5. Gestión de la configuración.

La gestión de la configuración en el desarrollo de aplicaciones web en el servidor es uno de esos temas que, aunque a veces no se le presta suficiente atención, resulta ser fundamental para que todo funcione correctamente. Pero, ¿qué se entiende por gestión de la configuración? Básicamente, se trata de cómo se organizan y gestionan los diferentes ajustes y parámetros que una aplicación necesita para funcionar en diferentes entornos, como desarrollo, pruebas y producción. Es como tener un manual que le dice a la aplicación cómo comportarse dependiendo de dónde se ejecute.

5.1. Configuración de descriptores.

Cuando hablamos de descriptores de configuración, nos referimos a archivos específicos que le dicen al servidor o a la aplicación qué recursos están disponibles y cómo deben utilizarse. Un ejemplo típico en aplicaciones Java es el archivo web.xml, que se usa en aplicaciones Java EE. Este archivo describe cosas como qué servlets manejarán ciertas solicitudes, cómo se gestionarán las sesiones, y qué parámetros de inicialización se usarán.

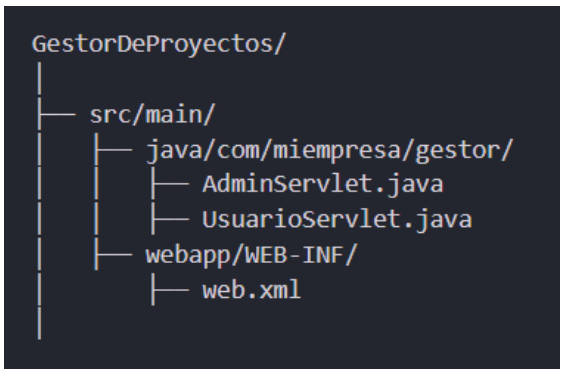
Imagina que tienes una aplicación que maneja diferentes tipos de usuarios: administradores y usuarios normales. El descriptor de configuración podría especificar qué secciones de la aplicación están restringidas a los administradores y qué partes están abiertas a todos los usuarios. Esto se configura antes de que la aplicación se inicie y asegura que cada usuario acceda a lo que le corresponde.

Ejemplo

Supongamos que estás desarrollando una aplicación web en Java EE llamada "Gestor de Proyectos", que será utilizada por dos tipos de usuarios: administradores y usuarios normales. Los administradores pueden acceder a todas las funciones de la aplicación, mientras que los usuarios normales solo pueden ver y gestionar sus propios proyectos.

Para manejar estos diferentes niveles de acceso, se utiliza un archivo de descriptor de configuración, comúnmente llamado web.xml, que define cómo la aplicación debe manejar las solicitudes, las sesiones, y las restricciones de acceso.

Primero, asegúrate de que tu proyecto Java EE tiene la siguiente estructura básica:



El archivo web.xml es el corazón de la configuración en una aplicación Java EE. Este archivo se encuentra en la carpeta WEB-INF dentro del directorio webapp. A continuación, se muestra cómo podrías configurar este archivo para manejar las diferentes rutas y accesos para administradores y usuarios normales.

```xml
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
         http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">

    <!-- Configuración del servlet para administradores -->
    <servlet>
        <servlet-name>AdminServlet</servlet-name>
        <servlet-class>com.miempresa.gestor.AdminServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>AdminServlet</servlet-name>
        <url-pattern>/admin/*</url-pattern>
    </servlet-mapping>

    <!-- Configuración del servlet para usuarios normales -->
    <servlet>
        <servlet-name>UsuarioServlet</servlet-name>
        <servlet-class>com.miempresa.gestor.UsuarioServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>UsuarioServlet</servlet-name>
        <url-pattern>/usuario/*</url-pattern>
    </servlet-mapping>
```

```
28
29      <!-- Configuración de seguridad -->
30      <security-constraint>
31          <web-resource-collection>
32              <web-resource-name>Admin Pages</web-resource-name>
33              <url-pattern>/admin/*</url-pattern>
34          </web-resource-collection>
35          <auth-constraint>
36              <role-name>ADMIN</role-name>
37          </auth-constraint>
38      </security-constraint>
39
40      <security-constraint>
41          <web-resource-collection>
42              <web-resource-name>User Pages</web-resource-name>
43              <url-pattern>/usuario/*</url-pattern>
44          </web-resource-collection>
45          <auth-constraint>
46              <role-name>USER</role-name>
47          </auth-constraint>
48      </security-constraint>
49
50      <!-- Configuración de roles -->
51      <login-config>
52          <auth-method>BASIC</auth-method>
53          <realm-name>ProyectoGestorRealm</realm-name>
54      </login-config>
55
56      <security-role>
57          <role-name>ADMIN</role-name>
58      </security-role>
59
60      <security-role>
61          <role-name>USER</role-name>
62      </security-role>
63
64  </web-app>
```

AdminServlet: Este servlet maneja todas las solicitudes que comienzan con /admin/. Se encarga de las funciones que solo pueden realizar los administradores, como la gestión de todos los proyectos, la asignación de usuarios, etc.

UsuarioServlet: Este servlet maneja todas las solicitudes que comienzan con /usuario/. Está diseñado para que los usuarios normales puedan gestionar sus propios proyectos, pero sin acceso a las funciones administrativas.

Admin Pages: Se define una restricción de seguridad que asegura que solo los usuarios con el rol ADMIN puedan acceder a las URL que comienzan con /admin/.

User Pages: Similarmente, las URL que comienzan con /usuario/ solo pueden ser accedidas por usuarios con el rol USER.

Se configuran los roles ADMIN y USER dentro del archivo web.xml. Esto indica al servidor de aplicaciones qué roles existen y cómo deben aplicarse para las diferentes partes de la aplicación.

Login Config: Se establece el método de autenticación como BASIC, lo que significa que la aplicación usará la autenticación básica HTTP, solicitando nombre de usuario y contraseña a través de un cuadro de diálogo del navegador. El realm-name define el ámbito de autenticación, que se podría configurar en el servidor para validar los usuarios.

Con esta configuración en web.xml, cuando se despliegue la aplicación en un servidor como Apache Tomcat o GlassFish, el servidor sabrá exactamente cómo manejar las solicitudes entrantes. Por ejemplo:

Un usuario normal que intente acceder a una página bajo /admin/* será redirigido a una página de error o se le negará el acceso.

Un administrador podrá acceder tanto a /admin/* como a /usuario/*, pero los usuarios normales solo podrán acceder a /usuario/*.

ACTIVIDAD 22

A continuación, se presentan una serie de afirmaciones relacionadas con la gestión de bibliotecas y la gestión de la configuración en el desarrollo web del lado del servidor. Lee cada afirmación cuidadosamente y señala si es Verdadera (V) o Falsa (F).

Los gestores de paquetes, como npm en Node.js o pip en Python, permiten a los desarrolladores instalar y gestionar bibliotecas de código de manera sencilla y eficiente.

La gestión de la configuración en aplicaciones web es menos importante en entornos de producción, ya que la aplicación no necesita ajustarse a diferentes entornos.

En una aplicación Java EE, el archivo web.xml es un ejemplo de un descriptor de configuración que define cómo se deben gestionar los servlets y las sesiones.

Es mejor escribir todo el código desde cero en lugar de utilizar bibliotecas externas, ya que esto asegura que todo el código sea original y único.

Los descriptores de configuración permiten definir qué partes de una aplicación web son accesibles para diferentes tipos de usuarios, como administradores y usuarios normales.

5.2. Configuración de ficheros.

Por otro lado, la configuración de ficheros se refiere a archivos donde se guardan parámetros y ajustes específicos que la aplicación utiliza durante su ejecución. Estos archivos suelen estar en formatos como JSON, YAML o INI. Por ejemplo, en una aplicación desarrollada con Node.js, es común tener un archivo config.json donde se especifican cosas como la URL de la base de datos, las claves de API, o los puertos en los que se ejecutará el servidor.

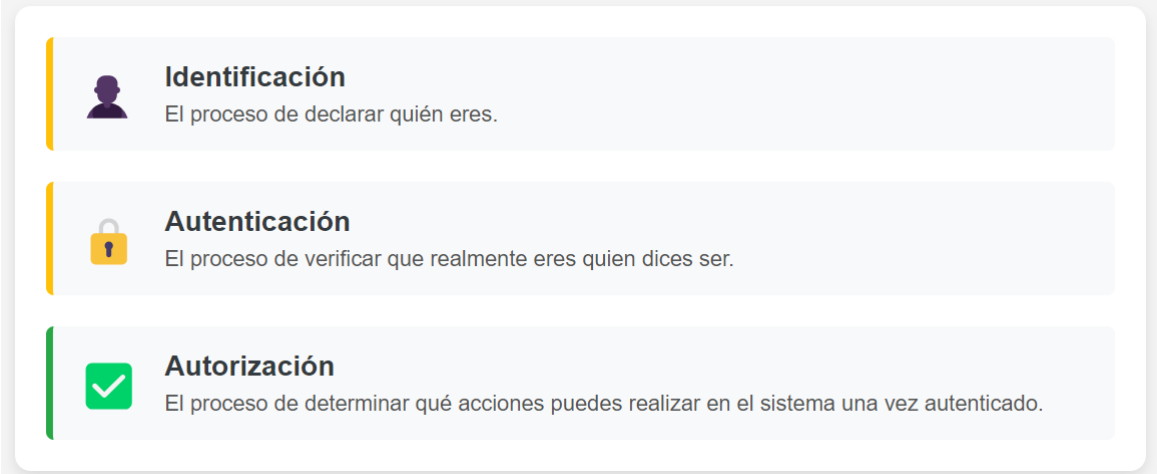
Un ejemplo práctico: supón que tu aplicación tiene que conectarse a una base de datos. En lugar de tener la dirección de la base de datos escrita directamente en el código, se guarda en un archivo de configuración. De esta manera, si la base de datos cambia de lugar, solo es necesario actualizar el archivo de configuración, sin tener que modificar el código de la aplicación. Esto facilita mucho el mantenimiento y la escalabilidad.

6. Gestión de la seguridad

La seguridad en aplicaciones web es uno de los aspectos más importantes a considerar. No es solo una cuestión técnica, sino una necesidad básica para proteger la información y los datos de los usuarios. En el desarrollo en el lado del servidor, se gestionan aspectos críticos como la identificación, autenticación y autorización de los usuarios, así como las técnicas para gestionar las sesiones de manera segura.

6.1. Conceptos de identificación, autenticación y autorización.

Empecemos por aclarar qué significa cada uno de estos términos:



Identificación es simplemente el proceso de decir quién eres. Por ejemplo, cuando un usuario ingresa su nombre de usuario en un formulario de inicio de sesión, se está identificando.

Luego viene la autenticación, que es el proceso de verificar que realmente eres quien dices ser. Esto normalmente se hace mediante una contraseña, pero también puede incluir otros métodos como autenticación de dos factores (2FA), donde además de la contraseña, se solicita un código enviado al teléfono del usuario.

Finalmente, está la autorización, que es el proceso de determinar qué acciones puede realizar un usuario en el sistema una vez autenticado. Por ejemplo, un usuario puede estar autenticado correctamente, pero solo se le permite ver ciertas páginas y no modificar datos críticos, lo que sería una función reservada para un administrador.

6.2. Técnicas para la gestión de sesiones.

La gestión de sesiones es otro pilar de la seguridad en aplicaciones web. Cuando un usuario inicia sesión en una aplicación, se crea una sesión que almacena información sobre el usuario mientras dure su visita. Esta sesión suele estar representada por un identificador único, que se guarda en una cookie en el navegador del usuario.

¿Por qué es importante gestionar bien las sesiones? Porque si no se hace correctamente, un atacante podría robar la sesión de un usuario y suplantar su identidad. Para evitar esto, se utilizan técnicas como el cifrado de las cookies de sesión, el uso de tokens de seguridad (como JWT) y la expiración

automática de las sesiones después de un tiempo de inactividad. A continuación, se exponen en profundidad estas técnicas:

1. Cifrado de cookies de sesión

Las cookies de sesión son pequeños archivos que se almacenan en el navegador del usuario y contienen información sobre la sesión activa, como el identificador de sesión (session ID). Este identificador se envía al servidor con cada solicitud, permitiendo que el servidor identifique al usuario y mantenga su sesión activa.

¿Por qué cifrar las cookies de sesión?

- Las cookies de sesión pueden ser un objetivo para los atacantes, que podrían interceptarlas (mediante ataques de "man-in-the-middle", por ejemplo) y utilizarlas para suplantar la identidad del usuario. Cifrar estas cookies es una forma de proteger la información sensible que contienen.

¿Cómo funciona el cifrado de cookies de sesión?

- Cuando una cookie de sesión se cifra, el contenido de la cookie se transforma en una cadena de caracteres ilegible a menos que se tenga la clave de cifrado adecuada. Esto significa que, incluso si un atacante logra interceptar la cookie, no podrá extraer información útil sin la clave de cifrado.

Supongamos que una cookie de sesión contiene un identificador de usuario como user_id=12345. Si la cookie no está cifrada, un atacante podría interceptar esta información y utilizarla para suplantar al usuario. Con el cifrado, el contenido de la cookie podría verse como algo similar a ERUIOHVFDS0EFA, que es ilegible para cualquiera que no tenga la clave de cifrado.

En muchos frameworks y lenguajes de servidor (como Django en Python o Express en Node.js), es común utilizar middleware que automáticamente cifra las cookies de sesión antes de enviarlas al navegador del usuario. El servidor luego descifra la cookie cuando recibe una solicitud.

```
# Ejemplo en Django (Python)
# En settings.py
SESSION_COOKIE_SECURE = True
SESSION_COOKIE_HTTPONLY = True
SESSION_COOKIE_NAME = 'sessionid'
SESSION_COOKIE_SAMESITE = 'Lax'
```

2. Uso de tokens de seguridad (JWT - JSON Web Tokens)

¿Qué es un token de seguridad (JWT)?

- Un JSON Web Token (JWT) es un estándar para crear tokens de seguridad que representan datos de manera compacta y auto-contenida. Un JWT suele incluir información sobre el usuario (como su ID, nombre, roles, etc.) y está firmado digitalmente para asegurar que el contenido no ha sido manipulado.

Un JWT tiene tres partes:

- Header (Encabezado): Incluye el tipo de token y el algoritmo de cifrado.
- Payload (Carga útil): Contiene las declaraciones (claims), como la identidad del usuario y cualquier otra información relevante.
- Signature (Firma): Se crea combinando el encabezado y la carga útil con una clave secreta, asegurando que el token no ha sido alterado.

Cuando un usuario inicia sesión, el servidor genera un JWT que contiene información sobre el usuario y lo envía al cliente. Este token se almacena en el navegador (generalmente en una cookie

o en el almacenamiento local) y se envía con cada solicitud al servidor. El servidor valida la firma del token para asegurarse de que no ha sido manipulado.

3. Expiración automática de sesiones después de un tiempo de inactividad

¿Qué es la expiración automática de sesiones?

- La expiración automática de sesiones es una técnica que finaliza automáticamente la sesión de un usuario después de un período de inactividad. Esto ayuda a proteger las cuentas de los usuarios en caso de que dejen su sesión abierta en un dispositivo compartido o público.

¿Cómo funciona la expiración de sesiones?

- El servidor mide el tiempo de inactividad del usuario, es decir, el tiempo desde la última acción realizada. Si el usuario no interactúa con la aplicación dentro de un período de tiempo determinado (por ejemplo, 30 minutos), la sesión se invalida automáticamente.
- Si se está utilizando un JWT, este token puede ser configurado para que expire automáticamente después de un tiempo. Una vez que el token expira, el usuario deberá autenticarse nuevamente.

Muchos frameworks web permiten configurar fácilmente la expiración de las sesiones. A continuación, se muestra cómo se haría en Django y en Express:

```python
# Ejemplo en Django (Python)
# En settings.py
SESSION_COOKIE_AGE = 1800  # 30 minutos
SESSION_EXPIRE_AT_BROWSER_CLOSE = True
```

```javascript
// JavaScript (Node.js)

const jwt = require('jsonwebtoken');

// Generar un token con expiración
function generateToken(user) {
    const token = jwt.sign({ id: user.id, username: user.username }, 'secret_key', {
        expiresIn: '30m'  // El token expira en 30 minutos
    });
    return token;
}
```

Importante

- Es importante implementar el "logout" o cierre de sesión seguro, que invalida la sesión activa en el servidor cuando el usuario decide cerrar su sesión. De esta manera, aunque alguien robara la cookie de sesión, ya no podría usarla para acceder a la cuenta del usuario.

Todo esto, aunque parece técnico, es esencial para proteger tanto la aplicación como a los usuarios que la utilizan. En un entorno donde las amenazas son constantes, no se puede pasar por alto la correcta gestión de la seguridad.

7. Gestión de errores.

Cuando se desarrolla una aplicación web en el lado del servidor, una de las cosas que siempre se debe tener en mente es que, tarde o temprano, ocurrirán errores. Esto es completamente normal y parte del proceso de programación. La gestión de errores no solo se trata de evitar que la aplicación falle, sino de manejar esos fallos de manera que no afecten negativamente la experiencia del usuario y, a la vez, se puedan detectar y corregir problemas de manera eficiente.

7.1. Técnicas de recuperación de errores.

La recuperación de errores se refiere a las estrategias que se implementan para que, cuando ocurre un error, la aplicación pueda continuar funcionando o, al menos, volver a un estado estable sin causar mayores inconvenientes al usuario. Una técnica común es la implementación de "fallbacks", que son alternativas que se activan si algo falla.

Por ejemplo, imagina que una aplicación web debe mostrar datos de un servicio externo, como el clima. Si por alguna razón el servicio de clima no está disponible, la aplicación podría mostrar un mensaje genérico como "Información no disponible en este momento", en lugar de mostrar un error técnico. Esto permite que la aplicación siga funcionando para otras tareas.

Otra técnica es el uso de logs o registros. Es muy útil registrar los errores que ocurren, con detalles como la fecha, hora, y el contexto en el que ocurrió el error. Esto ayuda a los desarrolladores a identificar y corregir errores más rápido. Estos logs no deberían mostrarse al usuario final, pero son una herramienta fundamental para el equipo técnico.

7.2. Programación de excepciones.

La programación de excepciones es una práctica que se usa para manejar situaciones anómalas en el código. En lugar de dejar que un error detenga la ejecución del programa, se captura ese error (o "excepción") y se maneja de una manera controlada. Esto puede ser tan simple como mostrar un mensaje de error amigable al usuario o tan complejo como iniciar un proceso de recuperación.

Por ejemplo, en Python, se podría manejar una excepción al intentar dividir por cero:

```python
# python

try:
    resultado = 10 / 0
except ZeroDivisionError:
    print("No se puede dividir por cero")
```

Aquí, en lugar de que la aplicación se detenga abruptamente, se captura la excepción ZeroDivisionError y se muestra un mensaje explicativo. Esto asegura que la aplicación siga funcionando incluso cuando ocurren problemas.

8. Transacciones y persistencia.

Cuando se habla de transacciones y persistencia en el contexto del desarrollo de aplicaciones web en servidor, nos referimos a cómo la aplicación maneja y almacena datos de manera confiable y segura. Las transacciones permiten agrupar una serie de operaciones de base de datos en una sola unidad de trabajo, lo que significa que o bien todas las operaciones se completan con éxito, o ninguna se realiza, manteniendo la integridad de los datos.

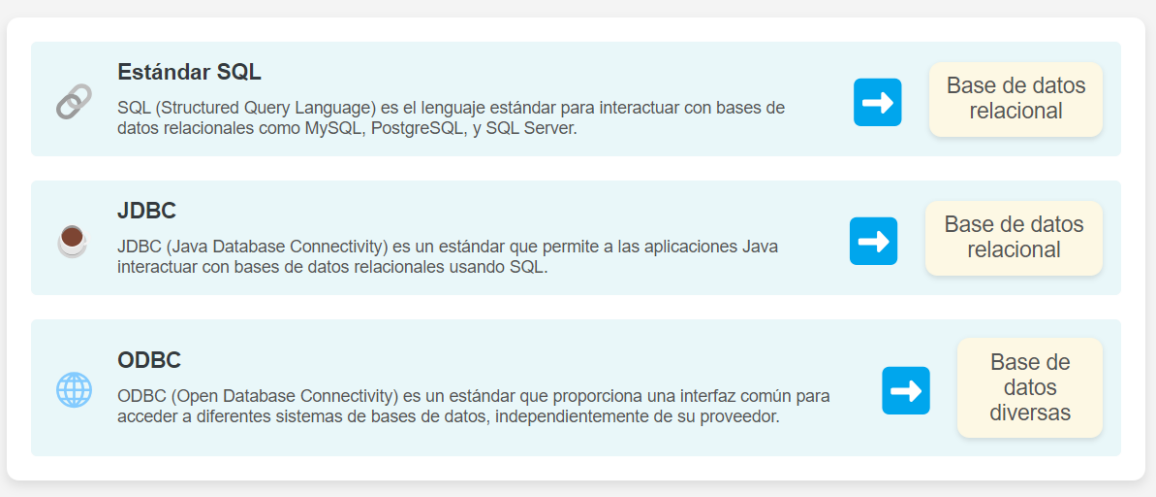
8.1. Acceso a bases de datos. Conectores.

El acceso a bases de datos es fundamental para casi cualquier aplicación web. Los conectores son las piezas de software que permiten que la aplicación en el servidor se comunique con la base de datos. Cada base de datos tiene su propio conector específico. Por ejemplo, para acceder a una base de datos MySQL desde una aplicación en Python, se usa el conector mysql-connector-python. Estos conectores se encargan de enviar consultas a la base de datos y devolver los resultados a la aplicación:

Supón que tienes una aplicación de ventas en línea. Cuando un cliente hace una compra, la aplicación usa un conector para acceder a la base de datos y actualizar el inventario. Sin un conector adecuado, esta comunicación simplemente no podría ocurrir.

8.2. Estándares para el acceso a bases de datos.

Los estándares son guías o protocolos que aseguran que las aplicaciones puedan acceder a las bases de datos de manera uniforme, sin importar qué base de datos específica se esté utilizando:



Uno de los estándares más conocidos es SQL (Structured Query Language), que es un lenguaje usado para interactuar con bases de datos relacionales como MySQL, PostgreSQL y SQL Server.

Además de SQL, existen estándares como JDBC (Java Database Connectivity) para aplicaciones Java, y ODBC (Open Database Connectivity), que es más general y permite a las aplicaciones comunicarse con diferentes sistemas de bases de datos utilizando una interfaz común.

Estos estándares son importantes porque permiten que los desarrolladores creen aplicaciones que pueden trabajar con diferentes bases de datos sin necesidad de reescribir grandes partes del código, haciendo el desarrollo más flexible y eficiente.

8.3. Gestión de la configuración de acceso a bases de datos.

Configurar correctamente el acceso a la base de datos es esencial para que la aplicación pueda operar correctamente. Esto incluye especificar la ubicación de la base de datos (como una URL), las credenciales necesarias para conectarse (nombre de usuario y contraseña), y cualquier otro parámetro necesario, como el tiempo de espera de las conexiones.

Esta configuración generalmente se almacena en archivos de configuración, como un archivo config.json en aplicaciones Node.js o un archivo .env que almacena variables de entorno. Estos archivos permiten cambiar la configuración sin tener que modificar el código fuente, lo cual es útil cuando se despliega la aplicación en diferentes entornos (desarrollo, pruebas, producción).

El siguiente esquema ejemplifica la gestión de la configuración de acceso a bases de datos, ilustrando cómo se maneja en diferentes entornos:

8.4. Acceso a directorios y otras fuentes de datos.

Además de las bases de datos tradicionales, las aplicaciones web a menudo necesitan acceder a otras fuentes de datos, como archivos en un sistema de archivos o servicios web externos. Esto se gestiona de manera similar al acceso a bases de datos, pero con sus propias consideraciones.

Por ejemplo, si una aplicación necesita leer un archivo CSV que está almacenado en un servidor, se debe asegurar que la aplicación tenga los permisos correctos para acceder al directorio donde está guardado el archivo. Del mismo modo, si se accede a una API externa, la configuración debe incluir la URL de la API y cualquier token o clave de acceso necesaria.

8.5. Programación de transacciones.

Las transacciones son una manera de asegurar que una serie de operaciones en la base de datos se completen todas correctamente o ninguna se realice. Esto es especialmente importante en situaciones donde múltiples operaciones dependen entre sí.

Imagina que, en una aplicación bancaria, un usuario transfiere dinero de una cuenta a otra. Primero, se debe restar la cantidad de la cuenta de origen y luego sumar esa cantidad a la cuenta de destino. Si solo se completa una de estas operaciones, el saldo total del banco sería incorrecto. Aquí es donde las transacciones entran en juego: si ambas operaciones se completan con éxito, la transacción se confirma (se hace un "commit"); si alguna falla, la transacción se revierte (se hace un "rollback"), y se vuelve al estado inicial.

Importante

- La programación de transacciones es vital para mantener la integridad y consistencia de los datos en cualquier aplicación que maneje operaciones complejas.

ACTIVIDAD 23

Imagina que estás trabajando como desarrollador en una aplicación web de comercio electrónico que maneja miles de transacciones diarias. Los usuarios compran productos, y cada compra implica actualizar el inventario, registrar el pedido, y procesar el pago. Un día, descubres que hay un problema en la aplicación: algunos usuarios están reportando que, después de realizar un pedido, su pago ha sido procesado, pero los productos siguen mostrando disponibilidad en el inventario.

Esto podría significar que las operaciones de base de datos que actualizan el inventario y registran el pedido no se están completando correctamente, pero el procesamiento del pago sí. Este problema pone en riesgo la integridad de los datos y la confianza de los usuarios en la plataforma.

¿Qué papel juegan los conectores de base de datos y la configuración de acceso en este tipo de problemas? Considera cómo una configuración incorrecta o un fallo en el conector podría afectar la fiabilidad de la comunicación con la base de datos.

¿Cómo crees que un problema como este afecta la percepción de los usuarios sobre la plataforma? Reflexiona sobre la importancia de la consistencia y confiabilidad en las aplicaciones que manejan datos críticos, como transacciones financieras.

¿Qué medidas podrías implementar para prevenir este tipo de problemas en el futuro? Considera la posibilidad de mejorar la gestión de transacciones, revisar la configuración de acceso a la base de datos, o implementar alertas que te notifiquen sobre posibles fallos en el procesamiento de transacciones.

9. Componentes en servidor. Ventajas e inconvenientes en el uso de contenedores de componentes.

Al desarrollar aplicaciones web en el servidor, uno de los enfoques más comunes es el uso de componentes. Un componente en servidor es una pieza de código que realiza una función específica dentro de una aplicación. Pueden ser responsables de manejar la lógica del negocio, gestionar la conexión a la base de datos o procesar solicitudes de los usuarios.

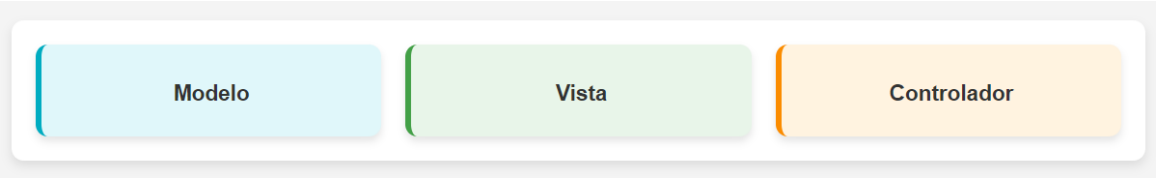
Ahora bien, para manejar estos componentes de manera eficiente, se suelen utilizar contenedores de componentes. Estos contenedores son entornos que proporcionan un marco donde los componentes pueden vivir, comunicarse y trabajar juntos. Ejemplos de contenedores populares son Apache Tomcat para aplicaciones Java o Docker, que permite empaquetar y desplegar aplicaciones en contenedores.

A continuación, detallan las ventajas e inconvenientes del uso de contenedores de componentes:

- ✓ Aislamiento: cada componente se ejecuta en su propio entorno, lo que evita conflictos entre ellos. Imagina que tienes dos versiones diferentes de una biblioteca; con contenedores, cada una puede coexistir sin problemas.
- ✓ Escalabilidad: los contenedores permiten escalar una aplicación de manera sencilla. Por ejemplo, si un componente específico recibe mucha carga, se puede replicar el contenedor que lo aloja y distribuir la carga entre varias instancias.
- ✓ Portabilidad: los contenedores aseguran que la aplicación se ejecute de manera consistente sin importar dónde se despliegue. Por ejemplo, lo que funciona en el entorno de desarrollo funcionará igual en producción.
- × Complejidad en la gestión: aunque los contenedores simplifican muchas cosas, también añaden una capa de complejidad. Por ejemplo, gestionar múltiples contenedores y asegurarse de que todos se comuniquen correctamente puede ser complicado.
- × Consumo de recursos: cada contenedor consume recursos del sistema (memoria, CPU), lo que puede ser un problema si no se gestiona bien. Especialmente en servidores con recursos limitados, esto puede llevar a problemas de rendimiento.
- × Curva de aprendizaje: herramientas como Docker requieren tiempo y esfuerzo para aprender, lo que puede retrasar el inicio del desarrollo si el equipo no está familiarizado con ellas.

10. Modelos de desarrollo. El modelo vista controlador.

El modelo vista controlador (MVC, por sus siglas en inglés) es uno de los patrones de diseño más utilizados en el desarrollo de aplicaciones web. Este patrón divide la aplicación en tres componentes principales: modelo, vista y controlador, lo que facilita la organización y mantenimiento del código:



10.1. Modelo: programación de acceso a datos.

El modelo es la parte de la aplicación que se encarga de manejar los datos. Aquí es donde se define cómo los datos se almacenan, se recuperan y se manipulan. El modelo interactúa directamente con la base de datos y otras fuentes de datos.

Por ejemplo, en una tienda en línea, el modelo gestionaría la información sobre los productos, como su nombre, precio y disponibilidad. Cuando un usuario busca un producto, el modelo accede a la base de datos para recuperar esa información y se la pasa al controlador.

10.2. Vista: desarrollo de aplicaciones en cliente. Eventos e interfaz de usuario.

La vista es la parte de la aplicación que interactúa con el usuario. Se encarga de mostrar los datos y recibir las entradas del usuario. En una aplicación web, la vista generalmente está compuesta por HTML, CSS y JavaScript.

Por ejemplo, en la misma tienda en línea, la vista mostraría la lista de productos y permitiría al usuario filtrar los resultados o agregar un producto al carrito. Los eventos en la vista, como hacer clic en un botón, son capturados y enviados al controlador para ser procesados.

10.3. Programación del controlador.

El controlador actúa como intermediario entre el modelo y la vista. Recibe las solicitudes del usuario (a través de la vista), las procesa, y decide qué datos necesita del modelo. Luego, pasa esa información a la vista para que se muestre al usuario.

Siguiendo con el ejemplo de la tienda en línea, si el usuario busca un producto específico, el controlador recibe la solicitud, le pide al modelo que busque ese producto en la base de datos, y luego le pasa los resultados a la vista para que los muestre.

10.4. Documentación del software. Inclusión en código fuente. Generadores de documentación

La documentación es una parte fundamental del desarrollo de software. No solo sirve para recordar cómo funciona el código, sino que también ayuda a otros desarrolladores a entender y mantener la aplicación.

Incluir documentación directamente en el código es una práctica recomendada. Esto se hace a través de comentarios que explican qué hace cada parte del código. El siguiente esquema explica cómo se incluyen comentarios en diferentes lenguajes de programación del lado del servidor:

Además de los comentarios en el código, existen herramientas que pueden generar documentación automáticamente a partir de estos comentarios. Por ejemplo, en Python, se puede usar Sphinx para generar documentación en varios formatos (HTML, PDF, etc.) a partir de los comentarios en el código. En JavaScript, herramientas como JSDoc cumplen una función similar.

ACTIVIDAD 24

Relaciona cada concepto con su correspondiente definición o ejemplo:

Modelo
Vista
Controlador
Documentación del software
Generadores de documentación

A. Parte de la aplicación que interactúa directamente con el usuario, mostrando datos y capturando entradas mediante HTML, CSS y JavaScript.

B. Parte de la aplicación encargada de manejar los datos, interactuando con la base de datos y otras fuentes de datos para almacenar, recuperar y manipular la información.

C. Actúa como intermediario entre la vista y el modelo, procesando las solicitudes del usuario, obteniendo datos del modelo, y enviándolos a la vista para su presentación.

D. Proceso y herramientas utilizadas para crear explicaciones claras y detalladas sobre el funcionamiento del código, facilitando el entendimiento y mantenimiento del software.

E. Herramientas que toman los comentarios en el código fuente y generan documentación en varios formatos, como HTML o PDF, para facilitar su consulta y distribución.

11. Prueba de autoevaluación.

¿Cuál es una característica clave de los lenguajes de programación en servidor?

a) Solo manejan HTML y CSS.

b) Manejan la lógica de negocio, interactúan con bases de datos y gestionan la seguridad.

c) Son responsables únicamente del diseño gráfico de la página.

¿Qué lenguaje se utiliza principalmente con el framework Django?

a) PHP.

b) Python.

c) JavaScript.

¿Cuál es una ventaja de los lenguajes compilados como Java y C#?

a) Son más lentos pero más fáciles de aprender.

b) Tienen un rendimiento superior en tiempo de ejecución.

c) No necesitan configurarse para trabajar en múltiples plataformas.

¿Qué característica hace que Node.js sea popular para aplicaciones en tiempo real?

a) Su capacidad para manejar bases de datos SQL.

b) Su modelo de ejecución basado en eventos.

c) Su capacidad para compilar código Java.

¿Qué se entiende por persistencia de datos en aplicaciones web?

a) La capacidad de una aplicación para recordar datos entre sesiones.

b) El proceso de diseño gráfico de las páginas web.

c) La velocidad con la que se ejecutan las solicitudes.

Los lenguajes de programación en servidor son esenciales para manejar la ___ de negocio y el acceso a datos.

PHP, Python y Ruby son ejemplos de lenguajes ___ orientados a servidor.

Node.js permite ejecutar código ___ en el servidor.

Los lenguajes compilados suelen tener un rendimiento ___ en tiempo de ejecución.

El patrón MVC divide la aplicación en ___, vista y controlador.

Resumen

El desarrollo de software es un proceso complejo que requiere planificación meticulosa y un enfoque estructurado. Existen varios modelos de ciclo de vida del software, como el modelo en cascada, iterativo, incremental, en V, basado en componentes y el desarrollo rápido. Cada modelo tiene sus ventajas y desventajas, y se selecciona en función de factores como la claridad de los requisitos, plazos, complejidad del proyecto y retroalimentación del cliente. Además, el análisis de requisitos es esencial para asegurar que el producto final cumpla con las expectativas del cliente.

La orientación a objetos es un paradigma de programación que organiza el software en torno a objetos y clases. Facilita la modularidad, reutilización de código y mantenimiento del software. Sus principios incluyen encapsulación, herencia y polimorfismo, y se emplea en lenguajes como Java, Python y C#. Además, se destacan herramientas como UML para modelar sistemas orientados a objetos.

Las arquitecturas web son fundamentales para estructurar cómo los componentes de una aplicación en línea se organizan y comunican entre sí. Este enfoque utiliza un modelo de capas, que separa la presentación, la lógica de negocio y el acceso a datos, lo que facilita el mantenimiento y la escalabilidad. También se destacan las plataformas y herramientas clave para el desarrollo en el lado del servidor, como Node.js, Java con Spring, y Django, que son fundamentales para construir aplicaciones web eficientes.

Los lenguajes de programación del lado servidor son fundamentales para desarrollar aplicaciones web dinámicas y seguras, gestionando la lógica de negocio, acceso a bases de datos y seguridad. Los lenguajes más comunes incluyen PHP, Python, Ruby, JavaScript (Node.js), Java y C#. Cada uno tiene ventajas y desventajas que deben considerarse al elegir el más adecuado para un proyecto, teniendo en cuenta factores como rendimiento, seguridad y facilidad de uso.