1. **¿Qué es Java EE y qué tecnologías lo componen?**

Java EE era el "paquete" de especificaciones oficiales para construir aplicaciones empresariales en Java. Desde 2019 pasó a llamarse **Jakarta EE** (mismo concepto, nuevo nombre). La idea es que define **interfaces y contratos**; luego distintos servidores las implementan (Tomcat, Payara/GlassFish, WildFly, etc.).

Componentes más conocidos (no hace falta usarlos todos):

- **Servlets**: controladores HTTP (peticiones GET/POST).
- **JSP + JSTL**: vistas en el servidor (HTML con expresiones y tags).
- **JAX-RS**: servicios REST.
- **JPA**: acceso a datos con ORM (tipo Hibernate).
- **CDI**: inyección de dependencias.
- **Bean Validation**: validaciones con anotaciones.
- **WebSocket**, **JMS**, **Security** (autenticación/autorización), entre otras.

2. **¿Cuál es el rol de JSP, Servlets y DAO en una arquitectura J2EE?**

- **Servlets (Controlador)**: reciben la petición, leen parámetros, llaman a la capa de negocio/DAO y **deciden a qué vista** ir (forward/redirect). No renderizan HTML directo, solo coordinan.
- **JSP (Vista):** Generan el **HTML** que ve el usuario. Usan ${atributos} y tags JSTL para mostrar datos que el servlet dejó en request.setAttribute(...).
- **DAO (Modelo – acceso a datos):** Clases que **hablan con la BD** (JDBC/JPA). Tienen métodos tipo listarLibros(), insertarSolicitud(...). Encapsulan SQL y evitan mezclarlo con la vista o el controlador.

3. **¿Qué ventajas ofrece frente a otras tecnologías de desarrollo web?**

- **Estandarización**: no dependes de un framework "de moda"; las APIs son estándar (Jakarta EE). Cambias de servidor y la app sigue funcionando con pocos ajustes.
- **Madurez y estabilidad**: Java/Jakarta llevan años en producción grande. Mucha documentación y soluciones ya probadas.
- **Ecosistema enorme**: librerías para todo (seguridad, BD, pruebas, logging, etc.).

- **Escalabilidad y rendimiento**: el stack Java está optimizado para cargas serias si se configura bien.
- **Tipos fuertes y herramientas**: compilador, IDEs (IntelliJ/Eclipse/VS Code), análisis estático… ayudan a evitar bugs.
- **Patrones claros (MVC)**: separa responsabilidades (vistas/servlets/DAO) y el código queda más mantenible.

Comparado con otras opciones:

- **Vs. Node.js/Express**: Node es rápido para prototipos y tiempo real, pero en Java tienes tipado fuerte y un ecosistema enterprise más tradicional.
- **Vs. .NET**: .NET también es potente; la diferencia suele ser el **ecosistema** que ya tengas y el **hosting** (Windows/Linux) y experiencia del equipo.
- **Vs. frameworks "todo en uno" (Spring Boot)**: Spring Boot simplifica mucho la configuración; Jakarta EE/JSP/Servlet es más "crudo", pero está **muy bien para aprender** los fundamentos web en Java y entender el MVC de base.