



# JavaEE Application

Day 3: JPA (Java Persistence API) & Database Integration

**Alan Biju**  
@itsmeambro



# 1. JPA with Hibernate – Core Concepts

JPA (Java Persistence API) is the standard API for ORM in Java EE. Hibernate is the most popular JPA implementation used for persistence.

## ◆ Understanding JPA Components

1. EntityManager – Handles CRUD operations on entities.
2. Persistence Context – Maintains the state of entities.
3. Transactions – Ensures data consistency using ACID properties.
4. JPQL (Java Persistence Query Language) – A query language similar to SQL but operates on entities instead of tables.
5. Criteria API – A type-safe and programmatic way to build queries.
6. Relationships in ORM
  - One-to-One – e.g., User ↔ Profile
  - One-to-Many – e.g., Customer ↔ Orders
  - Many-to-Many – e.g., Students ↔ Courses

## ◆ Defining an Entity with Relationships

@Entity

```
public class Customer {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column(nullable = false)  
    private String name;  
  
    @Column(unique = true, nullable = false)  
    private String email;  
  
    @OneToMany(mappedBy = "customer", cascade = CascadeType.ALL)  
    private List<Order> orders;  
}
```

## 2. Database Connection & Configuration

To integrate JPA with MySQL/PostgreSQL in a Jakarta EE application, configure persistence.xml.

- ◆ Setting Up MySQL/PostgreSQL with JPA
  - Use MySQL Connector/J or PostgreSQL JDBC Driver.
  - Configure connection pooling for better performance.

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence" version="2.1">
  <persistence-unit name="myPU">
    <properties>
      <property name="jakarta.persistence.jdbc.driver"
        value="com.mysql.cj.jdbc.Driver"/>
      <property name="jakarta.persistence.jdbc.url"
        value="jdbc:mysql://localhost:3306/mydb"/>
      <property name="jakarta.persistence.jdbc.user" value="root"/>
      <property name="jakarta.persistence.jdbc.password"
        value="password"/>

      <!-- Hibernate properties -->
      <property name="hibernate.hbm2ddl.auto" value="update"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>

      <!-- Connection Pooling with HikariCP -->
      <property name="hibernate.hikari.maximumPoolSize" value="10"/>
      <property name="hibernate.hikari.minimumIdle" value="5"/>
    </properties>
  </persistence-unit>
</persistence>
```

### 3. Mini Project: RESTful CRUD App using JPA + Servlets

#### Project Overview

We will build a RESTful CRUD application where users can be created, retrieved, updated, and deleted using JPA, Servlets, and JSON responses.

#### ◆ Features:

- ✓ Create and retrieve database records using JPA.
- ✓ Implement EntityManager to interact with the database.
- ✓ Develop RESTful endpoints using HttpServlet.
- ✓ Use JPA Transactions to ensure data integrity.

#### ◆ Project Structure

JavaEE\_CRUD

```
| — src/main/java
|   | — com.example.config (Configuration classes)
|   | — com.example.model (Entity classes)
|   | — com.example.dao (Data Access Layer)
|   | — com.example.service (Business Logic Layer)
|   | — com.example.servlet (Servlets - Controller Layer)
| — src/main/webapp
|   | — WEB-INF
|   |   | — web.xml (Deployment Descriptor)
|   |   | — index.jsp (Frontend - optional)
| — pom.xml (Maven dependencies)
| — persistence.xml (JPA Configuration)
```

## Step 1: Create the Entity Class (User.java)

📌 Path: src/main/java/com/example/model/User.java

```
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String email;

    // Constructors
    public User() {}
    public User(String name, String email) {
        this.name = name;
        this.email = email;
    }

    // Getters & Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
}
```

## Step 2: Create a DAO (Data Access Object)

📌 Path: src/main/java/com/example/dao/UserDAO.java

```
public class UserDAO {
    private EntityManagerFactory emf =
Persistence.createEntityManagerFactory("UserPU");

    public void createUser(User user) {
        EntityManager em = emf.createEntityManager();
        EntityTransaction tx = em.getTransaction();
        tx.begin();
        em.persist(user);
        tx.commit();
        em.close();
    }

    public User getUser(Long id) {
        EntityManager em = emf.createEntityManager();
        User user = em.find(User.class, id);
        em.close();
        return user;
    }

    public List<User> getAllUsers() {
        EntityManager em = emf.createEntityManager();
        List<User> users = em.createQuery("SELECT u FROM User u",
User.class).getResultList();
        em.close();
        return users;
    }

    public void updateUser(User user) {
        EntityManager em = emf.createEntityManager();
        EntityTransaction tx = em.getTransaction();
        tx.begin();
        em.merge(user);
        tx.commit();
        em.close();
    }
}
```

```
public void deleteUser(Long id) {  
    EntityManager em = emf.createEntityManager();  
    EntityTransaction tx = em.getTransaction();  
    tx.begin();  
    User user = em.find(User.class, id);  
    if (user != null) {  
        em.remove(user);  
    }  
    tx.commit();  
    em.close();  
}
```



### Step 3: Create the Servlet (UserServlet.java)

✚ Path: src/main/java/com/example/dao/UserDAO.java

```
@WebServlet("/users")
public class UserServlet extends HttpServlet {

    private UserDAO userDAO = new UserDAO();

    protected void doPost(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
        String name = request.getParameter("name");
        String email = request.getParameter("email");

        User user = new User(name, email);
        userDAO.addUser(user);

        response.getWriter().write("User added successfully!");
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        response.setContentType("application/json");

        String userIdParam = request.getParameter("id");
        if (userIdParam != null) {
            int id = Integer.parseInt(userIdParam);
            User user = userDAO.getUserById(id);
            if (user != null) {
                out.print("{\"id\": " + user.getId()
                    + ", \"name\": \"" + user.getName()
                    + "\", \"email\": \"" + user.getEmail() + "\"}");
            } else {
                response.setStatus(HttpServletResponse.SC_NOT_FOUND);
                out.print("{\"error\": \"User not found\"}");
            }
        } else {
            List<User> users = userDAO.getAllUsers();
```



```

        out.print("[");
        for (int i = 0; i < users.size(); i++) {
            User u = users.get(i);
            out.print("{ \"id\": " + u.getId()
                + ", \"name\": \" " + u.getName()
                + "\", \"email\": \" " + u.getEmail() + "\" }");
            if (i < users.size() - 1) out.print(",");
        }
        out.print("]");
    }
}

protected void doPut(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
    int id = Integer.parseInt(request.getParameter("id"));
    String name = request.getParameter("name");
    String email = request.getParameter("email");

    userDAO.updateUser(id, name, email);
    response.getWriter().write("User updated successfully!");
}

protected void doDelete(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
    int id = Integer.parseInt(request.getParameter("id"));
    userDAO.deleteUser(id);
    response.getWriter().write("User deleted successfully!");
}
}

```

**If you find this  
helpful, like and  
share it with your  
friends**



**Alan Biju**  
@itsmeambro