UNIVERSIDAD POLITÉCNICA DE TECÁMAC.

Materia: Cliente/Servidor.

Docente: Emmanuel Torres Servín.

Trabajo: Investigación Unidad 2.

Team:

- Natali Joselin Alemán Perez.
- Luz Alexia Fuentes Cortes.
- Marco Joel Ángel Velasco.
- José Eduardo de la Cruz Medina.

Grupo: 1523IS.

Fecha de entrega: 13 de Marzo del 2023.

Índice:

1.Estartegias de reparto de complejidad. 2

1.1 Conceptos de presentación distribuida, presentación remota, proceso distribuido y bases de datos distribuidas 2

1.2 Conceptos de lógica de acceso y negocio a datos. 3

1.3 Proceso de diseño de lógica de acceso a datos y lógica de ampliación. 3

1.4 Proceso de desarrollo de lógica de acceso a datos, lógica de negocio o lógica de ampliación. 4

2. Modelos Multinivel 5

2.1 Concepto de nivel vinculado a programación web. 5

2.2 Proceso de planificación en dos niveles. 5

2.3 Proceso de planificación en tres niveles. 5

2.4 proceso de planificación multiniveles. 6

2.5 Problemas de actualización y mantenimiento de aplicaciones multinivel. 6

3. Modelo Vista / Controlador. 8

3.1 Conceptos de Modelo, control, y vista en las arquitecturas Cliente / Servidor. 8

3.2 Concepto de modelo vista controlador (MVC) en las arquitecturas Cliente / Servidor. 8

3.3 Proceso de flujo de control a partir del MVC,) en las arquitecturas Cliente / Servidor. 9

3.4 Desarrollo de software a partir del MVC, en las arquitecturas Cliente / Servidor. 10

4. Sockets 12

4.1 Concepto de comunicación orientada a conexión e interfaz de programación de aplicaciones API. 12

4.2 Proceso de comunicación y configuración orientada a conexión e interfaz de programación de aplicaciones 13

4.3 Concepto de sockets 13

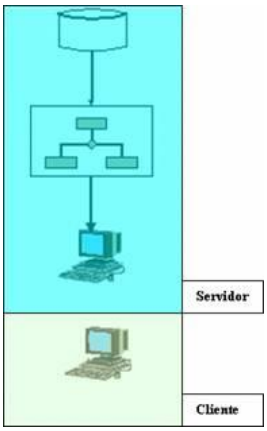
4.4 Proceso del uso de sockets en aplicaciones Cliente/Servidor 14

Link de repositorio GitHub: 15

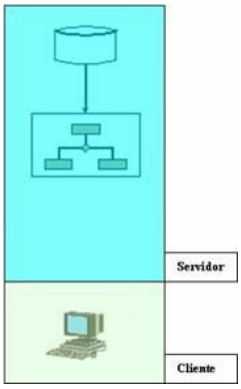
1.Estartegias de reparto de complejidad.

1.1 Conceptos de presentación distribuida, presentación remota, proceso distribuido y bases de datos distribuidas.

Presentación distribuida: La capa de presentación se encuentra distribuida entre el cliente y el servidor, de manera que en el cliente se modifica o adapta la presentación que ofrece el servidor. Este tipo de sistemas tienen un difícil mantenimiento.



Presentación remota: La capa de presentación de datos, se ejecuta en el cliente totalmente. En ella se realizan las validaciones de los datos de entrada, el formateo de los de salida, etc. La lógica de negocio y el acceso a la base de datos se aloja en el servidor.



Proceso distribuido: En este modelo, la capa que implementa la lógica de negocio se encuentra dividida entre el cliente y el servidor. El acceso a la base de datos se encuentra en el servidor y la capa de presentación en el cliente.

Base de datos distribuidas: Las capas de negocio y de presentación se ejecutan completamente en el cliente, mientras que la base de datos está distribuida entre el cliente y el servidor. Se requieren de mecanismos para asegurar la coherencia en los datos.

Base de datos remota: Las capas de negocio y de presentación se ejecutan completamente en el cliente, mientras que la base de datos está completamente en el servidor.

1.2 Conceptos de lógica de acceso y negocio a datos.

En informática y ciencias de la computación, en particular en análisis y diseño orientado a objetos, el término lógica de negocio es la parte de un sistema que se encarga de codificar las reglas de negocio del mundo real que determinan cómo la información puede ser creada, almacenada y cambiada. En programación es una de las capas del modelo MVC Modelo vista controlador separando así la complejidad del desarrollo en capas independientes.

Son rutinas que realizan entradas de datos, consultas a los datos, generación de informes y más específicamente todo el procesamiento que se realiza detrás de la aplicación visible para el usuario.

En el contexto de la orientación a objetos, la lógica del negocio es tomada como aquella funcionalidad ofrecida por el software. El software se comunica de manera amigable con el usuario a partir de la interfaz, pero el procesamiento de los datos capturados como entrada y la posterior entrega de resultados al usuario por medio de la interfaz, es conocido como la Lógica de Negocio.

1.3 Proceso de diseño de lógica de acceso a datos y lógica de ampliación.

El proceso de diseño de bases de datos consiste en definir la estructura lógica y física de una o más bases de datos para responder a las necesidades de los usuarios con respecto a la información y para un conjunto concreto de aplicaciones.

Mediante un proceso de diseño de bases de datos, se pueden decidir las tablas y relaciones que debe tener una base de datos determinada, los atributos de las diferentes tablas, las claves primarias y las claves foráneas que se deben declarar en cada tabla, etc. Todas estas tareas forman parte del proceso de diseño de bases de datos. Para poder tomar estas decisiones de la manera más correcta posible, hay que tener en cuenta las necesidades de información de los usuarios en relación con un conjunto concreto de aplicaciones.

Los requisitos que debe cumplir un sistema de información y la complejidad de la información que se presenta en él provocan que el diseño de una base de datos sea

un proceso complicado. Para simplificar este proceso, es muy recomendable utilizar la estrategia de “divide y vencerás” (divide and conquer).

Si aplicamos este concepto, obtenemos las diferentes etapas del diseño de bases de datos. Estas etapas son secuenciales y el resultado de cada una sirve de punto de partida de la etapa siguiente. El resultado de la última etapa será el diseño final de nuestra base de datos. De este modo, un proceso de una cierta complejidad se descompone en diferentes procesos de menor complejidad. La figura 1 muestra las distintas etapas del diseño de bases de datos.

En primer lugar, tenemos la recogida y análisis de requisitos. Esta etapa debe permitir obtener los requisitos y las restricciones de los datos del problema. Para obtener esta información será necesario mantener conversaciones con los diferentes usuarios de la futura base de datos y de las aplicaciones que estén relacionadas con ésta. Sólo si se cruzan los requisitos de los diferentes perfiles de usuarios será posible establecer un marco completo de requisitos y las restricciones de los datos relacionados con la futura base de datos.

1.4 Proceso de desarrollo de lógica de acceso a datos, lógica de negocio o lógica de ampliación.

La programación por capas es un modelo de desarrollo software en el que el objetivo primordial es la separación (desacoplamiento) de las partes que componen un sistema software o también una arquitectura cliente-servidor: lógica de negocios, capa de presentación y capa de datos. De esta forma, por ejemplo, es sencillo y mantenible crear diferentes interfaces sobre un mismo sistema sin requerirse cambio alguno en la capa de datos o lógica.

La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, solo afectará al nivel requerido sin tener que revisar entre el código fuente de otros módulos, dado que se habrá reducido el Acoplamiento informático hasta una interfaz de paso de mensajes.

Además, permite distribuir el trabajo de creación de una aplicación por niveles; de este modo, cada grupo de trabajo está totalmente abstraído del resto de niveles, de forma que basta con conocer la API que existe entre niveles.

En el diseño de sistemas informáticos actual se suelen usar las arquitecturas multinivel o programación por capas. En dichas arquitecturas a cada nivel se le confía una misión simple, lo que permite el diseño de arquitecturas escalables.

2. Modelos Multinivel

2.1 Concepto de nivel vinculado a programación web.

En la programación web, el concepto de nivel se refiere a la estructura jerárquica de los componentes de un sitio web o aplicación web. Cada nivel se encarga de una tarea específica y diseñado para interactuar con los niveles superiores e inferiores. los niveles en la programación web son típicamente 3:

El nivel de presentación, el nivel de aplicación y el nivel de datos.

El nivel de presentación también conocido como el **nivel de interfaz de usuario**, es el nivel más alto y se encargada de la presentación visual y la interacción donde el usuario con hoy el sitio web o aplicación. Este nivel utiliza lenguajes marcado de como HTML CCS y JavaScript.

El nivel de aplicación es el nivel intermedio y es responsable de procesar y manejar la lógica del negocio de la aplicación. Este nivel utiliza lenguajes de programación cómo PHP, Ruby, Python, etc.

El nivel de datos es el nivel más bajo y se encarga de almacenar y administrar los datos de la aplicación. Este nivel utiliza sistemas de bases de datos como MySQL, PostgreSQL, MongoDB, etc.

2.2 Proceso de planificación en dos niveles.

El proceso de planificación en los niveles implica la división de la aplicación en 2 niveles: el nivel de presentación y el nivel de datos.

El nivel de presentación se encarga de la interfaz de usuario y la presentación visual, mientras que el nivel de datos se encarga del almacenado y administración de los datos.

En este proceso de planificación, el nivel de presentación se diseña primero y luego se conecta al nivel de daros a través de un API. Este enfoque es adecuado para aplicaciones web simples que no tienen una lógica de negocio compleja.

2.3 Proceso de planificación en tres niveles.

El proceso de planificación en 3 niveles implica la división de la aplicación en 3 niveles: El nivel de presentación, el nivel de aplicación y el nivel de datos.

hoy en este proceso de planificación se desarrolla de forma independiente y se conecta a los niveles superior e inferior a través de interfaces de programación de aplicaciones (API).

El nivel de presentación se encarga de la interfaz de usuario y la presentación visual, el nivel de aplicación se encarga de la lógica de negocio de la aplicación, y el nivel de datos se encarga del almacenamiento y la administración de datos. este enfoque es adecuado para aplicaciones web más complejas que tienen una lógica de negocio compleja

2.4 proceso de planificación multiniveles.

El proceso de planificación de multiniveles implica la división de la aplicación en varios niveles , cada uno con una tarea especifica y claramente definida. En este proceso de planificación cada nivel se desarrolla de forma independiente y se conecta a los niveles superior e inferior a través de interfaces de interfaces de programación de aplicaciones (API).

Este enfoque es el adecuado para las aplicaciones web muy complejas que tienen múltiples componentes y una lógica de negocio muy compleja. El proceso de planificación de multiniveles permite una mayor escalabilidad, flexibilidad y modularidad de la aplicación, lo que facilita el mantenimiento y la actualización a largo plazo.

2.5 Problemas de actualización y mantenimiento de aplicaciones multinivel.

Uno de los mayores desafíos del desarrollo de aplicaciones multinivel es el mantenimiento y actualización de la aplicación a lo largo del tiempo. A medida que la aplicación crece y se vuelve más compleja, el mantenimiento y la actualización se vuelven cada vez más difíciles. Algunos de los problemas comunes asociados con la actualización y el mantenimiento de aplicaciones multinivel incluyen:

1. Coordinación entre los niveles: Con múltiples niveles, es importante que los cambios realizados en uno de los niveles no afecten negativamente a los otros niveles. Es importante coordinar los cambios para asegurarse de que todos los componentes de la aplicación funcionen correctamente.
2. Dificultad para rastrear errores: Con múltiples niveles, rastrear errores puede ser complicado, especialmente cuando un error afecta a varios niveles. Puede ser difícil identificar el origen del problema y solucionarlo.
3. Dependencia de terceros: Los componentes de cada nivel a menudo dependen de bibliotecas y herramientas de terceros. Si una biblioteca o herramienta cambia o se actualiza, puede afectar a la funcionalidad de la aplicación en su conjunto.
4. Actualización de datos: Las actualizaciones de datos pueden ser un desafío en aplicaciones multinivel. Si se cambia la estructura de la base de datos, puede afectar a otros niveles de la aplicación.

5. Mantenimiento a largo plazo: Con el tiempo, los componentes individuales de la aplicación pueden volverse obsoletos o desactualizados. Es importante realizar actualizaciones regulares para asegurarse de que la aplicación se mantenga actualizada y funcione correctamente.

Para abordar estos problemas, es importante tener un proceso sólido de mantenimiento y actualización en su lugar y asegurarse de que cada nivel de la aplicación esté bien documentado. Además, es importante trabajar con desarrolladores experimentados y capacitados para garantizar que la aplicación se mantenga actualizada y funcione correctamente a lo largo del tiempo.

3. Modelo Vista / Controlador.

3.1 Conceptos de Modelo, control, y vista en las arquitecturas Cliente / Servidor.

Modelo: Es la representación de los datos y la lógica de negocio en una aplicación. El modelo en una arquitectura Cliente/Servidor se encarga de manejar la información que se encuentra en el servidor, y de procesarla según las necesidades de la aplicación. Por ejemplo, en una aplicación de comercio electrónico, el modelo podría incluir información sobre los productos, los precios, los clientes y los pedidos.

Control: Es la parte de la aplicación que se encarga de coordinar la interacción entre el usuario y el modelo. El control en una arquitectura Cliente/Servidor se ejecuta en el lado del cliente y se encarga de enviar solicitudes al servidor para obtener o actualizar la información del modelo. Por ejemplo, en una aplicación de comercio electrónico, el control podría manejar la interacción del usuario con la página web y enviar solicitudes al servidor para obtener información sobre los productos o para realizar un pedido.

Vista: Es la parte de la aplicación que se encarga de mostrar la información al usuario. La vista en una arquitectura Cliente/Servidor se ejecuta en el lado del cliente y se encarga de mostrar los datos que se encuentran en el modelo de una manera amigable para el usuario. Por ejemplo, en una aplicación de comercio electrónico, la vista podría mostrar una lista de productos con sus respectivos precios y una imagen de cada uno de ellos.

La arquitectura Cliente/Servidor divide las responsabilidades de una aplicación en tres partes: el modelo, el control y la vista. El modelo maneja la información y la lógica de negocio, el control coordina la interacción entre el usuario y el modelo, y la vista muestra la información al usuario de manera amigable.

3.2 Concepto de modelo vista controlador (MVC) en las arquitecturas Cliente / Servidor.

Modelo Vista Controlador: El Modelo-Vista-Controlador (MVC) es un patrón de arquitectura de software que se utiliza en el diseño de aplicaciones cliente/servidor. Este patrón se utiliza para separar la lógica de la aplicación en tres componentes principales: el modelo, la vista y el controlador.

La ventaja del patrón MVC es que permite separar la lógica de negocio de la aplicación de su presentación visual. Esto hace que sea más fácil de mantener y de escalar la aplicación en el futuro. Además, también permite que diferentes desarrolladores trabajen en diferentes componentes de la aplicación de manera independiente, lo que puede acelerar el proceso de desarrollo. MVC es un patrón

de arquitectura de software que se utiliza en las aplicaciones cliente/servidor para separar la lógica de negocio de la presentación visual de la aplicación. El modelo representa los datos y la lógica de negocio, la vista muestra los datos al usuario y el controlador coordina la interacción entre el usuario y el modelo.

Modelo: El modelo representa los datos y la lógica de negocio de la aplicación. Es la parte de la aplicación que se encarga de manejar la información que se encuentra en el servidor y de procesarla según las necesidades de la aplicación.

Vista: La vista es la parte de la aplicación que se encarga de mostrar los datos al usuario. Es la parte visual de la aplicación y se ejecuta en el lado del cliente. Su función es proporcionar una interfaz de usuario amigable para que el usuario pueda interactuar con la aplicación.

Controlador: El controlador es la parte de la aplicación que se encarga de coordinar la interacción entre el usuario y el modelo. Es el intermediario entre la vista y el modelo. Se encarga de recibir las solicitudes del usuario y de enviarlas al modelo correspondiente para obtener o actualizar la información. Además, también se encarga de actualizar la vista con la información obtenida del modelo.

3.3 Proceso de flujo de control a partir del MVC,) en las arquitecturas Cliente / Servidor.

Proceso de flujo de control: El proceso de flujo de control sigue un orden específico. A continuación, se describe el flujo de control básico en una aplicación MVC:

- El usuario interactúa con la vista, haciendo clic en botones, escribiendo texto en campos de entrada, etc.
- La vista envía una solicitud de acción al controlador.
- El controlador recibe la solicitud de acción y se encarga de procesarla. Esto puede implicar obtener información del modelo o actualizarla, dependiendo de la acción solicitada.
- El controlador actualiza el modelo con la información relevante, si es necesario.
- El controlador selecciona la vista adecuada para mostrar los resultados de la acción solicitada. La vista recibe la información del controlador y se actualiza para mostrar los resultados al usuario.
- El usuario ve los resultados de la acción en la vista y, si es necesario, realiza otra acción.

- El proceso se repite a partir del paso 1, dependiendo de las acciones realizadas por el usuario.

Este proceso de flujo de control permite que la lógica de la aplicación se mantenga separada de la presentación visual. Además, también permite una mayor flexibilidad y escalabilidad en el desarrollo de la aplicación, ya que los diferentes componentes se pueden desarrollar y mantener de manera independiente.

3.4 Desarrollo de software a partir del MVC, en las arquitecturas Cliente / Servidor.

El desarrollo de software utilizando el patrón de arquitectura MVC en una arquitectura Cliente/Servidor implica la creación y organización de los componentes clave: el modelo, la vista y el controlador.

A continuación se describen los pasos básicos para el desarrollo de software utilizando el patrón MVC en una arquitectura Cliente/Servidor:

Diseño del modelo: En primer lugar, se debe diseñar el modelo, que es responsable de manejar los datos y la lógica de negocio de la aplicación. El modelo debe ser capaz de realizar todas las operaciones necesarias para manipular los datos de la aplicación, como la creación, lectura, actualización y eliminación (CRUD).

Diseño de la vista: Luego, se debe diseñar la vista, que es responsable de mostrar los datos al usuario. La vista debe ser lo más fácil de usar y comprender para el usuario final, y debe ser compatible con las plataformas que se utilizarán para acceder a la aplicación.

Diseño del controlador: El siguiente paso es diseñar el controlador, que es responsable de recibir las solicitudes del usuario y enviarlas al modelo correspondiente para obtener o actualizar la información. El controlador también es responsable de actualizar la vista con la información obtenida del modelo.

Desarrollo y pruebas: Una vez que se ha diseñado el modelo, la vista y el controlador, se deben desarrollar y probar para asegurarse de que estén funcionando correctamente. Esto puede requerir el uso de herramientas de desarrollo, como entornos de programación integrados (IDE) y herramientas de prueba de software.

Implementación: Después de desarrollar y probar los componentes de la aplicación, se debe implementar la aplicación en un entorno de producción. Esto puede implicar la instalación de la aplicación en un servidor o en una plataforma de alojamiento en la nube.

Mantenimiento: Una vez que la aplicación está en producción, se debe mantener regularmente para asegurarse de que funcione correctamente y se puedan realizar mejoras en el futuro.

Este proceso de desarrollo permite una mayor flexibilidad y escalabilidad en el desarrollo de la aplicación, ya que los diferentes componentes se pueden desarrollar y mantener de manera independiente.

4. Sockets

4.1 Concepto de comunicación orientada a conexión e interfaz de programación de aplicaciones API.

Comunicación orientada a la conexión

La comunicación orientada a la conexión se aplica en el ámbito de la experiencia de usuario y el diseño de interfaces de usuario. En este contexto, la comunicación orientada a la conexión se centra en crear interfaces que sean intuitivas, atractivas y fáciles de usar para los usuarios.

Los desarrolladores de software deben tener en cuenta las necesidades, intereses y emociones de los usuarios al diseñar y desarrollar la interfaz de usuario. La comunicación debe ser clara y concisa, y los usuarios deben poder encontrar fácilmente la información que necesitan.

También se puede aplicar en el soporte técnico y la atención al cliente. En este caso, se trata de establecer una comunicación empática y respetuosa con los usuarios para ayudarles a resolver sus problemas y satisfacer sus necesidades.

La comunicación orientada a la conexión se enfoca en establecer una conexión auténtica y significativa entre los usuarios y el software, ya sea a través de la interfaz de usuario o del soporte técnico y la atención al cliente. Esto mejora la experiencia del usuario y ayuda a crear una relación de confianza y fidelidad con el software y su equipo de desarrollo.

Interfaz de programación de aplicaciones

Es un conjunto de reglas, protocolos y herramientas que permiten a los desarrolladores de software interactuar con una aplicación o sistema de software.

Una API define como una aplicación debe comunicarse con otra para acceder a sus servicios o funcionalidades. La API proporciona una interfaz clara y estandarizada para que los desarrolladores puedan integrar las funcionalidades de una aplicación en sus propios proyectos.

Permite ahorrar tiempo y esfuerzo al no tener que desarrollar una funcionalidad desde cero. Además, al utilizar una API, los desarrolladores pueden aprovechar la experiencia y conocimientos de otros desarrolladores y empresas en la creación de aplicaciones.

4.2 Proceso de comunicación y configuración orientada a conexión e interfaz de programación de aplicaciones

El proceso de comunicación en una API orientada a la conexión sigue los mismos principios que cualquier otra forma de comunicación orientada a la conexión. A continuación, se describen las etapas del proceso de comunicación en una API orientada a la conexión:

Establecimiento de la conexión: La comunicación comienza cuando el desarrollador de software establece una conexión con la API, utilizando un conjunto de protocolos y herramientas definidos por la API.

Identificación de necesidades y objetivos: En esta etapa, el desarrollador de software identifica sus necesidades y objetivos al utilizar la API. Por ejemplo, puede querer acceder a ciertos datos o funcionalidades de la aplicación que ofrece la API.

Escucha y comprensión: En esta etapa, el desarrollador de software escucha y comprende la respuesta de la API. La API proporciona una respuesta estandarizada que el desarrollador de software puede interpretar y utilizar para su aplicación.

Interacción y ajuste: En esta etapa, el desarrollador de software interactúa con la API para realizar la tarea deseada, y ajusta su estrategia si es necesario.

Cierre de la conexión: Finalmente, el desarrollador de software cierra la conexión con la API y finaliza la comunicación.

En cuanto a la configuración de una API orientada a la conexión, esto implica definir y establecer los protocolos y herramientas necesarios para permitir la comunicación entre dos aplicaciones o sistemas de software. Esto incluye la definición de los métodos de comunicación, la identificación de los datos necesarios para la comunicación, la definición de los protocolos de seguridad y la implementación de las herramientas necesarias para que la comunicación sea fluida y sin interrupciones.

4.3 Concepto de sockets

Un socket es un punto final de una conexión de red bidireccional que se puede utilizar para enviar y recibir datos entre dos dispositivos conectados en una red.

Un socket puede ser visto como una interfaz entre el software de aplicación y el sistema operativo de una computadora. Permite a los programas enviar y recibir datos en una red mediante la creación de una conexión entre dos dispositivos a través de un canal de comunicación.

Existen dos tipos principales de sockets: los sockets orientados a conexión y los sockets sin conexión. Los sockets orientados a conexión se utilizan para establecer una conexión fiable entre dos dispositivos y transferir datos de manera confiable, mientras que los sockets sin conexión se utilizan para enviar y recibir datos sin establecer una conexión previa.

4.4 Proceso del uso de sockets en aplicaciones Cliente/Servidor

El uso de sockets en aplicaciones cliente/servidor sigue un proceso generalmente dividido en dos etapas: la creación de la conexión y la transmisión de datos.

Creación de la conexión:

La primera etapa es la creación de la conexión entre el cliente y el servidor. El cliente crea un socket y utiliza una dirección IP y un número de puerto para conectarse al servidor a través de la red. El servidor también crea un socket y espera que el cliente se conecte a él.

Transmisión de datos:

Una vez que se ha establecido la conexión, el cliente y el servidor pueden intercambiar datos a través de los sockets. El cliente envía una solicitud al servidor y espera una respuesta, mientras que el servidor escucha las solicitudes de los clientes y responde a ellas.

El proceso de transmisión de datos puede ser orientado a conexión o sin conexión, dependiendo de la aplicación. En el caso de una conexión orientada, se establece una conexión estable y se transmiten los datos de manera confiable a través de ella. En el caso de una transmisión sin conexión, los datos se envían sin establecer una conexión previa, lo que puede ser más rápido, pero menos confiable.

Link de repositorio GitHub:

https://github.com/NataliAleman/1523IS_Investigaci-n.git