Especialización Java Enterprise Edition

Desarrollo Web con Servlets, JSP y Jakarta EE

Nivel: Básico / Iniciación | Código: UNW-JEE-101 | Duración: 20 horas académicas

uneweb.edu.ve
Programa de Formación en Desarrollo Empresarial

Java EE — Nivel I

Desarrollo Web con Servlets, JSP y Jakarta EE

Presentación del curso

Este curso introduce al participante en el desarrollo de aplicaciones web empresariales con la plataforma Jakarta EE (anteriormente Java EE). Partiendo de los conceptos del protocolo HTTP, el participante construirá aplicaciones web dinámicas con Servlets y JavaServer Pages (JSP), gestionará sesiones de usuario, aplicará el patrón MVC básico y desplegará sus aplicaciones en un servidor de aplicaciones Jakarta EE. Es el punto de entrada ideal para quienes ya dominan Java SE y desean ingresar al mundo del desarrollo empresarial backend.

Objetivo general

Capacitar al participante en el desarrollo de aplicaciones web dinámicas utilizando la capa web de Jakarta EE: Servlets, JSP, JSTL y el patrón MVC, con despliegue en un servidor de aplicaciones Jakarta EE compatible.

Objetivos específicos

- Comprender la arquitectura web de Jakarta EE: contenedor web, ciclo de vida de Servlets y configuración con web.xml y anotaciones.
- Procesar peticiones HTTP GET y POST, leer parámetros, cabeceras y cuerpo de la solicitud.
- Generar respuestas HTML dinámicas desde Servlets y gestionar redirecciones y reenvíos (forward/redirect).
- Crear vistas dinámicas con JavaServer Pages (JSP): directivas, scriptlets, expresiones EL y acciones estándar.
- Utilizar la librería de etiquetas estándar JSTL para control de flujo, iteración y formateo sin código Java en JSP.
- Gestionar el estado del usuario con cookies y sesiones HTTP (HttpSession).
- Implementar el patrón MVC con Servlet como controlador, Java Beans como modelo y JSP como vista.
- Acceder a bases de datos desde la capa web usando JDBC y un DataSource configurado en el servidor.
- Desplegar y probar aplicaciones WAR en Apache Tomcat / WildFly.

Dirigido a

- Desarrolladores Java SE que deseen ingresar al desarrollo web empresarial.
- Profesionales que quieran comprender las bases sobre las que se construyen frameworks como Spring MVC y Jakarta Faces.
- Estudiantes que busquen una comprensión sólida del protocolo HTTP y el modelo petición-respuesta.

Requisitos previos

- Haber aprobado Java SE Nivel II (UNW-JSE-201) o dominio equivalente de POO y colecciones en Java.
- Conocimientos básicos de HTML y CSS.
- Familiaridad con el concepto de base de datos relacional y SQL básico.

Metodología

El proyecto integrador del curso es un sistema de gestión de tareas (To-Do App) con autenticación de usuario, que evoluciona módulo a módulo desde una respuesta HTML estática hasta una aplicación MVC completa con persistencia en base de datos. Claude.ai se utiliza para interpretar mensajes de error del servidor, generar fragmentos de configuración XML y revisar el diseño de flujos HTTP.

Plan de contenidos (20 horas académicas)

Módulo	Tema	Horas
1	Arquitectura web empresarial: HTTP, Jakarta EE, contenedor web, instalación de Tomcat/WildFly y primer Servlet	2
2	Servlets avanzados: ciclo de vida, parámetros, cabeceras, forward, redirect y filtros (Filter)	3
3	JavaServer Pages (JSP): directivas, scriptlets, Expression Language (EL) y objetos implícitos	2
4	JSTL: core (c:forEach, c:if, c:choose), fmt (formateo de fechas y números) y fn (funciones de cadena)	2
5	Gestión de estado: cookies, sesiones HTTP (HttpSession) e invalidación de sesión	2
6	Patrón MVC: Servlet controlador, Java Beans como modelo, JSP como vista y arquitectura de paquetes	3
7	Acceso a datos desde la capa web: JDBC, DataSource en JNDI, DAO y pool de conexiones	3
8	Seguridad básica, manejo de errores, empaquetado WAR, despliegue en servidor y proyecto final	3

Detalle de módulos

Módulo 1 — Arquitectura web empresarial y primer Servlet

- El protocolo HTTP: métodos (GET, POST, PUT, DELETE), códigos de estado (2xx, 3xx, 4xx, 5xx), cabeceras y cuerpo.
- Arquitectura Jakarta EE: servidor de aplicaciones, contenedor web, contenedor EJB y servicios de plataforma.
- Instalación y configuración de Apache Tomcat 10.x (Jakarta EE 10 compatible).
- Estructura de una aplicación web Java (WAR): WEB-INF, web.xml, clases y dependencias.
- Primer Servlet con @WebServlet: sobreescribir doGet(), HttpServletRequest y HttpServletResponse.
- Configuración del proyecto con Maven (maven-war-plugin) y despliegue en Tomcat desde IntelliJ IDEA.

Módulo 2 — Servlets avanzados y filtros

- Ciclo de vida del Servlet: init(), service(), destroy(); instancia única y thread safety.
- Lectura de parámetros de formulario: getParameter(), getParameterValues() y getParameterMap().
- Cabeceras de petición y respuesta: getHeader(), setHeader(), setContentType(), setCharacterEncoding().
- RequestDispatcher.forward() vs. HttpServletResponse.sendRedirect(): cuándo usar cada uno.
- Listeners: ServletContextListener, HttpSessionListener para eventos del ciclo de vida.
- Filtros (Filter): interceptación de peticiones para logging, autenticación y compresión; cadena de filtros.

Módulo 3 — JavaServer Pages (JSP)

- JSP como solución a la generación de HTML en Servlets: separación de presentación.
- Directivas: page (contentType, import, errorPage), include, taglib.
- Scriptlets <% %>, expresiones <%= %> y declaraciones <%! %> (legacy; uso desaconsejado).
- Expression Language (EL): ${expresión}, acceso a atributos de request/session/application, operadores y funciones.
- Objetos implícitos: request, response, session, application, pageContext, out, config.
- Acción estándar , , y .

Módulo 4 — JSTL

- Librería JSTL Core: , , , , //, , , , .

- JSTL fmt: , , para i18n básica.
- JSTL fn: funciones de cadena fn:length(), fn:contains(), fn:substring(), fn:toUpperCase().
- Buenas prácticas: eliminar scriptlets de JSP; usar JSTL + EL exclusivamente en la vista.
- Laboratorio: listado de tareas con JSTL (filtros, contadores y formateo de fechas).

Módulo 5 — Gestión de estado: cookies y sesiones

- El problema del estado en HTTP sin estado (stateless): necesidad de mecanismos de sesión.
- Cookies: javax.servlet.http.Cookie; atributos path, maxAge, secure, httpOnly.
- HttpSession: creación con request.getSession(), setAttribute/getAttribute, invalidate().
- Seguimiento de sesión: URL rewriting como alternativa cuando las cookies están deshabilitadas.
- Expiración de sesión: setMaxInactiveInterval() y configuración en web.xml.
- Laboratorio: sistema de login/logout que almacena el usuario en sesión y protege rutas.

Módulo 6 — Patrón MVC con Servlets y JSP

- El patrón MVC aplicado a la web: Modelo (datos y lógica de negocio), Vista (JSP) y Controlador (Servlet).
- Front Controller: un único Servlet que despacha peticiones a handlers específicos.
- Transferencia de datos entre controlador y vista mediante atributos de request y session.
- Organización de paquetes: controller, model, dao, service, util.
- Validación de formularios en el controlador y mensajes de error en la vista.
- Laboratorio: CRUD de tareas con MVC completo (crear, listar, editar, eliminar).

Módulo 7 — Acceso a datos desde la capa web

- JDBC en aplicaciones web: problemas del DriverManager en entornos multi-hilo.
- DataSource y JNDI: configuración de pool de conexiones en Tomcat (context.xml) y lookup con InitialContext.
- Patrón DAO en el contexto web: inyección del DataSource en el DAO.
- PreparedStatement para prevención de SQL Injection en formularios web.
- Manejo de transacciones en operaciones multi-tabla desde el controlador.
- Laboratorio: persistencia de tareas en PostgreSQL mediante DAOs con DataSource de Tomcat.

Módulo 8 — Seguridad, errores, despliegue y proyecto final

- Seguridad declarativa en web.xml: , roles y login-config (FORM-based auth).
- Manejo de errores: página de error global en web.xml para códigos HTTP y excepciones.
- Empaquetado WAR con Maven: dependencias con scope provided, war:war goal.
- Despliegue en Apache Tomcat (webapps) y en WildFly (standalone/deployments).
- Variables de entorno y configuración externalizada para distintos ambientes (dev/prod).
- Proyecto final: sistema To-Do con login, sesión, CRUD de tareas, filtros por estado y persistencia en base de datos.

Evaluación

- Laboratorios prácticos por módulo (40%).
- Quiz de HTTP, Servlets y ciclo de vida MVC (20%).
- Proyecto final sistema To-Do con autenticación y persistencia (40%).

Recursos y herramientas

- JDK 21 LTS, Apache Tomcat 10.x o WildFly 30+.
- IntelliJ IDEA Ultimate (o Community con plugin Tomcat) / Eclipse IDE for Enterprise Java.
- Maven 3.9+, Jakarta EE 10 Web Profile API (jakarta.servlet-api, jakarta.servlet.jsp-api, jstl).

- PostgreSQL 16 como base de datos de práctica.
- Claude.ai para interpretar errores del servidor, generar configuraciones XML y revisar flujos HTTP.

Especialización Java Enterprise Edition

CDI, EJB, JPA y Persistencia Empresarial

Nivel: Intermedio | Código: UNW-JEE-201 | Duración: 20 horas académicas

uneweb.edu.ve
Programa de Formación en Desarrollo Empresarial

Java EE — Nivel II

CDI, EJB, JPA y Persistencia Empresarial

Presentación del curso

El Nivel II eleva al participante desde la capa web básica hacia el núcleo de la plataforma Jakarta EE: la inyección de dependencias con CDI (Contexts and Dependency Injection), los componentes de negocio con EJB (Enterprise JavaBeans), la persistencia de objetos Java en bases de datos relacionales mediante JPA (Jakarta Persistence API) con Hibernate como proveedor, y las transacciones gestionadas por el contenedor (CMT). Al finalizar, el participante podrá construir aplicaciones empresariales por capas bien definidas, totalmente gestionadas por el contenedor Jakarta EE.

Objetivo general

Capacitar al participante en el uso de CDI, EJB y JPA para construir la capa de negocio y persistencia de aplicaciones empresariales Jakarta EE, con gestión de transacciones y ciclo de vida administrado por el contenedor.

Objetivos específicos

- Comprender el modelo de inyección de dependencias de CDI: beans, ámbitos, calificadores e interceptores.
- Crear y utilizar EJBs de sesión sin estado (Stateless), con estado (Stateful) y singleton.
- Modelar entidades JPA con anotaciones: @Entity, @Table, @Id, @GeneratedValue, @Column y tipos de relación.
- Mapear relaciones entre entidades: @OneToOne, @OneToMany, @ManyToOne, @ManyToMany con fetch y cascade.
- Realizar consultas con JPQL, Criteria API y consultas con nombre (@NamedQuery).
- Gestionar transacciones con CMT (Container-Managed Transactions) y BMT (Bean-Managed Transactions).
- Configurar la unidad de persistencia (persistence.xml) y el DataSource JTA.
- Implementar el patrón Repository con JPA para abstraer el acceso a datos.

Requisitos previos

- Haber aprobado Java EE Nivel I (UNW-JEE-101).
- Dominio de JDBC, POO avanzada y patrones básicos de diseño (DAO, MVC).

Metodología

El proyecto integrador es un sistema de gestión de recursos humanos (empleados, departamentos, cargos y nóminas) construido con arquitectura de tres capas: capa web (Servlets/JSF básico), capa de negocio (EJBs) y capa de persistencia (JPA + Hibernate). Claude.ai apoya la configuración de persistence.xml, la interpretación de excepciones de JPA y la revisión de consultas JPQL.

Plan de contenidos (20 horas académicas)

Módulo	Tema	Horas
1	CDI: beans gestionados, ámbitos (@RequestScoped, @SessionScoped, @ApplicationScoped), @Inject y calificadore	3
2	CDI avanzado: productores (@Produces), interceptores (@Interceptor), decoradores y eventos CDI	2
3	EJB de sesión: @Stateless, @Stateful, @Singleton; interfaces local/remota y ciclo de vida del contenedor	3
4	JPA I: entidades, EntityManager, unidad de persistencia, ciclo de vida de entidades y operaciones CRUD	3
5	JPA II: mapeo de relaciones (@OneToMany, @ManyToMany), fetch LAZY/EAGER y estrategias de cascade	3
6	JPQL y Criteria API: consultas de selección, parámetros, JOIN FETCH, agregaciones y paginación	2
7	Transacciones: CMT con @TransactionAttribute, BMT con UserTransaction, rollback y propagación	2
8	Patrón Repository con JPA, integración CDI-EJB-JPA y proyecto final de RRHH	2

Detalle de módulos

Módulo 1 — CDI: Inyección de Dependencias

- ¿Por qué CDI? Desacoplamiento, testabilidad y gestión del ciclo de vida por el contenedor.
- Bean CDI: cualquier clase POJO con constructor sin argumentos (o @Inject en constructor).
- Ámbitos: @RequestScoped (por petición HTTP), @SessionScoped (por sesión), @ApplicationScoped (singleton), @Dependent (por defecto).
- @Inject: inyección en campos, métodos setter y constructores.
- Calificadores (@Qualifier): desambiguación cuando hay múltiples implementaciones de la misma interfaz.
- @Named: nombre para acceso desde EL en JSP/Facelets; @Vetoed: excluir bean del contenedor CDI.

Módulo 2 — CDI avanzado: productores, interceptores y eventos

- @Produces: métodos y campos productores para tipos no gestionados por CDI (Logger, EntityManager, propiedades).
- @Disposes: liberación de recursos producidos.
- Interceptores: @InterceptorBinding para logging, seguridad y métricas de rendimiento de forma transversal.
- @Priority y orden de interceptores; interceptores de ciclo de vida (@PostConstruct, @PreDestroy).
- Decoradores (@Decorator): patrón Decorator gestionado por CDI para extender comportamiento.
- Sistema de eventos CDI: @Observes para comunicación desacoplada entre beans; eventos síncronos y asíncronos (@ObservesAsync).

Módulo 3 — Enterprise JavaBeans (EJB)

- ¿Qué es un EJB? Componente de negocio gestionado por el contenedor EJB: pooling, seguridad, transacciones.
- @Stateless Session Bean: sin estado conversacional, pool de instancias, ideal para servicios sin estado.
- @Stateful Session Bean: estado conversacional por cliente; @Remove para destruir la instancia.
- @Singleton: instancia única con @Startup para inicialización al desplegar; @Lock(READ/WRITE).
- Interfaces @Local y @Remote; EJBs en diferentes JVMs con JNDI lookup remoto.
- Ciclo de vida de EJBs: @PostConstruct, @PreDestroy, @PrePassivate, @PostActivate (Stateful).

Módulo 4 — JPA I: Entidades y EntityManager

- ORM (Object-Relational Mapping): por qué JPA sobre JDBC directo; Hibernate como proveedor.
- Configuración: persistence.xml, unidad de persistencia, datasource JTA y propiedades de Hibernate.
- @Entity, @Table, @Column, @Id, @GeneratedValue (IDENTITY, SEQUENCE, TABLE, AUTO).
- Tipos de datos básicos, tipos enumerados (@Enumerated) y tipos temporales (@Temporal, LocalDate).
- EntityManager: persist(), find(), merge(), remove(), contains(), detach(), flush(), refresh().
- Ciclo de vida de una entidad: New → Managed → Detached → Removed; contexto de persistencia.

Módulo 5 — JPA II: Relaciones y estrategias de carga

- @OneToOne: relaciones bidireccionales y unidireccionales; @JoinColumn y @PrimaryKeyJoinColumn.
- @OneToMany / @ManyToOne: lado propietario y lado inverso (mappedBy); @JoinColumn.
- @ManyToMany: tabla de unión con @JoinTable; bidireccionalidad y el problema del dueño.
- FetchType.LAZY vs EAGER: impacto en rendimiento y el problema N+1 queries.
- CascadeType: PERSIST, MERGE, REMOVE, REFRESH, DETACH, ALL; cuándo y cuándo no usar CASCADE.REMOVE.
- @Embeddable y @Embedded: tipos de valor embebidos (Dirección, Dinero) sin tabla propia.

Módulo 6 — JPQL y Criteria API

- JPQL: lenguaje orientado a objetos; FROM Entidad e, SELECT, WHERE, JOIN, JOIN FETCH.
- @NamedQuery y @NamedQueries: consultas predefinidas compiladas al desplegar.
- Parámetros posicionales (?1) y nombrados (:nombre); paginación con setFirstResult/setMaxResults.
- Agregaciones: COUNT, SUM, AVG, MIN, MAX; GROUP BY y HAVING en JPQL.
- Subqueries, IN, EXISTS, NOT EXISTS en JPQL.
- Criteria API: consultas tipadas en tiempo de compilación con CriteriaBuilder, CriteriaQuery, Root y Predicate.

Módulo 7 — Gestión de transacciones

- Transacciones ACID en el contexto empresarial; transacciones JTA vs. RESOURCE_LOCAL.
- CMT (Container-Managed Transactions): @TransactionAttribute con valores REQUIRED, REQUIRES_NEW, SUPPORTS, NOT_SUPPORTED, MANDATORY, NEVER.
- Propagación de transacciones entre EJBs anidados; contexto de persistencia transaccional.
- BMT (Bean-Managed Transactions): UserTransaction.begin(), commit(), rollback(); cuándo usar BMT.
- Manejo de excepciones y rollback: @ApplicationException(rollback=true) vs. SystemException.
- Optimistic Locking con @Version: detección de conflictos concurrentes; manejo de OptimisticLockException.

Módulo 8 — Repository, integración y proyecto final

- Patrón Repository: abstracción sobre EntityManager con tipo genérico AbstractRepository.
- Integración CDI → EJB → JPA → Base de datos: flujo completo de una operación de negocio.
- @Inject EntityManager en EJBs gestionados por el contenedor; contexto de persistencia extendido en @Stateful.
- Pruebas unitarias de EJBs con Arquillian (contenedor embebido) o Mockito.
- Proyecto final: sistema de RRHH — entidades Empleado, Departamento, Cargo, Nomina con relaciones JPA, servicios EJB, CDI en la capa web y reportes con JPQL.

Evaluación

- Laboratorios prácticos por módulo (40%).
- Quiz de CDI, EJB y JPA (20%).
- Proyecto final sistema de RRHH en tres capas (40%).

Recursos y herramientas

- JDK 21 LTS, WildFly 30+ o Payara Server 6 (servidores de aplicaciones Jakarta EE 10 Full Profile).
- Hibernate ORM 6.x como proveedor JPA.
- Maven 3.9+, IntelliJ IDEA Ultimate o Eclipse Enterprise Edition.
- PostgreSQL 16 con pool de conexiones vía DataSource JTA.
- Claude.ai para configuración de persistence.xml, interpretación de excepciones JPA y revisión de JPQL.

Especialización Java Enterprise Edition

REST con JAX-RS, Jakarta Faces, Seguridad y Mensajería JMS

Nivel: Avanzado | Código: UNW-JEE-301 | Duración: 20 horas académicas

uneweb.edu.ve
Programa de Formación en Desarrollo Empresarial

Java EE — Nivel III

REST con JAX-RS, Jakarta Faces, Seguridad y Mensajería JMS

Presentación del curso

El Nivel III abarca las especificaciones más demandadas del ecosistema Jakarta EE en el desarrollo moderno: la construcción de APIs RESTful con JAX-RS, interfaces de usuario ricas con Jakarta Server Faces (JSF/Facelets), seguridad empresarial declarativa y programática con Jakarta Security, mensajería asíncrona con JMS (Jakarta Messaging) y cómo integrar estos componentes en una arquitectura coherente. El participante estará preparado para diseñar sistemas empresariales completos con comunicación síncrona y asíncrona.

Objetivo general

Capacitar al participante en el diseño y construcción de APIs REST con JAX-RS, interfaces de usuario empresariales con Jakarta Faces, seguridad basada en estándares Jakarta EE y comunicación asíncrona con JMS, integrando estas tecnologías en aplicaciones empresariales completas.

Objetivos específicos

- Construir APIs RESTful con JAX-RS: recursos, métodos HTTP, negociación de contenido JSON/XML y manejo de excepciones.
- Validar datos de entrada en APIs REST con Jakarta Bean Validation (@NotNull, @Valid, ConstraintValidator).
- Documentar APIs con MicroProfile OpenAPI o Jakarta RESTful Web Services y consumir APIs externas con JAX-RS Client API.
- Crear interfaces de usuario ricas con Jakarta Server Faces (JSF) usando Facelets, componentes PrimeFaces y backing beans CDI.
- Implementar navegación, conversores, validadores y el ciclo de vida de JSF.
- Aplicar seguridad declarativa y programática: autenticación FORM/JWT con Jakarta Security y autorización basada en roles.
- Enviar y recibir mensajes de forma asíncrona con Jakarta Messaging (JMS): queues, topics y Message-Driven Beans (MDB).
- Diseñar patrones de integración básicos: fire-and-forget, request-reply y pub/sub con JMS.

Requisitos previos

- Haber aprobado Java EE Nivel II (UNW-JEE-201).
- Dominio de CDI, EJB y JPA; familiaridad con el protocolo HTTP y REST.

Metodología

El proyecto integrador es una plataforma de gestión de pedidos en línea con: una API REST JAX-RS consumida por una interfaz JSF/PrimeFaces, seguridad JWT para la API y FORM para la interfaz web, y procesamiento asíncrono de pedidos mediante colas JMS con MDBs. Claude.ai se utiliza para diseñar contratos REST, revisar configuraciones de seguridad y depurar el ciclo de vida de JSF.

Plan de contenidos (20 horas académicas)

Módulo	Tema	Horas
1	JAX-RS I: @Path, @GET/@POST/@PUT/@DELETE, @Produces/@Consumes, Response y manejo de parámetros	3
2	JAX-RS II: Bean Validation, ExceptionMapper, filtros JAX-RS, Client API y consumo de APIs externas	2
3	Jakarta Server Faces (JSF) I: Facelets, backing beans CDI, ciclo de vida JSF y navegación	3
4	JSF II: conversores, validadores, PrimeFaces DataTable/Dialog/InputText y AJAX con f:ajax	3
5	Jakarta Security: autenticación FORM, HttpAuthenticationMechanism, IdentityStore y SecurityContext	2
6	JWT y API segura: generación de tokens JWT, @RolesAllowed en JAX-RS y filtro de autenticación	2
7	Jakarta Messaging (JMS): arquitectura, ConnectionFactory, Queue, Topic, MessageProducer y MessageConsumer	2
8	Message-Driven Beans (MDB), patrones de mensajería y proyecto final de plataforma de pedidos	3

Detalle de módulos

Módulo 1 — JAX-RS I: Recursos REST fundamentales

- Principios REST: recursos, representaciones, verbos HTTP, stateless, HATEOAS básico.
- @Path en clase e método; @PathParam, @QueryParam, @HeaderParam, @FormParam, @BeanParam.
- @Produces y @Consumes: negociación de contenido JSON (application/json) y XML.
- Clase Response: status codes semánticos (200, 201, 204, 400, 404, 409, 500), headers y entidad.
- Serialización/deserialización JSON con Jakarta JSON Binding (JSON-B) o Jackson.
- Laboratorio: API REST de productos (CRUD) con respuestas tipadas y códigos HTTP correctos.

Módulo 2 — JAX-RS II: Validación, filtros y cliente

- Jakarta Bean Validation en JAX-RS: @Valid en parámetros y cuerpo; @NotNull, @Size, @Email, ConstraintValidator personalizado.
- ExceptionMapper: mapeo de excepciones a respuestas HTTP con cuerpo de error estructurado.
- ContainerRequestFilter y ContainerResponseFilter: logging, CORS, compresión.
- JAX-RS Client API: ClientBuilder, WebTarget, Invocation.Builder; llamadas GET/POST a APIs externas.
- Procesamiento asíncrono en JAX-RS: @Suspended AsyncResponse para peticiones de larga duración.

Módulo 3 — Jakarta Server Faces I

- Arquitectura JSF: FacesServlet, árbol de componentes, contexto Faces y ciclo de vida de 6 fases.
- Facelets: plantillas con , , ; páginas maestras.
- Backing beans CDI: @Named + @ViewScoped, @SessionScoped; métodos de acción y action listener.
- Componentes básicos: , , , .
- Navegación implícita (return "vista") y explícita con faces-config.xml.
- Laboratorio: formulario de alta de pedido con validación y listado de pedidos.

Módulo 4 — Jakarta Server Faces II: PrimeFaces y AJAX

- PrimeFaces 13: integración vía dependencia Maven; namespace xmlns:p.
- : paginación, ordenación, filtrado, selección de filas y edición en línea.
- y : ventanas modales para formularios de edición.

- , , : componentes de entrada avanzados.
- AJAX con y : execute, render, listener y actualización parcial de la página.
- Conversores (@FacesConverter) y validadores (@FacesValidator) personalizados.

Módulo 5 — Jakarta Security: Autenticación y autorización

- Jakarta Security 3.0: HttpAuthenticationMechanism, IdentityStore, SecurityContext.
- Autenticación FORM personalizada con @FormAuthenticationMechanismDefinition.
- IdentityStore: implementación de base de datos para validar credenciales (PBKDF2PasswordHash).
- Autorización declarativa: @DenyAll, @PermitAll, @RolesAllowed en EJBs y JAX-RS.
- SecurityContext programático: isCallerInRole(), getCallerPrincipal(), hasAccessToWebResource().
- Configuración de HTTPS básico en WildFly/Payara para producción.

Módulo 6 — JWT y autenticación de APIs REST

- ¿Por qué JWT para APIs REST? Diferencias con autenticación de sesión en aplicaciones web.
- Estructura de un JWT: header.payload.signature; claims estándar (iss, sub, exp, iat, roles).
- Generación de JWT con la librería nimbus-jose-jwt o MicroProfile JWT.
- ContainerRequestFilter de autenticación: extraer y validar Bearer token en cada petición.
- @RolesAllowed en endpoints JAX-RS integrado con el contexto de seguridad del token.
- Refresh tokens: estrategia de renovación sin re-autenticación; revocación básica.

Módulo 7 — Jakarta Messaging (JMS)

- Arquitectura de mensajería: broker (ActiveMQ Artemis), ConnectionFactory, Destination (Queue/Topic).
- Configuración de recursos JMS en WildFly: administración de queues y topics.
- JMSContext (API simplificada JMS 2.0): createProducer(), createConsumer(), send(), receive().
- Tipos de mensaje: TextMessage, ObjectMessage, MapMessage, BytesMessage.
- Punto a punto (Queue) vs. publicación-suscripción (Topic); durable subscribers.
- Laboratorio: envío de notificación de nuevo pedido a una queue JMS.

Módulo 8 — MDBs, patrones de mensajería y proyecto final

- Message-Driven Bean (@MessageDriven): consumidor asíncrono gestionado por el contenedor; @ActivationConfigProperty.
- Integración MDB + EJB + JPA: procesamiento de pedido recibido en queue → servicio de negocio → persistencia.
- Patrón fire-and-forget: confirmación inmediata al cliente y procesamiento en background.
- Patrón request-reply con JMSReplyTo y TemporaryQueue para respuestas asíncronas.
- Reintentos y Dead Letter Queue (DLQ): manejo de mensajes que fallan repetidamente.
- Proyecto final: plataforma de pedidos con API JAX-RS, interfaz JSF/PrimeFaces, autenticación FORM+JWT y procesamiento asíncrono de pedidos con MDB.

Evaluación

- Laboratorios prácticos por módulo (40%).
- Quiz de JAX-RS, JSF y JMS (20%).
- Proyecto final plataforma de pedidos con API REST, UI JSF y mensajería asíncrona (40%).

Recursos y herramientas

- JDK 21 LTS, WildFly 30+ con ActiveMQ Artemis embebido o Payara Server 6.
- Jakarta EE 10 Full Profile: JAX-RS, JSF, CDI, EJB, JPA, JMS, Jakarta Security.

- PrimeFaces 13.x para la capa de presentación JSF.
- nimbus-jose-jwt o MicroProfile JWT para generación y validación de tokens.
- Claude.ai para diseño de contratos REST, revisión de flujo de seguridad JWT y depuración del ciclo de vida JSF.

Especialización Java Enterprise Edition

MicroProfile, Microservicios, Contenedores y CI/CD Empresarial

Nivel: Especialización / Profesional | Código: UNW-JEE-401 | Duración: 20 horas académicas

uneweb.edu.ve
Programa de Formación en Desarrollo Empresarial

Java EE — Nivel IV

MicroProfile, Microservicios, Contenedores y CI/CD Empresarial

Presentación del curso

El Nivel IV cierra la especialización con las prácticas y tecnologías que definen el desarrollo empresarial moderno: el estándar Eclipse MicroProfile para microservicios Jakarta EE, contenerización con Docker y orquestación básica con Kubernetes, comunicación entre servicios con REST y mensajería, resiliencia y tolerancia a fallos, observabilidad (métricas, trazas, logs estructurados) y el ciclo completo de integración y despliegue continuo (CI/CD) con Jenkins o GitHub Actions. Al finalizar, el participante podrá diseñar, desplegar y operar sistemas empresariales distribuidos basados en Jakarta EE y MicroProfile.

Objetivo general

Formar al participante en la arquitectura de microservicios con MicroProfile y Jakarta EE, incluyendo contenerización, orquestación básica, patrones de resiliencia, observabilidad y automatización del ciclo de vida mediante pipelines CI/CD, para operar aplicaciones empresariales en entornos cloud.

Objetivos específicos

- Diseñar microservicios Jakarta EE utilizando el perfil MicroProfile con Quarkus o Open Liberty.
- Implementar configuración externalizada con MicroProfile Config y gestión de secretos.
- Aplicar patrones de resiliencia con MicroProfile Fault Tolerance: @Retry, @CircuitBreaker, @Fallback, @Bulkhead, @Timeout.
- Exponer y consumir métricas con MicroProfile Metrics y OpenTelemetry para trazabilidad distribuida.
- Documentar APIs automáticamente con MicroProfile OpenAPI.
- Contenerizar aplicaciones Jakarta EE con Docker: Dockerfile multistage, imágenes base Quarkus/OpenLiberty.
- Orquestar contenedores con Kubernetes: Deployments, Services, ConfigMaps, Secrets e Ingress básico.
- Diseñar e implementar un pipeline CI/CD completo con GitHub Actions o Jenkins para build, test y despliegue automático.

Requisitos previos

- Haber aprobado Java EE Nivel III (UNW-JEE-301).
- Nociones básicas de Linux, línea de comandos y redes.
- Conocimientos básicos de Git y GitHub.

Metodología

El proyecto integrador descompone la plataforma de pedidos del Nivel III en tres microservicios independientes: Servicio de Catálogo, Servicio de Pedidos y Servicio de Notificaciones. Cada módulo añade una capacidad (resiliencia, métricas, contenedor, orquestación, pipeline) hasta culminar con un sistema distribuido completamente automatizado y observable. Claude.ai apoya el diseño de Dockerfiles, la creación de manifiestos Kubernetes y la revisión de pipelines CI/CD.

Plan de contenidos (20 horas académicas)

Módulo	Tema	Horas
1	Arquitectura de microservicios: principios, beneficios, retos y comparación con monolito modular	2
2	MicroProfile I: Config, Health, JWT Propagation; Quarkus o Open Liberty como runtime MicroProfile	3
3	MicroProfile II: Fault Tolerance (@Retry, @CircuitBreaker, @Fallback, @Bulkhead, @Timeout)	2
4	MicroProfile III: Metrics (Counters, Timers, Gauges), OpenTelemetry y trazabilidad distribuida	3
5	MicroProfile OpenAPI y REST Client: documentación automática y comunicación tipada entre servicios	2
6	Docker: Dockerfile multistage, imagen Quarkus/OpenLiberty, docker-compose para entorno local	3
7	Kubernetes: Pod, Deployment, Service, ConfigMap, Secret, Ingress y kubectl básico	3
8	CI/CD con GitHub Actions / Jenkins: pipeline build-test-push-deploy y proyecto final distribuido	2

Detalle de módulos

Módulo 1 — Arquitectura de microservicios

- Monolito vs. microservicios vs. monolito modular: trade-offs reales de cada arquitectura.
- Principios de microservicios: responsabilidad única, autonomía, descentralización de datos, design for failure.
- Comunicación entre servicios: síncrona (REST, gRPC) vs. asíncrona (mensajería JMS/Kafka).
- Desafíos: consistencia eventual, descubrimiento de servicios, latencia de red y complejidad operacional.
- El papel de Jakarta EE y MicroProfile en el ecosistema de microservicios Java.
- Estrategia de descomposición: Strangler Fig Pattern para migrar un monolito Jakarta EE.

Módulo 2 — MicroProfile Config, Health y JWT Propagation

- Eclipse MicroProfile: historia, especificaciones y runtimes compatibles (Quarkus, Open Liberty, Payara Micro, Helidon).
- Instalación y configuración de Quarkus: quarkus-maven-plugin, modo dev con hot reload.
- MicroProfile Config: fuentes de configuración (microprofile-config.properties, variables de entorno, System Properties), @ConfigProperty, Config programático.
- MicroProfile Health: @Liveness y @Readiness para sondas de Kubernetes; endpoint /health.
- MicroProfile JWT Propagation: propagación del token JWT entre microservicios; @Claim para extraer claims.
- Laboratorio: servicio de catálogo con configuración externalizada y endpoints de health.

Módulo 3 — MicroProfile Fault Tolerance

- El problema de la resiliencia en sistemas distribuidos: fallos de red, timeouts y cascadas de errores.
- @Timeout: interrumpir operaciones que superan un límite de tiempo; TimeoutException.
- @Retry: reintentos automáticos con delay, jitter y maxRetries; retryOn y abortOn.
- @Fallback: método o clase alternativa cuando todos los reintentos fallan.
- @CircuitBreaker: estados CLOSED → OPEN → HALF_OPEN; requestVolumeThreshold y failureRatio.
- @Bulkhead: limitación de concurrencia (semáforo) y cola de espera; aislamiento de fallos.

Módulo 4 — Métricas y trazabilidad distribuida

- MicroProfile Metrics 5.x: tipos de métrica — Counter, Timer, Gauge, Histogram, ConcurrentGauge.

- @Counted, @Timed, @Gauge: anotaciones para instrumentación declarativa.
- Exposición de métricas en formato Prometheus (/metrics); integración con Grafana.
- OpenTelemetry: SDK de Java, TracerProvider, Span y propagación de contexto entre servicios HTTP.
- Instrumentación automática con el agente Java de OpenTelemetry; exportación a Jaeger/Zipkin.
- Logs estructurados (JSON) con JBoss Logging o SLF4J + Logback; correlación con traceId.

Módulo 5 — MicroProfile OpenAPI y REST Client

- MicroProfile OpenAPI 3.1: generación automática de la especificación desde anotaciones JAX-RS.
- @Operation, @APIResponse, @Parameter, @Schema: documentación enriquecida de endpoints.
- UI Swagger integrada en Quarkus (/q/swagger-ui) y Open Liberty (/openapi/ui).
- MicroProfile REST Client: interfaz tipada para consumir APIs REST externas; @RegisterRestClient, @RestClient.
- @ClientHeaderParam y propagación de cabeceras de autenticación entre servicios.
- Laboratorio: servicio de pedidos que consume el servicio de catálogo con REST Client tipado.

Módulo 6 — Docker y contenerización

- Conceptos de Docker: imagen, contenedor, capa (layer), volumen, red y registro.
- Dockerfile para aplicaciones Quarkus: multistage build (build JDK → run JRE mínimo); imagen distroless.
- Modo nativo de Quarkus con GraalVM: compilación a ejecutable nativo; tiempo de arranque < 100ms.
- docker-compose.yml: orquestación local de múltiples servicios (catálogo, pedidos, notificaciones, PostgreSQL, ActiveMQ).
- Publicación de imágenes en Docker Hub o GitHub Container Registry (GHCR).
- Buenas prácticas: usuario no root en contenedor, .dockerignore, escaneo de vulnerabilidades con Trivy.

Módulo 7 — Kubernetes

- Arquitectura de Kubernetes: Control Plane (API Server, Scheduler, etcd) y Worker Nodes (kubelet, kube-proxy).
- Pod: unidad mínima de despliegue; Deployment: réplicas, rolling update y rollback.
- Service: ClusterIP, NodePort, LoadBalancer; resolución DNS interna entre pods.
- ConfigMap y Secret: externalización de configuración y credenciales; montaje como volumen o variable de entorno.
- Ingress: enrutamiento HTTP/HTTPS externo; anotaciones NGINX; TLS con cert-manager.
- Horizontal Pod Autoscaler (HPA): escalado automático basado en CPU/memoria; kubectl top.

Módulo 8 — CI/CD y proyecto final distribuido

- GitHub Actions: workflow YAML; triggers (push, pull_request, schedule); jobs y steps.
- Pipeline completo: checkout → build Maven → unit tests → build Docker → push GHCR → deploy Kubernetes (kubectl apply).
- Gestión de secretos en GitHub Actions: DOCKER_USERNAME, KUBECONFIG como repository secrets.
- Jenkins (alternativa): Jenkinsfile declarativo; agentes Docker; etapas paralelas.
- Pruebas de integración en el pipeline: Testcontainers para levantar PostgreSQL y ActiveMQ en CI.
- Proyecto final: sistema distribuido de tres microservicios (Catálogo, Pedidos, Notificaciones) con Fault Tolerance, métricas Prometheus, imágenes Docker, despliegue en Kubernetes y pipeline GitHub Actions completamente automatizado.

Evaluación

- Implementaciones de resiliencia, métricas y contenedores por módulo (30%).
- Quiz de MicroProfile, Docker y Kubernetes (20%).
- Proyecto final sistema distribuido con CI/CD automatizado (50%).

Recursos y herramientas

- JDK 21 LTS, Quarkus 3.x o Open Liberty 24.x como runtime MicroProfile.
- GraalVM CE 21 para compilación nativa (opcional).
- Docker Desktop o Podman, kubectl y minikube o kind para clúster Kubernetes local.
- GitHub Actions para CI/CD; Jaeger para trazabilidad; Prometheus + Grafana para métricas.
- Testcontainers 1.x para pruebas de integración en pipeline.
- Claude.ai para diseño de Dockerfiles multistage, manifiestos Kubernetes y pipelines CI/CD.