**UNIVERSIDAD LA SALLE**

**CARRERA DE INGENIERÍA DE SISTEMAS**

****

**AGRO VOLTAICO AUTOMATIZADO PARA GENERAR ENERGÍA Y CUIDAR LOS CULTIVOS**

**Equipo de desarrollo:**  Luis Ángel Vertiz Blanco Tarifa

Investigación

Arquitectura de software

Lenguaje Audiovisual

La Paz - Bolivia

2023

**Índice de Contenido**

1. Introducción.

La arquitectura es aquella disciplina la cual esta especificada en hacer la planificación, diseño y construcción de espacios en general siendo implementada en muchos ámbitos, como la arquitectura de construcciones, pero desde hace ya muchos años se viene implementando en la construcción de software.

Apareciendo en los años 60 con respecto al software se referían a la planificación de modelos, patrones y abstracción teórica de los sistemas a ser implementados.

Al momento de hacer software complejo se implementan previamente a la codificación lo cual nos permite entender como encajara cada parte del software en el producto final.

1. Objetivos
   1. Objetivo General

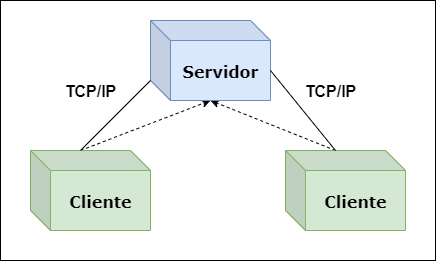
* Explicar acerca de las distintas arquitecturas de software.
  1. Objetivo Especifico
* Averiguar las ventajas de cada patrón.
* Conocer su implementación.

1. Marco Teórico.

4.1. Importancia

* Coste. Al implementar arquitectura de software a algún proyecto, estamos viendo a futuro la complejidad del mismo y cuanto de capital humano estamos dispuestos a gastar por el mismo.
* Tiempo. Al preguntarnos sobre los tiempos de entrega también estamos tocando el tema de costos y cuanto tiempo dispoemos para la fecha de entrega.
* Cantidad de Usuarios. A la hora de hacer el software la pregunta en general es la cantidad de los usuarios y a quien esta dirigido el software, gracias a estas preguntas se determina que tipo de arquitectura implementar.
* Nivel de aislamiento. Control sobre el producto en entornos aislados donde vemos si el producto necesita integrase con los usuarios o integrase con terceros .

4.2. Cliente-Servidor.



Es un tipo de desarrollo de software con labores que se racionan entre los recursos en donde el cliente hace solicitudes al programa y este debe responder a dicha solicitud.

En este patrón el cliente tiene un papel muy activo en el sistema de modo que comunica las acciones permitidas del mismo esperando una repuesta, las ventajas que nos brinda son las sgts.

* Centralización.Todos los recursos están centralizados en un solo punto lo cual hace mas sencilla la administración y evitar que el cliente haga acciones dañinas.
* Escalabilidad. Funciona de manera independiente lo cual mejora las piezas del sistema en forma separada.
* Mantenimiento. Al ser independiente y separar las responsabilidades , se mantiene eficientemente.

4.3. Arquitectura monolítica.



La arquitectura monolítica es un modelo tradicional de software donde todo se encuentra en una unidad unificada siendo esta autónoma e independiente a otras aplicaciones.

En simples palabras es una arquitectura de red informática grande y única, para hacerle cambios generalmente se actualiza todo el código desde la base y se implementa la versión actualizada.

Ventajas.

* Implementación Sencilla: archivo único.
* Desarrollo: aplicación única con código sencillo.
* Rendimiento: Todo esta centralizado y una API realiza la misma función.
* Pruebas simplificadas. Velocidad al hacer pruebas.
* Depuración: es más fácil rastrear las solicitudes y localizar incidencias.

Desventajas.

* Velocidad de desarrollo lento.
* No es escalable.
* Si hay un error en algún modulo, este afectara a todo el sistema.
* No es adaptable a cualquier cambio del lenguaje en el que este hecho.
* Un pequeño cambio afecta a cambiar todo el sistema por completo.

4.4. Microservicios.

Diagrama

Descripción generada automáticamente

Los microservicios son una metodología arquitectónica que se centra en servicios independientes con lógica empresarial y bases de datos específicas. Cada servicio se implementa, prueba, actualiza y escala de forma independiente. Aunque no reducen la complejidad, los microservicios la hacen más visible y manejable al descomponer tareas en procesos más pequeños e independientes que contribuyen al conjunto global. En esencia, desacoplan intereses específicos del dominio empresarial en unidades de código separadas, permitiendo un desarrollo más ágil y eficiente.

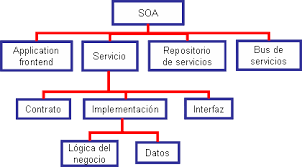
Ventajas.

* Promueve la agilidad en el trabajo por equipos.
* Es escalable de forma que se puede implementar fácilmente instancias al clúster y alivianar la presión.
* Implementación continua lo cual especifica las actualizaciones cada día de forma rápida y eficaz.
* Facilidad al mantener y probar.
* Implementación independiente por ser microservicios individuales.
* Flexibilidad donde los equipos eligen con libertad la herramienta que quieran implementar en el proyecto.
* Equipos satisfechos.

Desventajas.

* Desarrollo descontrolado
* Cortes exponenciales en la infraestructura.
* Sobrecarga organizativa
* Desafíos en la depuración
* Falta de estandarización.

4.5. Arquitectura Orientada a Servicios (SOA).



La Arquitectura Orientada a Servicios (SOA, por sus siglas en inglés, Service-Oriented Architecture) es un enfoque arquitectónico que organiza un sistema como un conjunto de servicios interconectados. Cada servicio realiza una función específica y se comunica con otros servicios a través de estándares bien definidos, como servicios web. SOA promueve la reutilización de servicios y la interoperabilidad entre sistemas.

Ventajas.

* Reutilización de Servicios: Los servicios pueden ser utilizados en múltiples aplicaciones, fomentando la reutilización de funcionalidades.
* Interoperabilidad: Facilita la comunicación entre sistemas heterogéneos a través de estándares, como XML y servicios web.
* Flexibilidad y Agilidad: Permite la rápida adaptación a cambios en los requisitos comerciales mediante la modificación o introducción de nuevos servicios.
* Escalabilidad: La arquitectura SOA permite escalar servicios individualmente, lo que mejora la escalabilidad del sistema en su conjunto.
* Facilita la Integración: Simplifica la integración de aplicaciones existentes y la construcción de nuevas aplicaciones a partir de servicios existentes.

Desventajas.

* Complejidad: La implementación de SOA puede ser compleja, especialmente en entornos grandes, debido a la necesidad de coordinar y gestionar múltiples servicios.
* Costos Iniciales: Los costos iniciales de implementación y formación pueden ser significativos.
* Gestión de la Transacción: Gestionar transacciones distribuidas puede ser complicado y requerir un diseño cuidadoso.
* Rendimiento: En algunas implementaciones, el rendimiento puede verse afectado debido a la sobrecarga asociada con la comunicación entre servicios.
* Dependencia de Estándares: La dependencia de estándares puede limitar la flexibilidad y la elección de tecnologías.

4.6. Arquitectura sin Servidor (Serverless).

Diagrama, Esquemático

Descripción generada automáticamente

La arquitectura sin servidor es un modelo de desarrollo de software en el cual el proveedor de servicios en la nube es responsable de la infraestructura subyacente. Los desarrolladores escriben funciones que se ejecutan en entornos de ejecución gestionados por el proveedor de la nube, sin preocuparse por la administración de servidores.

Ventajas.

* Escalabilidad Automática: La infraestructura se escala automáticamente según la demanda, lo que permite manejar picos de tráfico sin intervención manual.
* Costos Basados en el Uso: Se paga solo por el tiempo de ejecución de las funciones, lo que puede ser más rentable que mantener servidores en funcionamiento constantemente.
* Enfoque en la Lógica del Negocio: Los desarrolladores pueden centrarse en escribir código y en la lógica del negocio sin preocuparse por la administración de servidores.
* Despliegue Rápido: Las funciones se pueden implementar rápidamente sin la necesidad de gestionar la infraestructura subyacente.
* Facilita la Integración: Puede integrarse fácilmente con otros servicios en la nube y herramientas, facilitando la creación de sistemas complejos.

Desventajas.

* Limitaciones en el Tiempo de Ejecución: Algunas plataformas imponen límites en el tiempo de ejecución de las funciones, lo que puede afectar tareas que requieren tiempo prolongado.
* Dificultad en Operaciones Locales: La falta de entornos locales para desarrollo y prueba puede dificultar la depuración y prueba de funciones antes de la implementación.
* Problemas de Latencia: Puede haber cierta latencia asociada con el inicio de instancias de funciones, especialmente si no se utilizan con frecuencia.
* Falta de Control sobre la Infraestructura: La falta de control directo sobre la infraestructura subyacente puede ser una limitación para aplicaciones que requieren configuraciones específicas.
* No Adecuado para Todas las Aplicaciones: No todas las aplicaciones son adecuadas para un modelo sin servidor. Aplicaciones con procesos continuos o necesidades específicas de infraestructura pueden encontrar limitaciones.

4.7. Arquitectura Basada en Eventos.

Interfaz de usuario gráfica

Descripción generada automáticamente

La Arquitectura Basada en Eventos es un enfoque arquitectónico donde los componentes del sistema se comunican a través de eventos. Un evento representa un cambio de estado o una acción significativa que ocurre en el sistema. Los componentes pueden ser productores de eventos (generan eventos) o consumidores de eventos (responden a eventos). Esta arquitectura promueve la desvinculación y la escalabilidad al permitir que los componentes reaccionen de manera independiente a los eventos que ocurren en el sistema.

Ventajas.

* Desacoplamiento: Los componentes no necesitan conocerse entre sí. La comunicación se realiza a través de eventos, lo que facilita la independencia y el cambio de componentes sin afectar a otros.
* Escalabilidad: Los sistemas basados en eventos son inherentemente escalables. Puedes agregar nuevos componentes que respondan a eventos sin afectar negativamente la arquitectura general.
* Manejo de Concurrencia: La arquitectura basada en eventos es efectiva para manejar situaciones concurrentes, ya que los eventos pueden ser manejados de manera asincrónica.
* Flexibilidad: Facilita la introducción de nuevos servicios o funcionalidades sin necesidad de modificar los componentes existentes.
* Resiliencia: Al estar desacoplados, si un componente falla, los demás pueden seguir operando sin interrupciones.

Desventajas.

* Complejidad de Diseño: Puede requerir un diseño más complejo para gestionar la propagación y manejo de eventos.
* Dificultad de Rastreo: En sistemas complejos, el rastreo y la depuración de eventos pueden ser desafiantes, especialmente en entornos distribuidos.
* Consistencia: Asegurar la consistencia en sistemas distribuidos basados en eventos puede ser más complicado que en arquitecturas monolíticas.
* Tiempo Real: Algunos sistemas que requieren respuestas en tiempo real pueden encontrar limitaciones debido a la asincronía inherente en la gestión de eventos.
* Complejidad Operativa: La gestión de eventos a lo largo del ciclo de vida del sistema puede introducir complejidades operativas.

4.8. Arquitectura Hexagonal (Puertos y Adaptadores).

Diagrama

Descripción generada automáticamente

La Arquitectura Hexagonal, también conocida como Puertos y Adaptadores, es un modelo arquitectónico que organiza una aplicación en capas concéntricas alrededor de su núcleo. El núcleo de la aplicación contiene la lógica empresarial y está aislado de los detalles técnicos y externos. La arquitectura utiliza puertos y adaptadores para gestionar las interacciones entre el núcleo y los componentes externos, como bases de datos, interfaces de usuario y servicios externos.

Ventajas.

* Desacoplamiento: Permite un alto nivel de desacoplamiento entre la lógica empresarial y los detalles de implementación técnica.
* Testabilidad: Facilita la prueba unitaria y la prueba de integración al separar claramente la lógica empresarial de la implementación técnica.
* Flexibilidad: Permite cambiar componentes externos sin afectar la lógica empresarial central, ya que los adaptadores pueden ser reemplazados fácilmente.
* Reusabilidad: Los componentes externos pueden ser reutilizados en diferentes contextos ya que están separados del núcleo de la aplicación.
* Mantenibilidad: La estructura clara facilita el mantenimiento y la evolución de la aplicación a lo largo del tiempo.

Desventajas.

* Complejidad Inicial: Puede introducir una complejidad adicional en comparación con enfoques más simples, especialmente en aplicaciones pequeñas.
* Curva de Aprendizaje: Puede requerir que el equipo de desarrollo adquiera conocimientos adicionales para comprender y aplicar correctamente esta arquitectura.
* Overhead de Abstracción: La introducción de capas y abstracciones puede tener un pequeño overhead que podría afectar el rendimiento en sistemas muy simples o de baja complejidad.
* Posible Sobreingeniería: En algunos casos, podría llevar a una sobreingeniería si se aplica innecesariamente a proyectos más simples.
* Necesidad de un Diseño Cuidadoso: Requiere un diseño cuidadoso y una comprensión clara de las dependencias entre los componentes para evitar problemas de implementación.

1. Conclusiones.

En conclusión, no existe una arquitectura perfecta para todos los sistemas en general pero de todas las ya existentes están las que mas se adecuan a los requerimientos necesarios del software que se quiere crear.