Ministerio de Educación Superior, Ciencia y Tecnología (Mescyt)

Centro de Tecnología y Educación Permanente TEP

Pontificia Universidad Católica Madre y Maestra (PUCMM)

Diplomado en Programación en Lenguaje JAVA

Laboratorio 03

Módulo III

Introducción a Java Enterprise Edition 7

Facilitador: Ing. Eudris Cabrera

Fecha : 6 Agosto del 2016

Santiago de los Caballeros, República Dominicana

Introducción

Entendiendo el ecosistema Java

Plataforma Java:

- Multi-plataforma.
    - Utiliza una máquina virtual para su ejecución (JVM)
    - Está dividida en:
        - Java SE
        - Java EE
        - Java ME
        - JavaFX
- El estándar es manejado por Java Community Process (JCP)

Diferencia entre JAVA SE y JAVA EE

JAVA SE es la versión estándar de java. Es la api base del lenguaje mientras que JAVA EE podríamos decir que es una versión extendida de JAVA SE.

La plataforma Java EE consta de un conjunto de servicios, API y protocolos que proporcionan la funcionalidad necesaria para desarrollar aplicaciones basadas en la web de varios niveles.

Java EE simplifica el desarrollo de aplicaciones y reduce la necesidad de programación, al proporcional componentes modulares normalizados y reutilizables, así como al permitir controlar muchos aspectos de la programación automáticamente por nivel.

Java Platform, Enterprise Edition (EE)

Es un entorno independiente de la plataforma centrado en Java para desarrollar, crear e implementar en línea aplicaciones empresariales basadas en web.

Es el estándar en software empresarial. Se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

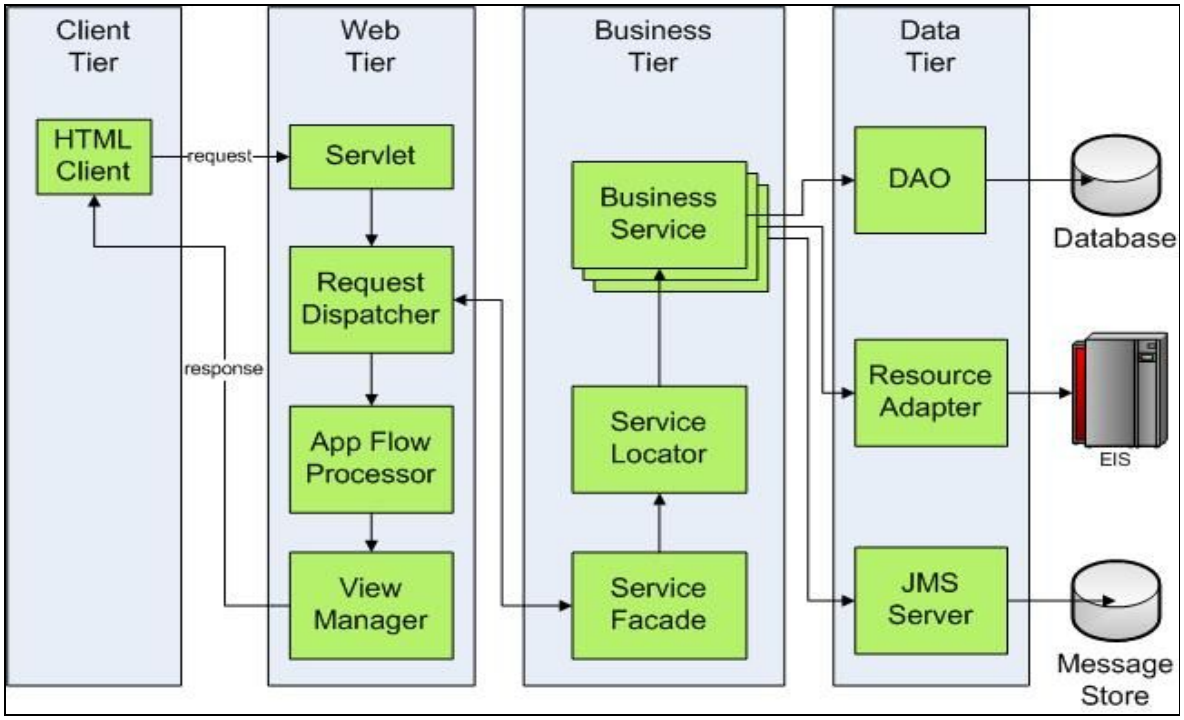
Las razones que empujan a la creación de la plataforma Java EE:

- Programación eficiente.
- Extensibilidad frente a la demanda del negocio.
- Integración.

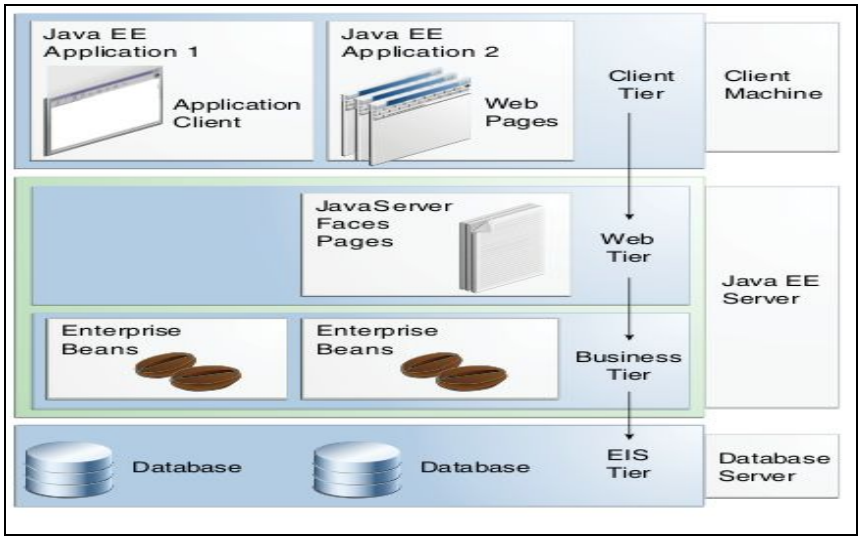
Arquitectura de Java EE

La arquitectura Java EE implica un modelo de aplicaciones distribuidas en diversas capas o niveles (tier).

Cliente	HTML, Applet, aplicaciones Java, etc.
Web	JSF, JSP, Servlet
Negocio	EJB, JPA, JAX-WS y JAX-RS Web services
Sistema de Información Empresariales	JDBC, JTA, Java EE Connector, JPA



Estructura de una Aplicación Empresarial



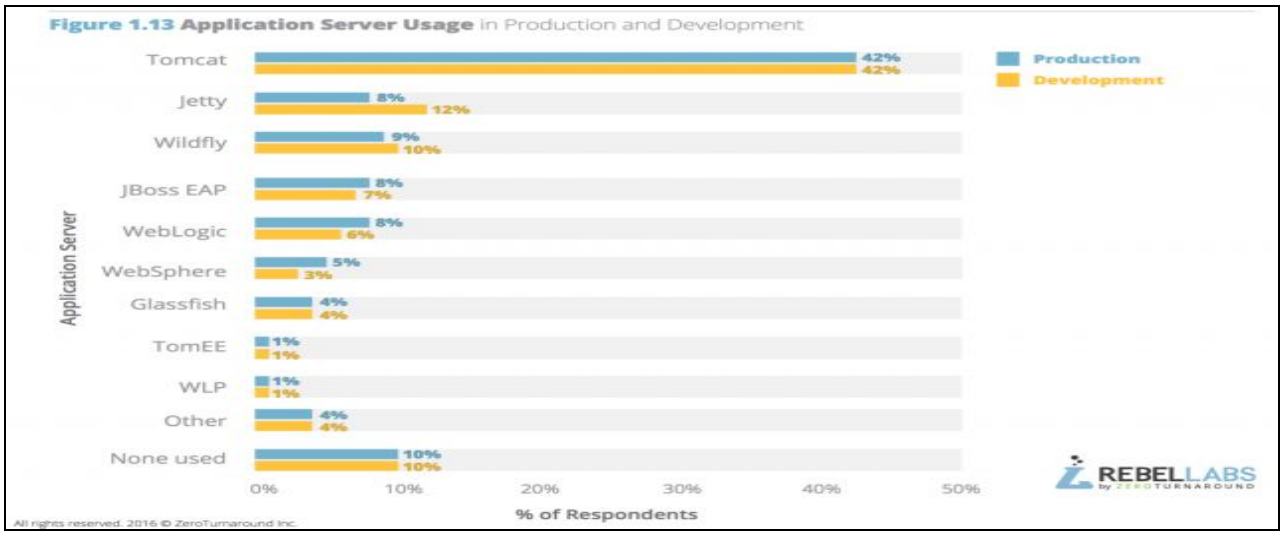
Servidores Java EE

Es un servidor de aplicaciones que implementa los APIs de la plataforma Java EE y provee los servicios del estándar Java EE.

Los servidores Java EE muchas veces son llamados servidores de aplicaciones, porque permiten servir datos a los clientes, de la misma forma que un servidor web permite servir páginas web a un browser.

Un servidor Java EE puede alojar varios tipos de componentes correspondientes a una aplicación multi-capas. En este sentido, ofrece un entorno de ejecución estandarizado para estos componentes.

Popularidad



Contenedores Java EE

Son la interfaz entre el componente y la funcionalidad de bajo nivel proporcionada por la plataforma para soportar ese componente.

La funcionalidad del contenedor está definido por la plataforma, y es diferente para cada tipo de componente.

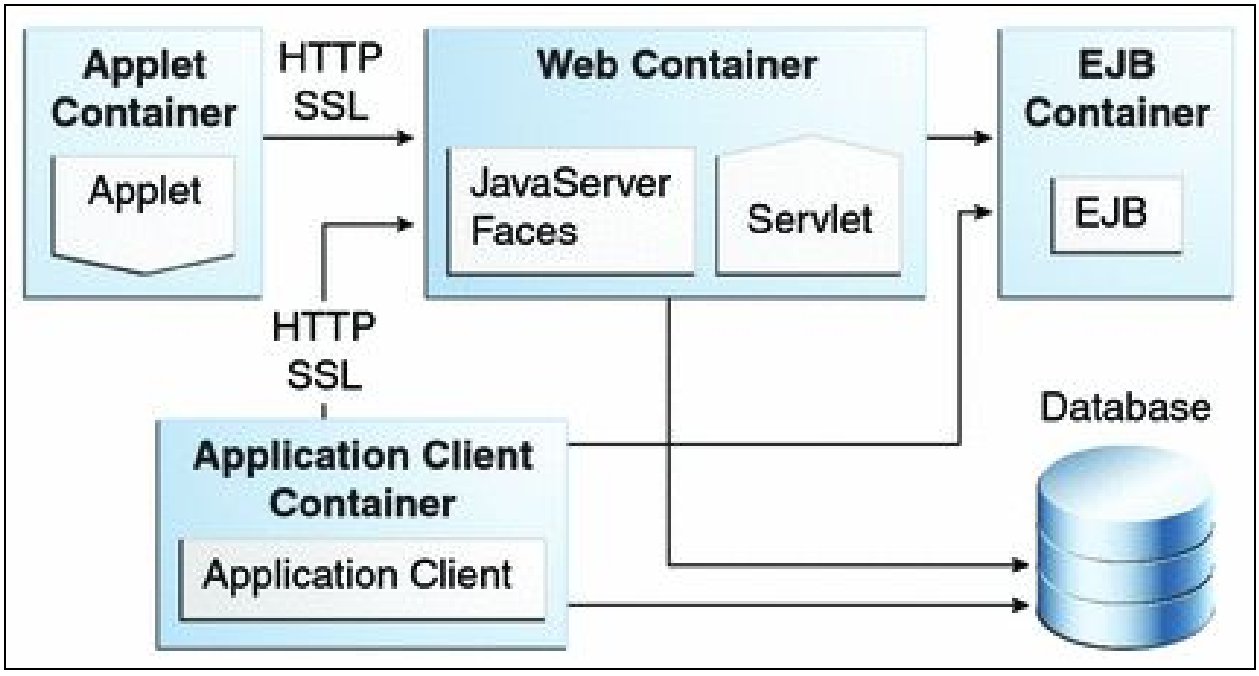
No obstante, el servidor permite que los diferentes tipos de componentes puedan trabajar juntos para proporcionar funcionalidad en una aplicación empresarial.

Tipos de contenedores

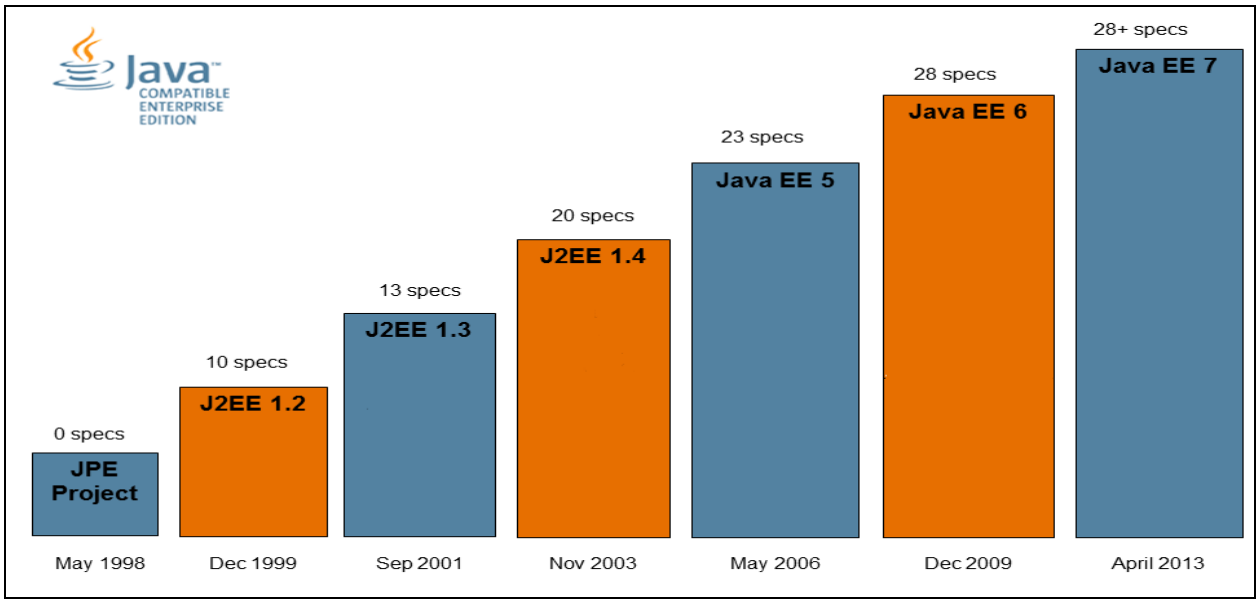
Contenedor Web:
Maneja la ejecución de los Servlets y páginas JSP y JSF.

Contenedor Cliente:
Provee una interfaz de conexión entre el servidor Java EE y las aplicaciones clientes, tales como aplicaciones Java SE, entre otras.

Contenedor EJB:
Gestiona la ejecución de los Enterprise JavaBeans.



Evolución de Java EE



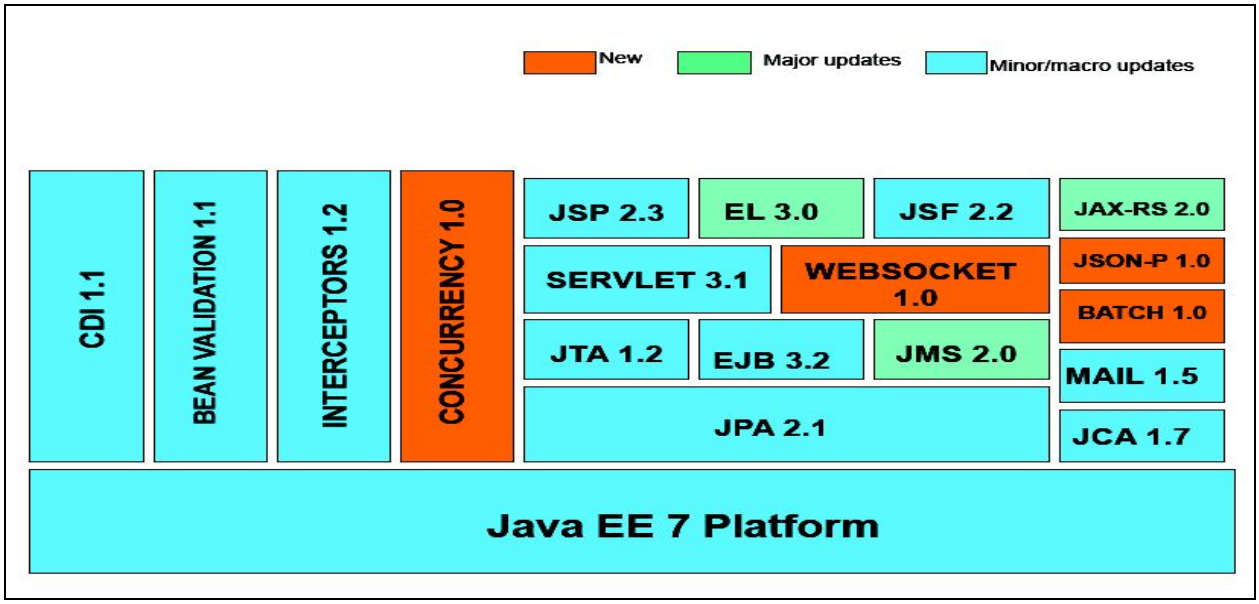
Java EE 6 cambió el juego ....

- Más Ligero.
- Introduce el concepto de perfiles (**Web Profile** and **Full Profile**).
- **EJB** empaquetado en archivos **war**.
- Servlet 3.0
- web.xml (opcional), **@WebServlet, @WebFilter**
- Soporte para servicios web RESTful con **JAX-RS 1.1**
- Contextos e Inyección de Dependencia (CDI) para Java EE

Java EE 7 productividad y enfocado en HTML5

- Construido sobre la base de Java EE6.
- Soporte para HTML5.
- 4 nuevas especificaciones.
- 3 especificaciones con cambios importantes.
- 6 especificaciones con cambios menores.
- 5 especificaciones con cambios micro.

Plataforma Java EE 7



Cómo empezar con Java EE 7 ?

Descargar e instalar:

JDK 8(ó superior):
http://www.oracle.com/technetwork/java/javase/downloads/index.html

Instalar el IDE de su preferencia:

- NetBeans 8.0.2 superior, versión completa o “Java EE”
- http://netbeans.org/downloads/
- Eclipse Luna (4.4.0) o superior
- http://www.eclipse.org/neon/
- Otros

Guía del laboratorio

Este laboratorio está enfocado en introducir los conceptos fundamentales de **Java Enterprise Edition**. Destacando los aspectos más importantes de Java EE 7.

Java EE 7 cubre las siguientes especificaciones:

- Java Persistence API 2.1 (JSR 338)
- Java API for RESTful Web Services 2.0 (JSR 339)
- Java Message Service 2.0 (JSR 343)
- JavaServer Faces 2.2 (JSR 344)
- Contexts and Dependency Injection 1.1 (JSR 346)
- Bean Validation 1.1 (JSR 349)
- Batch Applications for the Java Platform 1.0 (JSR 352) *
- Java API for JSON Processing 1.0 (JSR 353) *
- Java API for WebSocket 1.0 (JSR 356) *
- Java Transaction API 1.2 (JSR 907)

En este laboratorio nos enfocaremos en las especificaciones ~~**Java Persistence API 2.1 (JSR 338)**~~, **Contexts and Dependency Injection 1.1 (JSR 346)** y **Bean Validation 1.1 (JSR 349).**

Escenario

Petcare: Es una aplicación web que permite realizar citas médicas para las mascotas. El cliente o cualquier propietario de una mascota puede ver la agenda de cada médico y solicitar una cita en una determinada fecha, siempre y cuando haya creado una cuenta personal.

Los requerimientos de dicha aplicación son los siguientes:

1- Permitir que cualquier persona que acceda a dicha aplicación pueda crear una cuenta personal.
2- Permitir que un cliente registrado puede ver la agenda de un médico veterinario en específico.
3- Permitir que un cliente registrado puede solicitar una cita en una fecha determinada.
4- Permitir el registro de paciente (mascota) por parte de un cliente (dueño).
5- Acceder a cada una de las opciones a través de un menú.
6- La aplicación debe permitir agregar una foto de la mascota.
7- Autenticación de usuarios (Login de acceso y enlace para salir de la aplicación)
8- Consultar el listado de clientes registrados y poder filtrar por un nombre en específico
9- Consultar el listado de pacientes registrados y poder filtrar por un paciente en específico,
10-Permitir modificar una cita existente.

Requerimientos

Java 1.7+ (Recomendado Java 8)
Netbeans 8.0.1+
Apache Maven 3.2.1+
Glassfish 4.1.1
Mysql Server 5.0+

Tecnología	Java EE
Controlador de aplicación Web	Servlet
Plantilla de interfaz de usuario del lado del servidor	JSP
Servidor de Aplicación	GlassFish

Base de Datos	MySQL
Mapeo objeto-relacional (ORM)	EclipseLink

Configuración de Ambiente

Verificar que en su ambiente se encuentra creada la base de datos **petcare**.
En caso de no estar creada ejecutar el script **sql/mysql/petcare.sql** en una base de datos mysql.

Descargar el proyecto **jsp-twitter-bootstrap** del repositorio principal y ubicado en la ruta **programacion-web-java/recursos**.

Renombrar el proyecto **jsp-twitter-bootstrap** con el nombre **petcareapp-javaee.**

Información del proyecto

Nombre : **petcareapp-javaee**
artifactId: **petcareapp-javaee**
groupId: ***org.diplomado.pucmm.mescyt***
version: ***valor por defecto "1.0-SNAPSHOT"***
package: ***org.diplomado.pucmm.mescyt.java.petcareapp.javaee***

Modificar archivo pom.xml

Java EE

```
<dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-web-api</artifactId>
        <version>7.0</version>
        <scope>provided</scope>
</dependency>
```

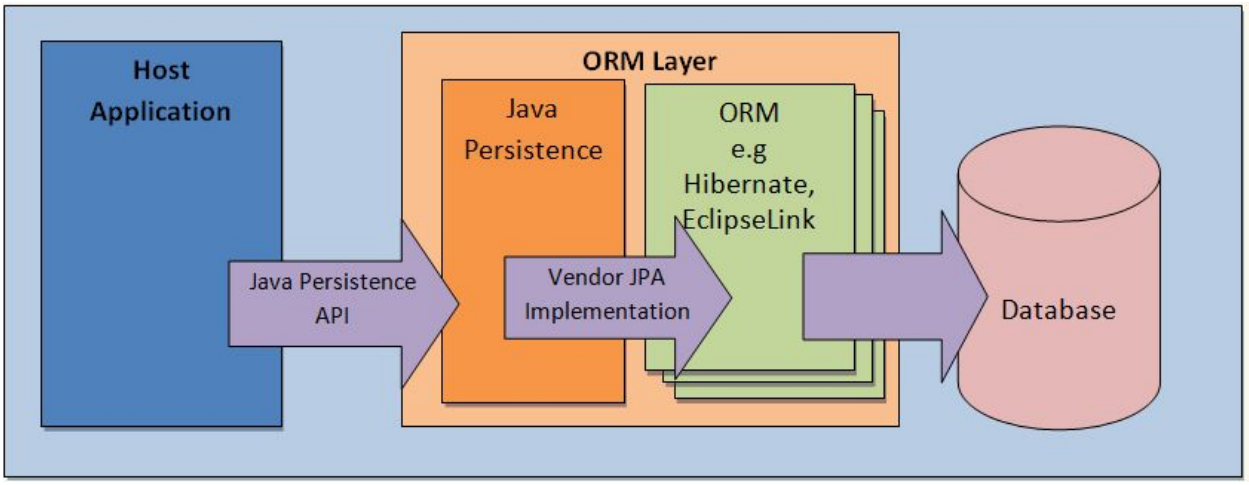
Crear la siguiente estructura de paquetes

org.diplomado.pucmm.mescyt.java.petcareapp.javaee.controller	Para las clases que tienen que ver con la navegación.
org.diplomado.pucmm.mescyt.java.petcareapp.javaee.entidades	Clase que representan el dominio

org.diplomado.pucmm.mescyt.java.petcareapp.javaee.servicios	Para contener las clases que tienen que ver con acceso a datos.
org.diplomado.pucmm.mescyt.java.petcareapp.javaee.filtros	Filtros para el manejo de acceso y autenticación
org.diplomado.pucmm.mescyt.java.petcareapp.javaee.sesiones	Contener las clases para iniciar o concluir una sesión.

Persistencia

Java Persistence API (JPA) 2.1



En la clase anterior trabajamos con **Java Persistence API 2.1 (JSR 338)** y conocemos cómo generar las entidades a partir de una base de datos ya existentes.

- Generar las entidades en el paquete especificado para tales fines.
- Crear servicios para tener acceso a la data de nuestras entidades

Un dato que tenemos que considerar es que nuestra unidad de persistencia **(PersistanceUnit)** tendrá un ligero cambio. El atributo **transaction-type** en lugar de decir **RESOURCE_LOCAL** debe ser cambiado a **JTA.**

También debemos configurar un datasource para la conexión a la base de datos.

En la creación de los servicios introducimos un concepto nuevo llamado **Contextos y la inyección de dependencias (Contexts and Dependency Injection).**

Contexts and Dependency Injection 1.1 (JSR 346)

La inyección de dependencias consiste en cargar las propiedades de los beans mediante su constructor o sus setters en el momento de iniciar la aplicación.
Sin inyección de dependencias, cada clase llama al objeto que necesita en tiempo de ejecución.

Mientras que con inyección de dependencias, cada objeto es cargado en cada clase que lo necesita en tiempo de inicialización.

La forma habitual de implementar este patrón es mediante un "Contenedor DI".

En Java EE 6, era necesario agregar explícitamente un archivo beans.xml con el fin de habilitar escaneo CDI.

La DI (Inyección de dependencia) se considera correctamente una preocupación generalizada, los desarrolladores de Java EE a menudo se encuentran con este requisito confuso.

Uno de los cambios más significativos de CD 1.1/Java EE 7 es que por petición popular, el CDI está activada de forma predeterminada.

Esto significa que no hay necesidad de añadir explícitamente un archivo beans.xml para permitir DI.

Sin embargo, también es muy importante entender que la CDI ahora también proporciona un control más preciso sobre la exploración de componente mediante el atributo **'bean-discovery-mode'**.

Este atributo tiene tres valores posibles:
'annotated' - traducido libremente, significa que sólo se procesan los componentes con una anotación de nivel de clase.
'all' - todos los componentes son procesados, al igual que lo fueron en Java EE 6, con la beans.xml explícito.

'none' - CDI estará desactivada.

De forma predeterminada, si no se especifica nada y no hay **beans.xml**, el modo bean-discovery **'annotated'** es asumido, en lugar de suponer "all".

Ejemplo básico

```
@PersistenceContext
private EntityManager entityManager;
```

Integración de Twitter Boostrap con JSP

Tomamos como base un proyecto **jsp-twitter-bootstrap**, quien ya contiene la integración de la plantilla https://getbootstrap.com/examples/jumbotron/, ademas de esa plantilla queremos agregarle un carousel en la pagina de inicio https://getbootstrap.com/examples/carousel/

Luego de agregar el carousel, personal las columnas con el siguiente contenido.

```
 <!-- Example row of columns -->
       <div class="row">
       <div class="col-md-4">
       <h2>Registrarse Hoy</h2>
       <p>Recibe las mejores atenciones en el cuidado de sus mascotas, personal especializado,
buen servicios y excelentes precios </p>
       <p><a class="btn btn-lg btn-primary"
href="${pageContext.request.contextPath}/signup.jsp" role="button">Registrarse</a></p>
       </div>
       <div class="col-md-4">
       <h2>Sobre Nosotros</h2>
       <p>Conoce más sobre los servicios que ofrecemos y sobre cada uno de nuestro personal.
</p>
       <p><a class="btn btn-default btn-lg active"
href="${pageContext.request.contextPath}/about.jsp" role="button">Leer más</a></p>
       </div>
       <div class="col-md-4">
       <h2>Contáctenos</h2>
       <p>Entre en contacto con nosotros. Escríbenos tus sugerencias o comentarios y en la
mayor brevedad un representante se pondrá en contacto con usted</p>
       <p><a class="btn btn-default btn-lg active"
href="${pageContext.request.contextPath}/contactos.jsp" role="button">Formulario de
Contacto</a></p>
       </div>
       </div>
```

Crear los archivos about y contactos que están siendo usado como enlace en la página principal.

about.jsp

```
<div class="container">
		<h1>PetCare</h1>
		<p class="lead">

		PetCare es una aplicación demo con fines demostrar conceptos básicos en el desarrollo de aplicaciones web en Java.
		</p>

	</div>
<div class="container">
	<div class="row marketing">
		<div class="col-lg-6">
		<h2>JSR 340</h2>
		<p>Java Servlets 3.1.</p>

		<h2>JSR 338</h2>
		<p>Java Persistence API 2.1 </p>


		</div>

		<div class="col-lg-6">

		<h2>Requerimientos</h2>
		<p>Java 1.8 +</p>
		<p>Netbeans 8.0.2+</p>
		<p>Glassfish 4.1.1 +</p>
		<p>Mysql Server 5.0+</p>
		<p>Apache Maven 3.2.1+ </p>

		</div>
	</div>
	<div>
		<h2>Características Adicionales</h2>
		<p>La plantilla base es una combinacion de las plantillas carousel y jumbotron de Twitter Bootstrap (Twitter Bootstrap 3.1.1)</p>
```

```
        </div>
        <div>
        <p>Autor: <a href="https://github.com/ecabrerar">[Eudris Cabrera]</a>
               </p>
        </div>

        <div>
        <h2>Enlaces</h2>
        <ul>
               <li><a href="http://getbootstrap.com/getting-started/">Twitter
Bootstrap</a></li>

        </ul>
        </div>
```

contactos.jsp

```
 <hr class="featurette-divider hidden-lg">
  <div class="col-lg-5 col-md-push-1">
        <address>
        <h3>Ubicación</h3>
        <p class="lead"><a href="https://www.google.com/maps/@19.4398451,-70.6730233,12z"
target="_blank">Santiago de los Caballeros<br>
Av. Hispanoamericana</a><br>
        Telefono: XXX-XXX-XXXX
        </address>
  </div>
```

También queremos incluir un formulario de registro http://bootsnipp.com/snippets/21jx para permitir que los usuarios puedan registrarse.

La página para registrarse debe llamarse **signup.jsp** e implementa la plantilla de acceso público.

Para el área de acceso restringido usaremos la plantilla https://getbootstrap.com/examples/dashboard/. En la integración a nuestro proyecto colocar el nombre **dashboard-layout.tag.**

Leer las especificaciones de la guia Integración de Twitter Boostrap (HTML5 y CSS3) con JSP

Puntos a realizar.
Los archivos jsp para el área de acceso restringido serán colocados con app.
Crear página index.jsp para al área de acceso restringido.

Crear servlets para controlar la navegación de la aplicación.

PetCareWebAppServlet : Controla la navegación de la aplicación.
AdminServlet : Controla la navegación de funcionalidades que están bajo la supervisión de un administrador.
HomeServlet: Inicio de la aplicación.

Realizar uno o varios mantenimientos básicos (CRUD).

Crear Servlet para Login y Logout.
LoginServlet.java

Validar que el usuario se encuentra realizado, si la respuesta es verdadera, hacer la siguiente acción.

```
 //guardar el usuario en session
session.setAttribute("currentSessionUser",opCliente.get());

//setting session to expiry in 30 mins
session.setMaxInactiveInterval(30*60);

Cookie userName = new Cookie("user", usuario);
userName.setMaxAge(30*60);

response.addCookie(userName);
response.sendRedirect("app/index.jsp");
```

En caso que el usuario no se encuentre registrado, hacer lo siguiente.

```
 RequestDispatcher rd = getServletContext().getRequestDispatcher("index.jsp");

              //Enviando mensaje a la pagina de login
              session.setAttribute("loginFailed","Usuario no registrado");
              response.sendRedirect("index.jsp");

              rd.include(request, response);
```

LogoutServlet.java

Terminar la sección activa del usuario registrado.

```
    //invalidate the session if exists
    HttpSession session = request.getSession(false);

  //invalidar session
  session.invalidate();

  //Enviando mensaje a la pagina de login
   response.sendRedirect("index.jsp");
```

Crear un filtro y colocarle el nombre de **AuthenticationFilter.java.**
La función de este filtro es no permitir el acceso al área restringida si no es un usuario logueado.

Validación a realizar en el método **doFilter(ServletRequest request, ServletResponse response, FilterChain chain).**

```
 req.getSession(false);

       if (req.getSession().getAttribute("currentSessionUser") == null
                && !(uri.endsWith("index.jsp") || uri.endsWith("LoginServlet") ||
uri.endsWith("signup.jsp")
                                ||
uri.matches(".*(css|jpg|png|gif|js|eot|svg|ttf|woff|woff2)"))) {

                this.context.log("Unauthorized access request");

               String viewPath = String.join("/", req.getContextPath(),"index.jsp") ;

               res.sendRedirect(viewPath);

       }else{

               // pass the request along the filter chain
               chain.doFilter(request, response);
       }
```

Bean Validation 1.1 (JSR 349)

Bean Validation define un modelo de metadatos y una interfaz para la validación de JavaBeans.

La fuente de los metadatos consiste en anotaciones, con la posibilidad de sobreescribir y extender estos metadatos por medio del uso de descriptores de validación en formato XML.

Java Bean Validation (JSR 303) fue aprobado por el JCP al 16 de noviembre de 2009, siendo aceptado como parte de la especificación Java EE 6.

Bean Validation 1.1 se estuvo enfocado en los siguientes temas:

- Apertura de la especificación y su proceso
- Validación a nivel de método (validación de parámetros o valores de retorno)
- Inyección de dependencias para los componentes Bean Validation
- Integración con el contexto y la inyección de dependencias (CDI)
- Conversión de grupo
- Interpolación de mensaje de error utilizando las expresiones EL

Validación de métodos

```
public String sayHello(@Size(max = 3)String name) {
    return "Hello " + name;
}

@Future
public Date showDate(boolean correct) {
    Calendar cal = Calendar.getInstance();
    cal.add(Calendar.DAY_OF_MONTH, correct ? 5 : -5);
    return cal.getTime();
}

public String showList(@NotNull @Size(min=1, max=3) List<String> list, @NotNull String prefix) {
    StringBuilder builder = new StringBuilder();

    for (String s : list) {
        builder.append(prefix).append(s).append(" ");
    }

    return builder.toString();
}
```

Integración con CDI

```
class ZipCodeValidator implements ConstraintValidator<ZipCode, String> {

    @Inject @France
    private ZipCodeChecker checker;

    public void initialize(ZipCode zipCode) {}

    public boolean isValid(String value, ConstraintValidationContext context) {
        if (value==null) return true;
        return checker.isZipCodeValid(value);
    }
}
```