15ª Emisión

DIPLOMADO

Desarrollo de Sistemas con Tecnología Java

Módulo 5-6

Desarrollo de aplicaciones empresariales con Jakarta EE

Uriel Hernández

DGTIC

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Dirección General de Cómputo y de Tecnologías de información y Comunicación
Dirección de Docencia en TIC

50 AÑOS
Educación Continua
1971 - 2021


JPA

JPA

- Java Persistence API.
- Es una especificación de Java para realizar la persistencia de objetos en un modelo relacional.
- JPA actúa como un puente entre modelos de dominio orientados a objetos (Java) y sistemas de base de datos relacionales (MariaDB, MySQL).
- JPA es sólo una especificación, por lo que requiere una implementación. Algunos frameworks de ORM como Hibernate, TopLink e iBatis implementan la especificación JPA.
- Hibernate es la implementación de la especificación por default.

Requerimientos Clase de Entidad

- La clase debe ser pública.
- Debe tener un constructor sin argumentos.
- Debe ser Serializable.

Anotaciones JPA

- **@Entity**: Declara la clase como una entidad JPA
- **@Table**: En caso de que el nombre de la clase no sea el mismo de la tabla en la base de datos, se puede utilizar para especificar el nombre de la tabla
- **@Id**: Especifica que la propiedad identifica la entidad de forma única en la base de datos (PK)
- **@GeneratedValue(strategy=GenerationType.IDENTITY)**: Especifica que el valor de la propiedad es autogenerado. En nuestro caso estamos utilizando valores autogenerados por la base de datos por lo que se utilizará la estrategia IDENTITY

Anotaciones JPA

- **@ManyToOne**: Especifica una propiedad como una relación uno-a-muchos en la base de datos. En Java hace referencia a una composición de objetos mientras que en la base de datos es un campo que hace referencia a una Foreign Key (FK)
- **@ManyToMany**: Permite definir una relación muchos-a-muchos. En Java hace referencia a una Colección de objetos
- **@JoinColum**: Permite especificar el nombre del campo en la base de datos que hace la relación de la FK. Necesario cuando el campo no cumple con la convención de nombrado para que se identifique de forma automática. La convención de nombrado por default es: entidad_id
- **@IdClass**: Permite definir una PK compuesta (por 2 o más campos). Puede utilizarse para definir una relación @ManyToMany de acuerdo a la estrategia de mapeo

Unidad de Persistencia

- Agrupación lógica de un conjunto de entidades que tienen la misma configuración.
- La unidad de persistencia declara el datasource y el tipo de transacción, junto con el conjunto de entidades que pertenecen a dicha unidad (sólo si el tipo de transacción no es JTA).
- Un datasource es un pool de conexiones a una base de datos que puede ser utilizado por JPA. El datasource es creado previamente en el servidor de aplicaciones donde la aplicación será desplegada.
- Una unidad de persistencia se define por medio del archivo resources/META-INF/persistence.xml
- Los valores que puede tomar la propiedad transaction-type son:
  - RESOURCE_LOCAL
  - JTA

persistence.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence xmlns="https://jakarta.ee/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
             https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd"
             version="3.0">

    <persistence-unit name="pixup" transaction-type="JTA">
        <jta-data-source>jdbc/pixupDS</jta-data-source>
    </persistence-unit>
</persistence>
```

PersistenceContext

- Un contexto persistente se utiliza para generar una instancia de una unidad de persistencia.
- Es un conjunto de entidades administradas pertenecientes a una unidad de persistencia.
- Se utiliza la anotación **@PersistenceContext** para generar una instancia de la unidad de persistencia especificada.

```
@PersistenceContext(unitName="pixup")
private EntityManager entityManager;
```

EntityManager

- Es una **interface** que tiene métodos para realizar las operaciones CRUD sobre entidades JPA (Create, Read, Update, Delete).
- Es un objeto administrado por el servidor de aplicaciones por lo que es responsabilidad del servidor crearlo. Para hacerlo se utiliza el mecanismo de inyección de dependencias para solicitar al JPA runtime la inyección de una instancia.
- Una vez que se tiene una instancia de EntityManager se pueden realizar las operaciones CRUD sobre las entidades.

Inversión de Control (IoC)

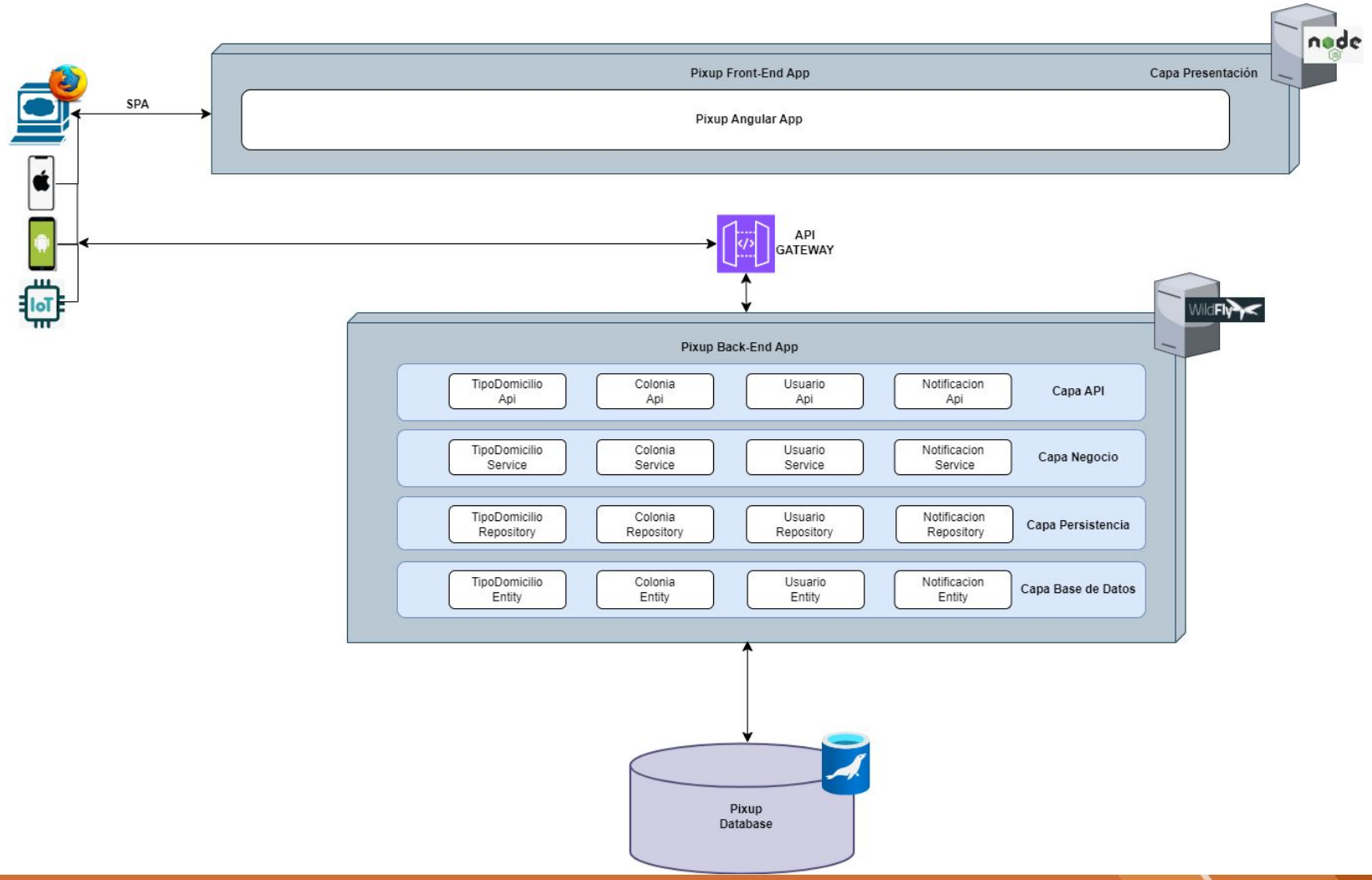
- Este concepto originalmente hacía referencia a un patrón de diseño el cual establecía que en lugar de que los componentes creen y mantengan el ciclo de vida de otros componentes de los que tienen dependencia, la aplicación delegue la creación de estos componentes a un contenedor, en nuestro caso el servidor de aplicaciones.
- Martin Fowler sugirió renombrar este patrón a un término más auto explicable y surgió el nombre de Inyección de Dependencias.

  https://martinfowler.com/articles/injection.html
- En la actualidad ambos términos Inversión de Control e Inyección de Dependencias hacen alusión al mismo concepto.

Layers Pattern (Capas)

- Es uno de los patrones de arquitectura más conocidos e implementados.
- Es el patrón de facto para la mayoría de aplicaciones Java EE.
- Consiste en dividir los componentes de una aplicación en capas horizontales, cada una desarrollando un rol específico en la aplicación.
- La mayoría de las aplicaciones que implementan el patrón de capas constan de 4 capas:
  - Capa de presentación
  - Capa de negocio
  - Capa de persistencia
  - Capa de base de datos

Pixup Capas - Angular



Pixup: Operaciones CRUD Entidad Colonia y TipoDomicilio

- Se implementará la funcionalidad de altas/bajas/cambios de la entidad colonia como parte del soporte del sistema Pixup (Mantenimiento Catálogo Colonia) y la consulta para obtener los TiposDomicilio.
- De manera general cuando se cree un nuevo componente el procedimiento será el siguiente:

1. Agregar/Actualizar las dependencias en Maven
2. Agregar/Actualizar las entidades del modelo de dominio (capa de base de datos)
3. Agregar/Actualizar los repositorios (capa de persistencia)
4. Agregar/Actualizar los servicios (capa de negocio)
5. Agregar/Actualizar los controllers (capa presentación - MVC tradicional)
6. Agregar/Actualizar los resources en caso de que el servicio se desee exponer como servicio REST (api) para ser consumido fuera de la aplicación, ej. angular, app android/iOS (capa API)
7. Realizar la configuración de los componentes CDI

Dependencias Maven

- Es buena práctica consolidar la información de las versiones del proyecto

```
<properties>
    <maven.compiler.release>17</maven.compiler.release>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <jakartaee.version>10.0.0</jakartaee.version>
    <ejb.version>4.0.1</ejb.version>
    <lombok.version>1.18.22</lombok.version>
</properties>
```

Dependencias Maven

```xml
<!-- JEE -->
<dependency>
    <groupId>jakarta.platform</groupId>
    <artifactId>jakarta.jakartaee-api</artifactId>
    <version>${jakartaee.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>jakarta.ejb</groupId>
    <artifactId>jakarta.ejb-api</artifactId>
    <version>${ejb.version}</version>
    <scope>provided</scope>
</dependency>

<!-- Domain Model -->
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>${lombok.version}</version>
    <optional>true</optional>
</dependency>
```

Modelo de Dominio: TipoDomicilio

```java
@Data
@NoArgsConstructor
@Entity
@Table(name="tipo_domicilio")
public class TipoDomicilio {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;
    private String descripcion;

}
```

Modelo de Dominio: Colonia

```java
@Data
@NoArgsConstructor
@Entity
public class Colonia {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;
    private String nombre;
    private String cp;
    @ManyToOne(targetEntity=Municipio.class)
    @JoinColumn(name="id_municipio", nullable=false)
    private Municipio municipio;

}
```

Jakarta Enterprise Beans (JEB)

- Jakarta Enterprise Beans (anteriormente Enterprise JavaBeans - EJB) fueron creados para separar la capa de vista de las capas de negocio y de persistencia.
- Pueden ser utilizados de forma local o remota.
- La idea de ejecutar servicios de negocio de manera remota se encuentra alineada a principios de arquitecturas distribuidas, las cuales ofrecen una serie de beneficios sobre los atributos de escalabilidad y disponibilidad.

Jakarta Enterprise Beans (JEB)

- El contenedor puede proporcionar servicios de gestión de transacciones y seguridad.
- Los JEB pueden ser invocados por otras aplicaciones Java.
- Existen 4 tipos de JEB:
  - stateless
  - stateful
  - singleton
  - message driven

Session Bean

- Son componentes que tienen alguna de las siguientes anotaciones:
  - @Stateless
  - @Stateful
  - @Singleton
- Cualquier bean que contenga algunas de estas anotaciones es un CDI bean (bean gestionado por el contenedor CDI) además de ser un EJB (Enterprise JavaBeans) y tener las características y servicios provistos por esta especificación.

Session Bean

- **@Stateless**: Son los más utilizados ya que no mantienen un estado conversacional, lo cual permite una alta escalabilidad.
- **@Stateful**: Sólo son utilizados en situaciones donde un estado conversacional es requerido a lo largo de múltiples peticiones de un mismo cliente. En la práctica casi nunca son utilizados ya que limita la escalabilidad de los sistemas y existen mecanismos distribuidos más eficientes para mantener el estado de un cliente (caché distribuidos, bases de datos distribuidas).
- **@Singleton**: Crea una sola instancia del bean.

Capa de Persistencia: TipoDomicilioRepository

```java
public interface TipoDomicilioRepository {

    Collection<TipoDomicilio> findAll();
    Optional<TipoDomicilio> findById(Integer id);

}
```

Capa de Persistencia: JpaTipoDomicilioRepository

```java
@Singleton
public class JpaTipoDomicilioRepository implements TipoDomicilioRepository {

    @PersistenceContext(unitName="pixup")
    private EntityManager entityManager;

    @Override
    public Collection<TipoDomicilio> findAll() {
        TypedQuery<TipoDomicilio> query = entityManager.createQuery(
                "SELECT td FROM TipoDomicilio td", TipoDomicilio.class);
        return query.getResultList();
    }

    @Override
    public Optional<TipoDomicilio> findById(Integer id) {
        TipoDomicilio tipoDomicilio = entityManager.find(TipoDomicilio.class, id);
        return tipoDomicilio != null ? Optional.of(tipoDomicilio) : Optional.empty();
    }
```

Capa de Persistencia: ColoniaRepository

```
public interface ColoniaRepository {

    Collection<Colonia> findByCp(String cp);
    Optional<Colonia> findById(Integer id);

}
```

Capa de Persistencia: JpaColoniaRepository

```java
@Singleton
public class JpaColoniaRepository implements ColoniaRepository {

    @PersistenceContext(unitName="pixup")
    private EntityManager entityManager;

    @Override
    public Collection<Colonia> findByCp(String cp) {
        TypedQuery<Colonia> query = entityManager.createQuery(
                "SELECT c FROM Colonia c WHERE c.cp = ?1", Colonia.class);
        query.setParameter(1, cp);
        return query.getResultList();
    }

    @Override
    public Optional<Colonia> findById(Integer id) {
        Colonia colonia = entityManager.find(Colonia.class, id);
        return colonia != null ? Optional.of(colonia) : Optional.empty();
    }
```

persistence.xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence xmlns="https://jakarta.ee/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
             https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd"
             version="3.0">

    <persistence-unit name="pixup" transaction-type="JTA">
        <jta-data-source>jdbc/pixupDS</jta-data-source>
        <properties/>
    </persistence-unit>
</persistence>
```

PRÁCTICA NO. 1

Contacto

Uriel Hernández

Solution Architect

urielhdezorozco@yahoo.com.mx

Redes sociales:

https://www.linkedin.com/in/juho-mex