ARQUITECTURAS DE SOFTWARE  GUIA DE APRENDIZAJE – ACTIVIDADES

**INTEGRANTES:**

ANYI ZUJEY GOMEZ CASANOVA

LAURA VALENTINA ARIZA ALEJO

WILLIAM STEBAN GONZALEZ CORTES

 YORDI ERIK NUÑEZ PINEDA

JULIAN DAVID FIERRO CASANOVA

CRISTIAN JEANPOOL BAHAMON

**TECNÓLOGO:**

ANÁLISIS Y DESARROLLO DE SOFTWARE

SERVICIO NACIONAL DE APRENDIZAJE SENA

NEIVA – HUILA

2023

Tabla de contenido

[1. Investigar sobre arquitecturas de software. 3](#_heading=h.gjdgxs)

[2. Su definición. 3](#_heading=h.30j0zll)

[3. Conveniencia de uso. 5](#_heading=h.1fob9te)

[4. Inconveniencia de uso. 7](#_heading=h.3znysh7)

[5. Imágenes de cada arquitectura. 9](#_heading=h.2et92p0)

[6. Sustentar qué arquitectura usaría en su proyecto y ¿Por qué? 12](#_heading=h.tyjcwt)

# Investigar sobre arquitecturas de software.

La arquitectura de software es la estructura básica que describe la estructura y composición de una aplicación de software. Estos marcos proporcionan un marco conceptual para construir y mantener sistemas, y establecen principios, pautas y restricciones que guían su desarrollo.

Hay muchos tipos de arquitectura informática, algunos de los más comunes son:

* Patrón cliente-servidor
* Patrón de capas
* Patrón master-slave
* Patrón modelo-vista-controlador (MVC)
* Patrón broker

# Su definición.

* Patrón cliente-servidor: El patrón cliente servidor es muy usado sobre todo en el diseño de webs y servicios online, y se basa en el concepto de la existencia de un servidor (que proporciona el servicio) y una serie de clientes, que piden al servidor y reciben una respuesta del mismo.
* Patrón de capas: En este patrón se subdivide la estructura del programa en un número de capas que representan una subtarea, cada una perteneciendo a un nivel de abstracción diferente. Cada capa está diseñada para proporcionar un servicio a la siguiente capa de mayor nivel.
* Patrón master-slave: Este patrón consiste en dos grupos, el primero es llamado el maestro (master) y el otro el grupo de esclavos (slaves). Los esclavos realizan la tarea propuesta por el maestro, computan los resultados y los envían de nuevo a este, quien los presenta, almacena o procesa.
* Patrón modelo-vista-controlador (MVC): Este famoso patrón, también conocido como patrón MVC, divide una aplicación interactiva en tres partes diferenciadas:

Modelo: Contiene la funcionalidad central y los datos.

Vista: Muestra la información al usuario, siempre es posible definir una o más vistas para una misma aplicación.

Controlador: Maneja la entrada del usuario. Esto se hace para separar las representaciones internas de la información de las formas en que se presenta y se acepta la información del usuario. De esta manera se desacopla los componentes y permite una reutilización eficiente del código.

* Patrón broker: Este patrón se utiliza para estructurar sistemas distribuidos con componentes desacoplados. Estos componentes pueden interactuar entre sí mediante la invocación de servicios remotos de forma que publicitan sus capacidades, solicitan un servicio y un componente, llamado broker, se encarga de coordinar la comunicación entre los componentes.

# Conveniencia de uso.

* Patrón cliente-servidor

Centralización - Todos los recursos disponibles se hallan centralizados en un único punto, lo que hace más sencillo su administración y más difícil para un cliente el uso de acciones dañinas.

Escalabilidad - Al funcionar de manera independiente es más sencillo mejorar cada pieza de forma separada o añadir nuevos nodos a la red creada.

Mantenimiento simplificado - Al funcionar de manera independiente y con separación clara de responsabilidades, es más sencillo mantener cada una de las piezas e incluso poder trasladar con sencillez el servidor a nuevo hardware/software si fuera necesario.

* Patrón de capas

Capacidad de testeo - Al tener cada capa por separado la implementación del testing es muy elevada respecto a otros patrones, ya que es posible realizar tests sobre cada capa de manera separada.

Facilidad de desarrollo - Debido a la alta distribución es mucho más sencillo coordinar un equipo para su desarrollo, ya que cada miembro o equipo tiene claro el objetivo de cada capa y sólo es necesario crear una interfaz clara de comunicación entre ellas.

* Patrón master-slave

Gestión centralizada - Este patrón destaca para el diseño de tareas multi-tarea ya que permite dividir las tareas en diferentes módulos que pueden ser ejecutados de forma independiente.

Control - Al partir todas las tareas de un único nodo orquestador, se mantiene mayor control ya que las tareas se ejecutan de manera independiente y reciben el contexto y su procesamiento final de un único punto de ejecución.

Escalabilidad - Al ser todas las tareas ejecutadas independientes las unas de las otras es posible escalar fácilmente el sistema de forma que añadir nuevos nodos esclavos se termina traduciendo en un más que posible aumento del rendimiento.

* Patrón modelo-vista-controlador (MVC)

Fácil colaboración - Al separar fuertemente responsabilidades es posible desarrollar muchas nuevas características sin necesidad muchas veces de tocar todas las piezas implicadas, de forma que la colaboración entre diferentes desarrolladores se amplifica notablemente.

Aplicaciones multi-vista - Al aislar las vistas del resto de la lógica de la aplicación es posible presentar las mismas funcionalidades en diferentes vistas que pueden dirigirse incluso a diversos dispositivos distintos.

* Patrón broker

Escalabilidad - Al igual que el patrón master-slave al ser todas las tareas ejecutadas independientes las unas de las otras es posible escalar fácilmente el sistema y sólo es necesario disponer de nuevos servidores que puedan procesar las peticiones del broker.

Rendimiento - Al ser un gran número de nodos conectados a lo que simplemente actúa como un puente con el cliente el rendimiento puede mantenerse siempre en altas escalas ya que es posible redistribuir si es necesario y procesar grandes cargas de peticiones en momentos de saturación

# Inconveniencia de uso.

* Patrón cliente-servidor

Disponibilidad - Al depender de un servidor para satisfacer las peticiones de los clientes se requiere que este esté activo y disponible en cada momento, una caída del servidor o incluso una congestión debido a la cantidad de peticiones de clientes resulta en una pérdida de funcionalidad absoluta del servicio.

Requisitos - Debido a que debe satisfacer un gran número de peticiones, el software y el hardware del servidor son determinantes a la hora de usar este patrón, y usar alguno deficiente impactará negativamente en el funcionamiento de nuestro sistema.

Distribución - El cliente no posee físicamente el producto ni tiene acceso a los recursos utilizados, cualquier caída del servidor implica que el cliente no pueda acceder a su trabajo en curso.

* Patrón de capas

Rendimiento - Debido a que cada funcionalidad está en una capa diferente normalmente este patrón sufre de menor rendimiento que otros debido a que cualquier petición debe de atravesar de forma individual cada capa de diseño.

Escalabilidad - Debido a que cada capa depende de la anterior y de la interfaz entre ellas, modificar un software que utilice este patrón puede ser tedioso y costoso ya que para modificar cualquiera de las capas es necesario hacer cambios en todas las anteriores, esto puede solucionarse subdividiendo las capas en módulos, pero de cualquier manera implica mayor esfuerzo respecto a otros patrones.

* Patrón master-slave

Implementación - Por como este sistema debe de funcionar, independientemente unas tareas de otras, no siempre es posible implementarlo para todos los proyectos, debido a que para ello debe de existir la posibilidad de dividirlo en módulos que puedan actuar aislados de la ejecución principal del programa.

Dependencia - En muchos casos una implementación de este patrón implica una fuerte dependencia con el sistema en el que se haya implementado que en muchas ocasiones hace difícil portarlo a otras máquinas distintas.

* Patrón modelo-vista-controlador (MVC)

Complejidad - Es un patrón complejo que requiere que los desarrolladores implicados tengan muy claros los conceptos de responsabilidad asignados a cada una de las tres partes implicadas.

Lento en ocasiones - En comparación a otros patrones desarrollar para este puede implicar tocar muchas y diversas piezas y tener que seguir un flujo de trabajo cerrado que a veces dificulta meter cambios que para cualquier otro desarrollo basado en otro patrón serían nimios.

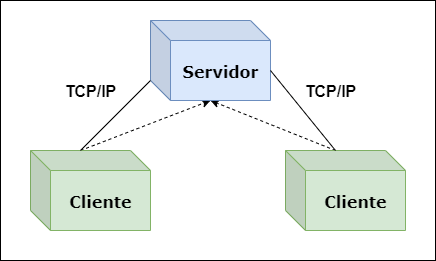
* Patrón broker

Coste - Evidentemente para implementar este sistema es necesario tener un gran número de servidores, que deben de ser mantenidos independientemente, con diferentes piezas de software distribuidas entre ellos.

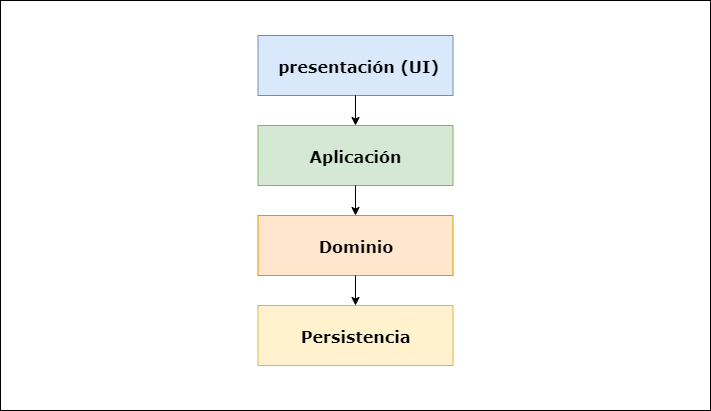
Mantenimiento - A diferencia de un sistema centralizado, este sistema distribuido implica que cualquier falla debe de poder deberse a cualquier pieza, igualmente realizar tareas de mantenimiento en la máquina broker, por ejemplo, puede implicar tiempos sin disponibilidad de servicio.

# Imágenes de cada arquitectura.

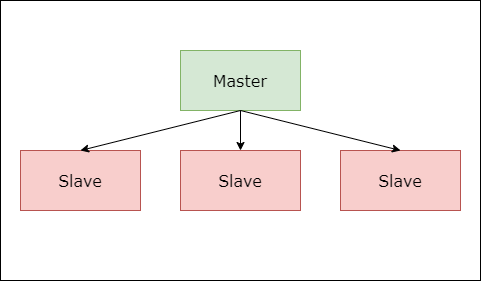
* Patrón cliente-servidor



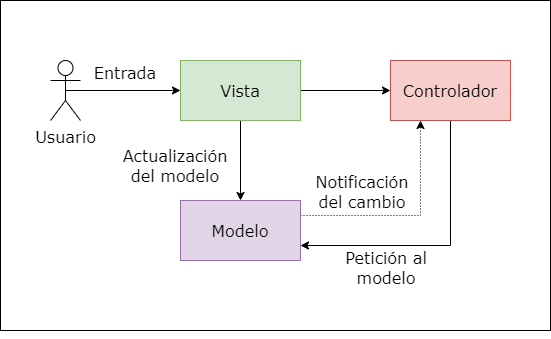
* Patrón de capas.



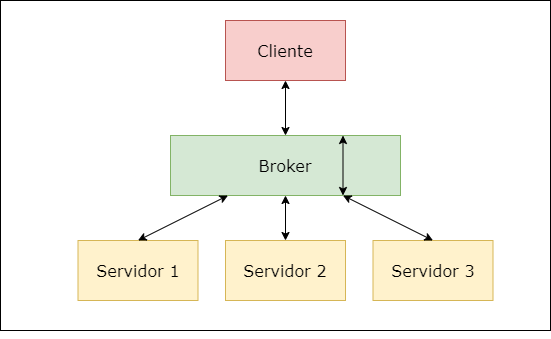
* Patrón master-slave



* Patrón modelo-vista-controlador (MVC)



* Patrón broker.



# Sustentar qué arquitectura usaría en su proyecto y ¿Por qué?

Para nuestro proyecto de Inventario de Bodega, utilizaremos la arquitectura de software de Patrón de Capas debido a su estructura modular y escalable. En este tipo de proyecto, es común tener una interfaz de usuario para realizar operaciones de entrada y consulta de inventario, una lógica de negocio que maneje las reglas y cálculos relacionados con el inventario, y una capa de acceso a datos para interactuar con la base de datos donde se almacenan los detalles de los productos y su disponibilidad.