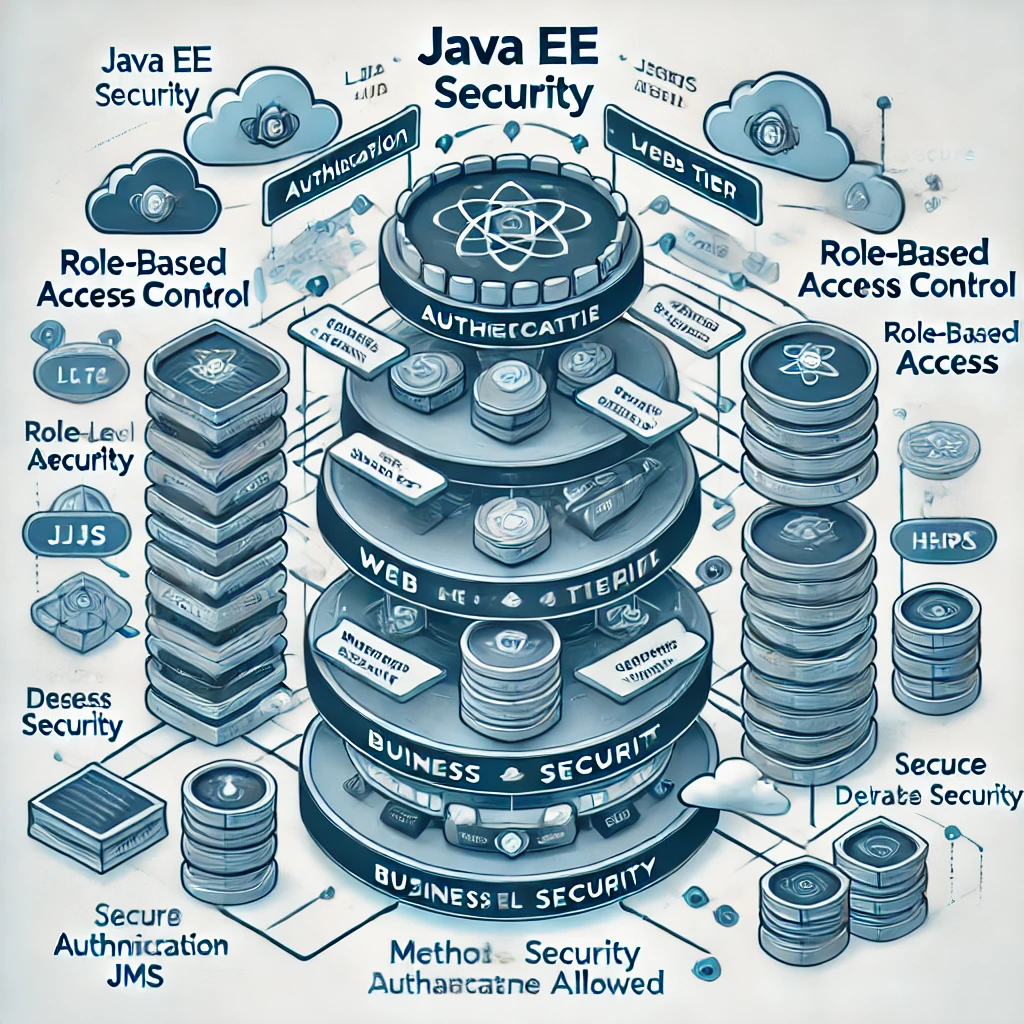
**Name :** Patel JenishKumar Kishorbhai

**Roll No :** 50

**Assignment 4**

**Java EE 8 Security Architecture:**



**The architecture of Java EE security is designed to protect applications and resources in a standardized and configurable manner.**

**Java EE Security Architecture Components**

1. **Authentication:**
   * **Ensures that users or systems are who they claim to be.**
   * **Java EE supports multiple authentication mechanisms:** 
     + **HTTP Basic Authentication**
     + **HTTP Digest Authentication**
     + **Form-based Authentication**
     + **Client Certificate Authentication**
     + **Custom Authentication (via JAAS or programmatic approaches)**
2. **Authorization:**
   * **Governs what authenticated users or systems are allowed to do.**
   * **Managed by:** 
     + **Role-based Access Control (RBAC): Maps users to roles and assigns permissions to roles.**
     + **Annotations like @RolesAllowed or @PermitAll to specify which methods or components are accessible by specific roles.**
     + **Deployment descriptors (web.xml or ejb-jar.xml) to define access policies.**
3. **Transport Security:**
   * **Ensures data integrity and confidentiality during transmission.**
   * **Uses protocols like HTTPS (SSL/TLS) for secure communication between clients and servers.**
4. **Secure Components:**
   * **Security features apply to various Java EE components, such as:** 
     + **Servlets: Secured via web.xml or annotations.**
     + **Enterprise JavaBeans (EJBs): Secured via annotations or deployment descriptors.**
     + **Java Message Service (JMS): Protected through authentication and authorization.**
     + **RESTful APIs: Secured via filters, interceptors, or annotations.**
5. **Security Realms:**
   * **A database or directory service (e.g., LDAP, file-based realm, or JDBC realm) that stores user credentials and role mappings.**
6. **Programmatic Security:**
   * **Developers can use APIs to enforce security rules dynamically at runtime.**
   * **Examples:** 
     + **HttpServletRequest.isUserInRole()**
     + **EJBContext.isCallerInRole()**
     + **SecurityContext.getCallerPrincipal()**

**Here’s the typical structure of Java EE security architecture:**

1. **Client Tier:**
   * **End users or systems interacting with the application via web browsers, standalone clients, or other services.**
   * **Authentication occurs here, usually via login forms, certificates, or tokens.**
2. **Web Tier (Web Container):**
   * **Handles HTTP requests and applies transport-layer security (HTTPS).**
   * **Provides mechanisms for role-based access control to servlets, JSPs, or REST APIs.**
3. **Business Tier (EJB Container):**
   * **Enforces method-level security using role-based annotations or deployment descriptors.**
   * **Verifies the identity and roles of users making requests.**
4. **Integration Tier:**
   * **Applies security when interacting with backend systems like databases, JMS queues, or external APIs.**

**Example Annotations in Java EE Security**

* **@RolesAllowed({"Admin", "User"}): Restrict access to certain roles.**
* **@DenyAll: Deny access to everyone.**
* **@PermitAll: Allow access to everyone.**
* **@DeclareRoles: Declares security roles for the application.**