad de desarrollo muy activa y un ecosistema de bibliotecas y extensiones. Esto significa que puedes encontrar una amplia gama de recursos y soluciones en línea.
* Compatibilidad con plataformas: Angular es versátil y puede utilizarse para desarrollar aplicaciones web y móviles. Puedes construir aplicaciones web progresivas (PWA), aplicaciones móviles nativas (usando Angular NativeScript o Ionic) y aplicaciones de escritorio (con proyectos como Electron).

**2-9.5 CARACTERISTICAS GENERALES DE IONIC**

Ionic es un popular marco de desarrollo de aplicaciones móviles híbridas que se basa en tecnologías web como HTML, CSS y JavaScript. Aquí tienes algunas de las características generales de Ionic:

* Desarrollo de aplicaciones multiplataforma: Ionic permite desarrollar aplicaciones móviles que funcionan en múltiples plataformas, incluyendo iOS, Android y la web. Puedes escribir una vez y ejecutar en cualquier lugar.
* Interfaz de usuario atractiva: Ionic ofrece una amplia variedad de componentes de interfaz de usuario y estilos prediseñados que facilitan la creación de aplicaciones con un aspecto moderno y atractivo. Estos componentes se asemejan a los de las aplicaciones móviles nativas.
* Tecnologías web estándar: Ionic se basa en tecnologías web estándar como HTML, CSS y JavaScript. Esto significa que puedes utilizar tus habilidades existentes para desarrollar aplicaciones móviles.
* Framework Angular: Ionic se integra estrechamente con el framework Angular de Google. Esto proporciona una estructura sólida y componentes reutilizables para el desarrollo de aplicaciones.
* Capacidades nativas: A través de plugins y API, Ionic permite acceder a las características nativas de los dispositivos, como la cámara, el GPS y las notificaciones push.
* Desarrollo rápido: Ionic incluye una interfaz de línea de comandos (CLI) que simplifica la creación de proyectos, la generación de componentes y la ejecución de pruebas.
* Soporte de PWA (Progressive Web App): Ionic facilita la creación de aplicaciones web progresivas que pueden instalarse en dispositivos y funcionar sin conexión. Esto mejora la experiencia del usuario y la accesibilidad de las aplicaciones.
* Comunidad activa y ecosistema de plugins: Ionic tiene una comunidad de desarrollo activa y una amplia gama de plugins y extensiones disponibles a través de Ionic Marketplace. Estos recursos permiten agregar funcionalidades adicionales a tus aplicaciones de forma sencilla.
* Herramientas de desarrollo en tiempo real: Ionic Live Reload y Ionic DevApp son herramientas que permiten una vista previa en tiempo real de las aplicaciones en dispositivos móviles durante el desarrollo.
* Personalización y theming: Ionic ofrece la capacidad de personalizar el aspecto y el estilo de tus aplicaciones con facilidad, lo que te permite adaptar la apariencia de tus aplicaciones según las necesidades de tu marca o proyecto.
* Facilidad de prueba: Ionic se integra con herramientas de prueba como Jasmine y Karma, lo que facilita la realización de pruebas unitarias y pruebas de extremo a extremo (E2E).
* Optimizado para rendimiento: Ionic está optimizado para el rendimiento de aplicaciones móviles, con enfoque en la velocidad y la eficiencia.

**2-9.6 CARACTERISTICAS GENERALES DE CAPACITOR**

Capacitor se enfoca en proporcionar a los desarrolladores de aplicaciones web las herramientas para crear aplicaciones móviles nativas utilizando tecnologías web estándar como HTML, CSS y JavaScript. Algunas de sus características clave incluyen:

* Multiplataforma: Permite el desarrollo de aplicaciones móviles para diversas plataformas, incluyendo iOS, Android y la web, utilizando un código base común.
* Acceso a API nativas: Proporciona una API unificada para acceder a las funciones nativas de dispositivos móviles, como la cámara y el GPS.
* Integración de plugins: Es altamente extensible a través de plugins que amplían las capacidades de la aplicación.
* Personalización de iconos y pantallas de inicio: Permite la personalización de la apariencia de la aplicación.
* Soporte para aplicaciones web progresivas (PWA): Facilita la creación de aplicaciones web instalables en dispositivos móviles.
* Desarrollo en tiempo real: Ofrece una función de "actualización en vivo" para ver cambios en tiempo real durante el desarrollo.
* Herramientas de desarrollo en línea de comandos (CLI): Facilita la gestión de proyectos y plugins.
* Facilita la implementación: Simplifica la distribución de aplicaciones en tiendas de aplicaciones.
* Amplia comunidad: Cuenta con una comunidad activa y documentación sólida.
* Soporte para desarrolladores web: Permite a los desarrolladores web aplicar sus conocimientos en la creación de aplicaciones móviles.
* Código abierto y gratuito: Es una solución de código abierto y gratuita, personalizable según las necesidades.

Capacitor es una herramienta versátil y accesible para la creación de aplicaciones móviles nativas a partir de tecnologías web.

**2-9.7 HTML (LENGUAJE DE MARCADO DE HIPERTEXTO)**

HTML, que significa "Hypertext Markup Language" (Lenguaje de Marcado de Hipertexto), es el estándar esencial para crear páginas web y aplicaciones web. Es un lenguaje de marcado que estructura y presenta contenido en la World Wide Web mediante etiquetas angulares, como <etiqueta>. HTML se compone de elementos y etiquetas, como encabezados, párrafos, imágenes, enlaces y formularios. Los elementos se organizan jerárquicamente, y muchos pueden tener atributos adicionales.

HTML permite la creación de enlaces, contenido multimedia, formularios interactivos y metadatos en las páginas web. Aunque se puede aplicar estilo básico, se recomienda usar CSS para un control más avanzado de la presentación. HTML es compatible con la mayoría de los navegadores modernos y ha evolucionado a lo largo del tiempo, siendo HTML 5 la versión más utilizada en la actualidad.

Combinado con CSS y JavaScript, HTML forma la base de la programación web moderna, proporcionando estructura, estilo y funcionalidad a las páginas web.

**2-9.8 CSS (STYLE)**

CSS (Cascading Style Sheets - Hojas de Estilo en Cascada) es un lenguaje de diseño ampliamente utilizado en desarrollo web para controlar la apariencia y presentación visual de páginas y aplicaciones web. Su concepto central es separar la estructura (HTML) del contenido del diseño y estilo, lo que ofrece flexibilidad, consistencia y mantenibilidad en el diseño web.

Sus características clave incluyen la separación de contenido y presentación, el uso de reglas y selectores para aplicar estilos a elementos HTML, propiedades y valores para describir aspectos visuales, una jerarquía y herencia que determina qué reglas aplicar, la reutilización de estilos en múltiples páginas, media queries para la creación de diseños responsivos, animaciones y transiciones, compatibilidad con navegadores, y su extensibilidad a través de preprocesadores de CSS como Sass y Less para escribir código CSS eficiente.

**2-9.8 JAVASCRIPT:**

JavaScript es un lenguaje de programación esencial en el desarrollo web, destinado a añadir interactividad y dinamismo a las páginas web. Se trata de un lenguaje de secuencias de comandos interpretado por el navegador del usuario, lo que significa que se ejecuta en su dispositivo.

Sus características principales incluyen la capacidad de crear interacciones en tiempo real en las páginas web, la manipulación del Document Object Model (DOM) para modificar el contenido y la estructura de una página sin necesidad de recargarla, la validación de formularios para mejorar la experiencia del usuario, la comunicación con el servidor a través de AJAX, la gestión de cookies y el almacenamiento local para mantener datos y el control de elementos multimedia como audio y video en la web.

**2-9.8.1 JQUERY:**

jQuery es una biblioteca de JavaScript de código abierto que simplifica la manipulación del DOM y la interacción con JavaScript. Proporciona una serie de funciones y métodos predefinidos para realizar tareas comunes de manera más sencilla y eficiente.

Características clave:

* Sintaxis simplificada: jQuery simplifica la selección y manipulación de elementos del DOM mediante una sintaxis más corta y legible.
* Gestión de eventos: Facilita la asignación y gestión de eventos, como clics, desplazamientos y cambios, en elementos HTML.
* Efectos y animaciones: jQuery proporciona una variedad de efectos y animaciones predefinidos que pueden aplicarse fácilmente a elementos HTML.
* AJAX simplificado: jQuery facilita la realización de solicitudes AJAX, lo que simplifica la comunicación con el servidor.
* Compatibilidad entre navegadores: jQuery maneja las diferencias de compatibilidad entre navegadores, lo que garantiza que el código funcione de manera consistente en múltiples navegadores.

**2-9.8.2 AJAX (ASYNCHRONOUS JAVASCRIPT AND XML):**

AJAX es una técnica de desarrollo web que utiliza JavaScript para realizar solicitudes asincrónicas al servidor sin recargar la página por completo. A pesar de llevar "XML" en su nombre, los datos pueden enviarse y recibirse en diferentes formatos, como JSON. Sus características clave incluyen solicitudes asincrónicas que no bloquean el resto del código, la posibilidad de cargar partes específicas de una página en lugar de toda la página, mejora de la interfaz de usuario para aplicaciones web interactivas, comunicación con el servidor sin recargar la página, amplia compatibilidad con navegadores y su papel esencial en aplicaciones web modernas.

En resumen, JavaScript es el lenguaje principal para agregar interactividad a las páginas web, jQuery simplifica la manipulación del DOM y la interacción con JavaScript, mientras que AJAX permite la comunicación asincrónica con el servidor, mejorando la experiencia del usuario en aplicaciones web modernas.

**2-9.8.3 JQUERY UI**

jQuery UI es una biblioteca de interfaz de usuario (UI) de código abierto basada en jQuery, diseñada para simplificar y mejorar la creación de interfaces de usuario interactivas en aplicaciones web. Ofrece componentes y widgets personalizables que facilitan la incorporación de interfaces de usuario atractivas y funcionales en sitios web y aplicaciones. Entre sus características destacadas se encuentran la interacción con el usuario, widgets como calendarios, menús desplegables y diálogos modales, personalización de temas, efectos visuales, eventos personalizados, compatibilidad con varios navegadores, documentación detallada, comunidad activa de desarrolladores y extensibilidad. jQuery UI mejora la usabilidad y la experiencia del usuario, lo que la hace una herramienta valiosa en el desarrollo web.

**2-9.9 BOOTSTRAP**

Bootstrap es un popular marco de diseño frontend de código abierto, inicialmente desarrollado por Twitter y ahora mantenido por la comunidad. Ofrece herramientas y estilos predefinidos para crear sitios web y aplicaciones atractivas y responsivas. Sus características principales incluyen un sistema de rejilla de 12 columnas para el diseño adaptable, una amplia gama de componentes CSS y JavaScript, opciones de personalización a través de temas y clases de estilo, compatibilidad con múltiples navegadores, documentación detallada y una comunidad activa de desarrolladores. Bootstrap se destaca por su facilidad de uso, lo que lo hace ideal para crear sitios web modernos y receptivos sin necesidad de escribir CSS y JavaScript personalizado. Para utilizarlo, simplemente se incorporan los archivos de Bootstrap en el código HTML y se aprovechan sus estilos y componentes.