

# PoC (Prueba de Concepto)

## Histórico de versiones

*Esta sección contiene la información de los cambios que se realizaron en este documento.*

<b>Versión</b>	<b>Fecha</b>	<b>Resumen de los cambios producidos</b>
1.0	30/06/2012	Versión Inicial del documento

## Control de la Preparación, Revisión y Aprobación

*Esta sección contiene la información de los responsables de los cambios que se realizaron en este documento.*

<b>Responsabilidad</b>	<b>Nombre Responsable</b>	<b>del</b>	<b>Cargo Rol</b>	<b>/</b>	<b>Fecha</b>
Revisión	Esteban Arlando		Analista		30/06/2012
Preparación	Damián Lezcano		Analista		03/07/2012
Supervisión	Ezequiel Fernández		Analista		30/06/2012
Aprobación	Arratta Sebastián		Analista		30/06/2012

## Tabla de Contenidos

Introducción.....	4
Descripción del Ejemplo.....	4
Javaee-oracle-poc-web.....	5
Estructura de directorios.....	5
Controlador de Pagina.....	6
Pagina web.....	9
Relacion Controlador / Pagina.....	11
Javaee-oracle-poc-service.....	12
Estructura de directorios.....	12
Servicios EJB.....	13
EmployeeService.....	13
EmailService.....	15
javaee-oracle-poc-model.....	17
Estructura de directorios.....	17
Employee.....	18
Payslip.....	20
Persistence.xml.....	21

## Introducción

Poc (Prueba de Concepto), la idea de este documento es brindar una guía conceptual basada en los lineamiento de desarrollo propuesto para la aplicación. Para ello vamos a dividirlo en las tres capas principales que conforman la arquitectura propuesta, estas son **web, servicio y persistencia**. Como se vera mas adelante, solamente se abordaran elementos (archivos, mapeos) necesarios para terminar de comprender la solución, esta prueba de concepto no sirve para ver en detalle como desarrollar la aplicación, sino mas bien una guía conceptual.

Para mas detalle ver el proyecto **java-oracle-poc.zip** el cual contiene todo el source necesario y probado.

## Descripción del Ejemplo

La aplicación es muy simple pero trata de englobar lo necesario para dejar en claro la arquitectura elegida y los puntos que quedan inconsistentes para terminar de cerrar los conceptos. Esta aplicación, es un ABM de empleados, donde todo se hace en la misma pagina, la misma consume los servicios ejb expuestos y los mismo interactuan con la capa de persistencia para cerrar el circuito. Se ve un ejemplo simple de como graficar utilizando los componentes que vienen incluidos en el framework de ADF Oracle, por otra parte tambien se explica en detalle con se envian mail a traves de la API JavaMail.

## Javaee-oracle-poc-web

### Estructura de directorios



## Controlador de Pagina

```
package com.solution.oracle.poc.web.controller;

import com.solution.oracle.poc.model.Employee;
import com.solution.oracle.poc.model.IdentityDocument;
import com.solution.oracle.poc.service.EmployeeService;
import com.solution.oracle.poc.web.util.ServiceLocator;
import oracle.adf.view.faces.bi.component.graph.UIGraph;

import java.util.List;

import org.apache.log4j.Logger;

public class EmployeeController {

    private List<Employee> employees;
    private Employee employee;
    private EmployeeService employeeService;
    private UIGraph barGraph;

    private static final Logger logger = Logger.getLogger(EmployeeController.class);

    public EmployeeController() {
        employeeService = (EmployeeService) ServiceLocator.getService("employeeService");
        // Acá se configura el origen de datos, para esto hay que implementar una interfaz
        // el cual nos obliga a implementar tres metodos, getValueX() y getValueY(),
        // ambos del tipo de dato double y getName()
        barGraph.setValues(null); //espera una lista de UIGraph
    }

    public void newEmployee() {
        logger.info("--- Creating new employee...");
        employee = new Employee();
        employee.setIdentityDocument(new IdentityDocument());
    }

    public void save() {
        try{
            if (employee.getId() == null) {
                employees.add(employee);
                logger.info("--- Saving new employee: " + employee.getName());
            } else {
                logger.info("--- Updating employee: " + employee.getName());
            }
            employeeService.save(employee);
            employee = null;
        }catch (Exception e){
```

```
        FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage(FacesMessage.SEVERITY_ERROR,
                "error message from the controller to try to save...", null));
    }
}

public void cancel() {
    employee = null;
}

public void remove() {
    logger.info("--- Removing employee: " + employee.getName());
    employees.remove(employee);
    employeeService.remove(employee);
    employee = null;
}

public void update() {
    logger.info("--- Updating employee: " + employee.getName());
}

public void generateImage(){
    //Recuperamos la imagen, no hacemos nada, pero se puede mandar por mail
    //o mostrarlo por pantalla
    ImageView imgView = barGraph.getImageView();
}

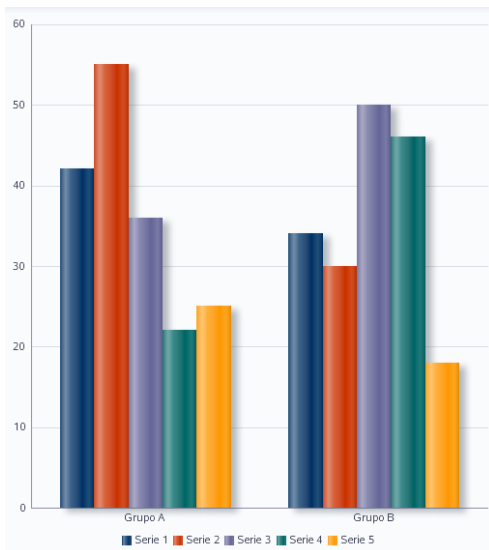
public void setEmployees(List<Employee> employees) {
    this.employees = employees;
}

public List<Employee> getEmployees() {
    if (employees == null) {
        logger.info("--- Obteniendo empleados ---");
        employees = employeeService.findAll(0, 15);
    }
    return employees;
}

... getters y setters
}
```

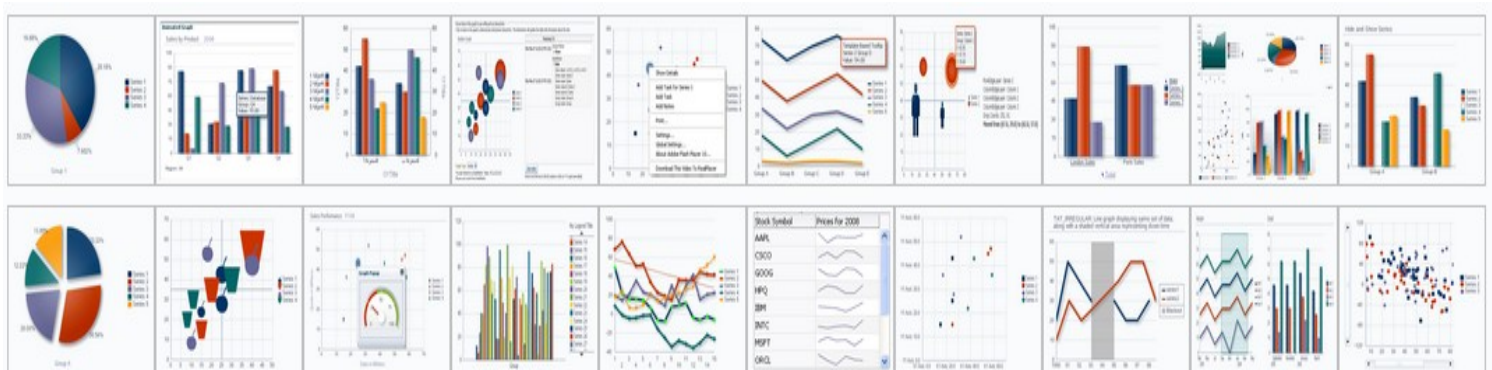
Este controlador, tiene todos los métodos necesarios para que desde la pagina (la interfaz gráfica que el usuario ve realmente) se puedan ejecutar las acciones necesarias: **Alta, Baja y Modificación**. Observar que en el método **save**, en el caso de ocurrir un error al intentar guardar, se lanzaría una excepción, el cual se mostraría en la pagina ("*error message from the controller to try to save*"), a excepcion de un mensaje en la pagina que es lanzado desde el cliente, este se lanza desde el servidor.

Por otra parte el método **generateImage** generará una imagen con lo datos provistos en forma previa (como se explica en el constructor) y a partir de lo visualizado por el componente **BarGraph** de ADF como se muestra a continuación, para ello se debe implementar la interfaz provista por ADF UIGraph el cual obliga a implementar tres metodos **getValueX()**, **getValueY()** ambos del tipo de dato Double, y **getName()** para las etiquetas, entonces por cada empleado creado se debe generar un objeto que implemente UIGraph y retornar en cada caso el valor que corresponde para que el mismo se pueda dibujar correctamente.



*Este grafico es a modo ilustrativo.*

### Otros tipos de graficos disponibles:



Para mas informacion: <http://jdevadf.oracle.com/adf-richclient-demo/faces/feature/graph/graphGallery.jspx>



## Pagina web

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<f:view xmlns:f="http://java.sun.com/jsf/core" xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
xmlns:h="http://java.sun.com/jsf/html">
  <af:document title="employees.jsf" id="d1">
    <af:form id="f1">
      <af:pageTemplate viewId="/oracle/templates/threeColumnTemplate.jspx" id="pt1">
        <f:facet name="center">
          <af:group id="g2">
            <af:panelCollection id="pc1" inlineStyle="width:inherit;" partialTriggers="t1:deleteLink ::cb2">
              <f:facet name="menus"/>
              <f:facet name="toolbar"/>
              <f:facet name="statusbar"/>
              <af:group rendered="#{viewScope.employeeController.employees.isEmpty}" >
                <af:outputText value="no results to show"/>
              </af:group>
              <af:table value="#{viewScope.employeeController.employees}"
                rendered="#{!viewScope.employeeController.employees.isEmpty}"
                var="_employee" rowBandingInterval="0" id="t1">
                <af:column sortable="false" headerText="Name" id="c1">
                  <af:outputText value="#{_employee.name}" id="ot1"/>
                </af:column>
                <af:column sortable="false" headerText="Second Name" id="c2">
                  <af:outputText value="#{_employee.secondName}" id="ot2"/>
                </af:column>
                <af:column sortable="false" headerText="Last Name" id="c3">
                  <af:outputText value="#{_employee.lastName}" id="ot3"/>
                </af:column>
                <af:column sortable="false" headerText="File Number" id="c4">
                  <af:outputText value="#{_employee.fileNumber}" id="ot4"/>
                </af:column>
                <af:column sortable="false" headerText="" id="c5">
                  <af:commandImageLink icon="/img/update_ena.png" id="updateLink" partialSubmit="true"
                    action="#{viewScope.employeeController.update}">
                    <f:setPropertyActionListener target="#{viewScope.employeeController.employee}"
                      value="#{_employee}"/>
                  </af:commandImageLink>
                </af:column>
                <af:column sortable="false" headerText="" id="c6">
                  <af:commandImageLink icon="/img/delete_ena.png" id="deleteLink" partialSubmit="true"
                    action="#{viewScope.employeeController.remove}">
                    <f:setPropertyActionListener target="#{viewScope.employeeController.employee}"
                      value="#{_employee}"/>
                  </af:commandImageLink>
                </af:column>
              </af:panelCollection>
            </af:group>
          </f:facet>
        </af:pageTemplate>
      </af:form>
    </af:document>
  </f:view>

```

```

        </af:column>
    </af:table>
</af:panelCollection>
<af:separator id="s1"/>
<af:commandImageLink icon="/img/new_ena.png" id="addLink"
    action="#{viewScope.employeeController.newEmployee}" partialSubmit="true"/>
<af:panelGroupLayout id="pg1"
    partialTriggers="cb1 cb2 addLink pc1:t1:updateLink pc1:t1:deleteLink">
    <h:panelGrid columns="8" id="pg1"
        rendered="#{viewScope.employeeController.employee != null}">
        <af:outputLabel id="label1" value="Name:"/>
        <af:inputText id="it1" value="#{viewScope.employeeController.employee.name}"
            required="true"/>
        <af:outputLabel id="label2" value="Second Name:"/>
        <af:inputText id="it2" value="#{viewScope.employeeController.employee.secondName}" />
        <af:outputLabel id="label3" value="Last Name:"/>
        <af:inputText id="it3"
            value="#{viewScope.employeeController.employee.lastName}" required="true"/>
        <af:outputLabel id="label4" value="File Number:"/>
        <af:inputText id="it4" value="#{viewScope.employeeController.employee.fileNumber}"
            required="true"/>
        <af:outputLabel id="label5" value="Document Type:"/>
        <af:inputText id="it5"
            value="#{viewScope.employeeController.employee.identityDocument.type}"/>
        <af:outputLabel id="label6" value="Document Number:"/>
        <af:inputText id="it6"
            value="#{viewScope.employeeController.employee.identityDocument.number}"/>
        <af:commandButton text="Cancel" id="cb1" immediate="true" partialSubmit="true"
            action="#{viewScope.employeeController.cancel}"/>
        <af:commandButton text="Save" id="cb2" partialSubmit="true"
            action="#{viewScope.employeeController.save}"/>
    </h:panelGrid>
    </af:panelGroupLayout>
</af:group>
</f:facet>
<f:facet name="header"/>
<f:facet name="end">
    <af:group id="g3">
        <!-- START TAG DEMO -->
        <dvt:barGraph shortDesc="Bar Graph" id="barGraph"
            binding="#{viewScope.employeeController.barGraph}" dynamicResize="DYNAMIC_SIZE">
        </dvt:barGraph>
        <!-- END TAG DEMO -->
    </af:group>
</f:facet>
<f:facet name="start"/>
<f:facet name="branding"/>
<f:facet name="copyright"/>
<f:facet name="status"/>

```

```

    </af:pageTemplate>
  </af:form>
</af:document>
<!--oracle-jdev-comment:auto-binding-backing-bean-name:employeeController-->
</f:view>

```

Esta pagina responde a las funcionalidades expuestas en el controlador, cada método expuesto en el mismo debe estar (si fuese necesario) registrado en esta pagina. Por ejemplo si no existen empleados cargados, se mostrar el siguiente mensaje *“no results to show”*, como puede ver este mensaje es propio de la pagina y no fue lanzado desde el controlador. Otro ejemplo es la validaciones, como puede observar algunos campos de entrada (`inputText`) se les ha configurado con la siguiente propiedad: `required="true"`, esto nos permite hacer una validación desde el cliente sin tener que pasar por el servidor, en esto caso lo que se valida es que se halla ingresado los datos, existe otros tags propios del framework que nos permiten hacer un uso mas especifico de las validaciones, como por ejemplo el `af:validateDateTimeRange`: para validar que la fecha no sea mayor ni menor a alguna configurada, `af:validateLongRange`: para validar que un numero no exceda el esperado, etc.

Por otra parte el componente `dvt:barGraph` nos permite generar un gráfico de barra totalmente dinámico, el mismo esta asociado a una variable del controller para configurar el origen de datos. Tambien existen otros tipos de graficos que de forma totalmente transparente se pueden intercambiar (no todos) sin tener que tocar el controlador (o el dataProvider), como por ejemplo: `dvt:areaGraph`, `dvt:pieGraph`, y otros vistos anteriormente.

## Relacion Controlador / Pagina

```

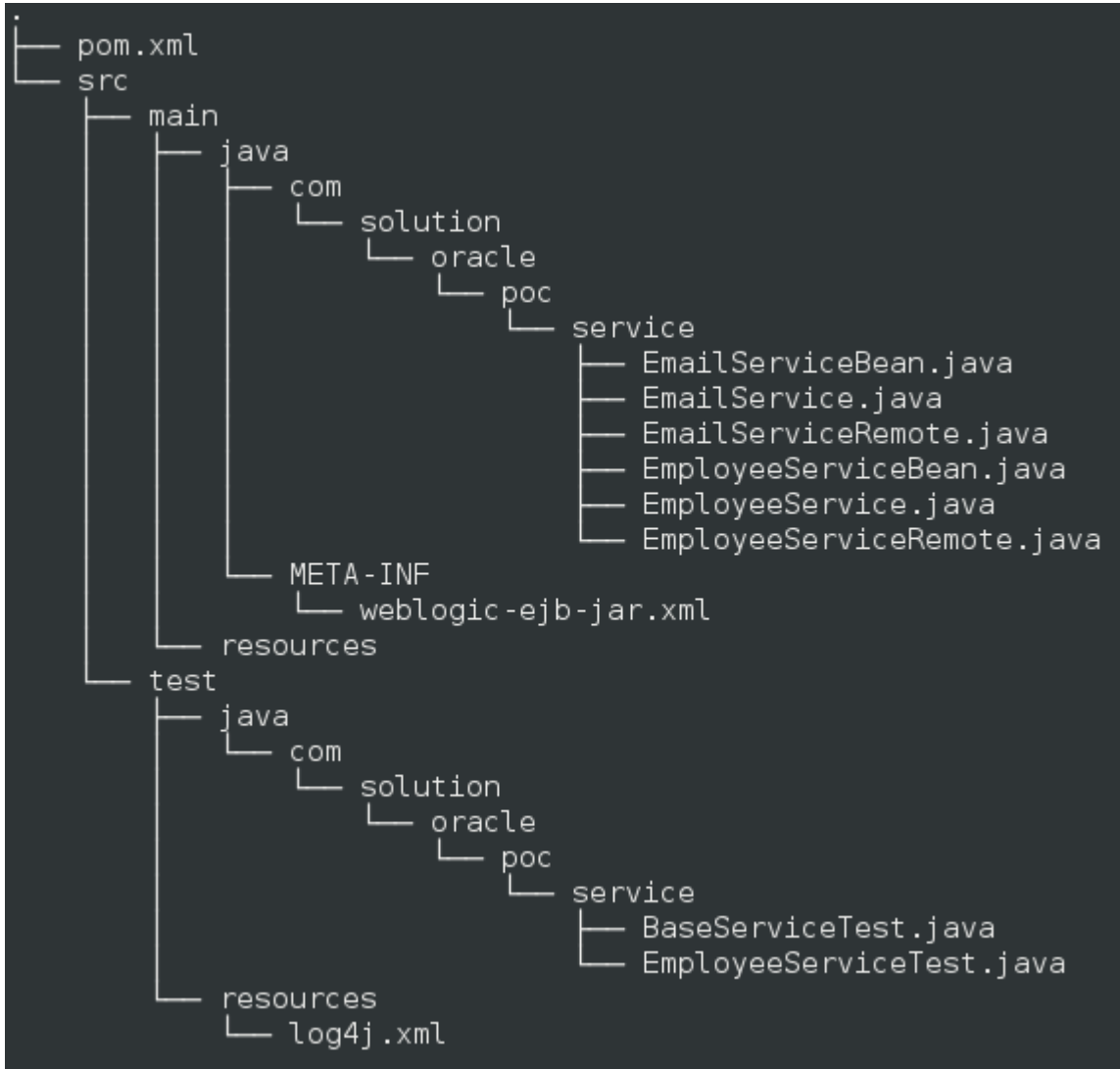
<?xml version="1.0" encoding="UTF-8" ?>
<adfc-config xmlns="http://xmlns.oracle.com/adf/controller" version="1.2">
  <managed-bean id="__1">
    <managed-bean-name>employeeController</managed-bean-name>
    <managed-bean-class>com.epidataconsulting.oracle.poc.web.controller.EmployeeController</managed-bean-
class>
    <managed-bean-scope>view</managed-bean-scope>
    <!--oracle-jdev-comment:managed-bean-jsp-link:1employees.jsf-->
  </managed-bean>
</adfc-config>

```

Este archivo nos permite relacionar un controlador con su pagina, agregando ademas el scope (contexto) en el que va a funcionar: [View](#), [Application](#), [Session](#), etc.

## Javaee-oracle-poc-service

### *Estructura de directorios*



## Servicios EJB

### EmployeeService

```

package com.solution.oracle.poc.service;

import com.solution.oracle.poc.model.Employee;

import java.util.List;

import javax.ejb.Local;
import javax.ejb.Remote;
import javax.ejb.Stateless;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;

@Stateless(name = "EmployeeService", mappedName = "javaee-oracle-poc/EmployeeService")
@Local(value = EmployeeService.class)
@Remote(value = EmployeeServiceRemote.class)
public class EmployeeServiceBean implements EmployeeService {

    @PersistenceContext
    private EntityManager entityManager;

    @EJB
    private EmailService emailService;

    @Override
    public void save(Employee employee) {
        entityManager.merge(employee);
    }

    @Override
    public List<Employee> findByLastName(String lastName) {
        Query query = entityManager.createNamedQuery(Employee.EMPLOYEE_FIND_BY_LAST_NAME);
        // Query query = entityManager.createQuery("select e FROM Employee e WHERE e.lastName = :lastName"); esto es lo mismo que arriba
        query.setParameter("lastName", lastName);
        List<Employee> employees = query.getResultList();
        return employees;
    }

    @Override
    public List<Employee> find(Employee employee) {

```

```
String jpql = "select e FROM Employee e WHERE e.name =:name and e.lastName = :lastName";

if(employee.getName() == null){
    jpql = jpql.replace("e.name=:name and", "");
}

if(employee.getLastName() == null){
    if(employee.getName() == null){
        jpql = "select e FROM Employee e";
    }else{
        jpql = jpql.replace("and e.lastName=:lastName", "");
    }
}

Query query = entityManager.createQuery(jpql);
query.setParameter("name", employee.getName());
query.setParameter("lastName", employee.getLastName());

List<Employee> employees = query.getResultList();
return employees;
}

@Override
public void remove(Employee employee) {
    entityManager.remove(employee);
    emailServiceBean.send(employee, "Fue eliminado");
}

@Override
public List<Employee> findAll(Integer pageNumber, Integer pageSize) {
    Query query = entityManager.createNamedQuery(Employee.EMPLOYEE_FIND_ALL);
    query.setFirstResult(pageNumber);
    query.setMaxResults(pageSize);
    return query.getResultList();
}

@Override
public List<Employee> findAll() {
    Query query = entityManager.createNamedQuery(Employee.EMPLOYEE_FIND_ALL);
    return query.getResultList();
}
}
```

Este EJB nos provee lo necesario para que nuestra aplicación interactue de forma correcta con los las capas adyacentes, entre ellos se puede guardar un empleado, buscar por nombre, eliminar y actualizar (esto se hace con el mismo save). Como se puede apreciar, jpa no es mágico, necesita por ejemplo para las consultas, la query escrita en JPQL propio de JPA que especifica (como en sql) como traer los datos. Por otra parte este ejb nos provee de forma transparente el manejo de las transacciones, de abrir

las conexiones y cerrarlas cuando sea necesario. Se puede ver además como se arma de forma dinámica una query, esto es necesario en las búsquedas donde desde las capas superiores no se envían un campo solamente con puede ser el nombre, sino una entidad, y en base a el contenido de esa entidad se traerán los datos existentes. Esto normalmente se hace que Criteria, pero para el caso práctico de este ejemplo se realizó de la manera descrita en el método ***find(Employee employee)***. Por último se observa que en el método ***remove*** una vez realizada la operación satisfactoriamente, invoca el servicio ejb para el envío de mail con el propósito de informar al destinatario de la operación realizada.

## EmailService

```
package com.solution.oracle.poc.service;

import [...]

@Stateless(name = "EmailService", mappedName = "javaee-oracle-poc/EmailService")
@Local(value = EmailService.class)
@Remote(value = EmailServiceRemote.class)
public class EmailServiceBean implements EmailService {

    void send(Employee employee, String message){

        try
        {
            // Propiedades de la conexión
            Properties props = new Properties();
            props.setProperty("mail.smtp.host", "smtp.gmail.com");
            props.setProperty("mail.smtp.starttls.enable", "true");
            props.setProperty("mail.smtp.port", "587");
            props.setProperty("mail.smtp.user", "info@gmail.com");
            props.setProperty("mail.smtp.auth", "true");

            // Preparamos la sesion
            Session session = Session.getDefaultInstance(props);

            // Construimos el mensaje
            MimeMessage message = new MimeMessage(session);
            message.setFrom(new InternetAddress("info@gmail.com"));
            message.addRecipient(
                Message.RecipientType.TO,
                new InternetAddress(employee.getMail()));
            message.setSubject("Enviado informacion");
            message.setText(message);

            // Lo enviamos.
```

```
Transport t = session.getTransport("smtp");
t.connect("info@gmail.com", "la clave");
t.sendMessage(message, message.getAllRecipients());

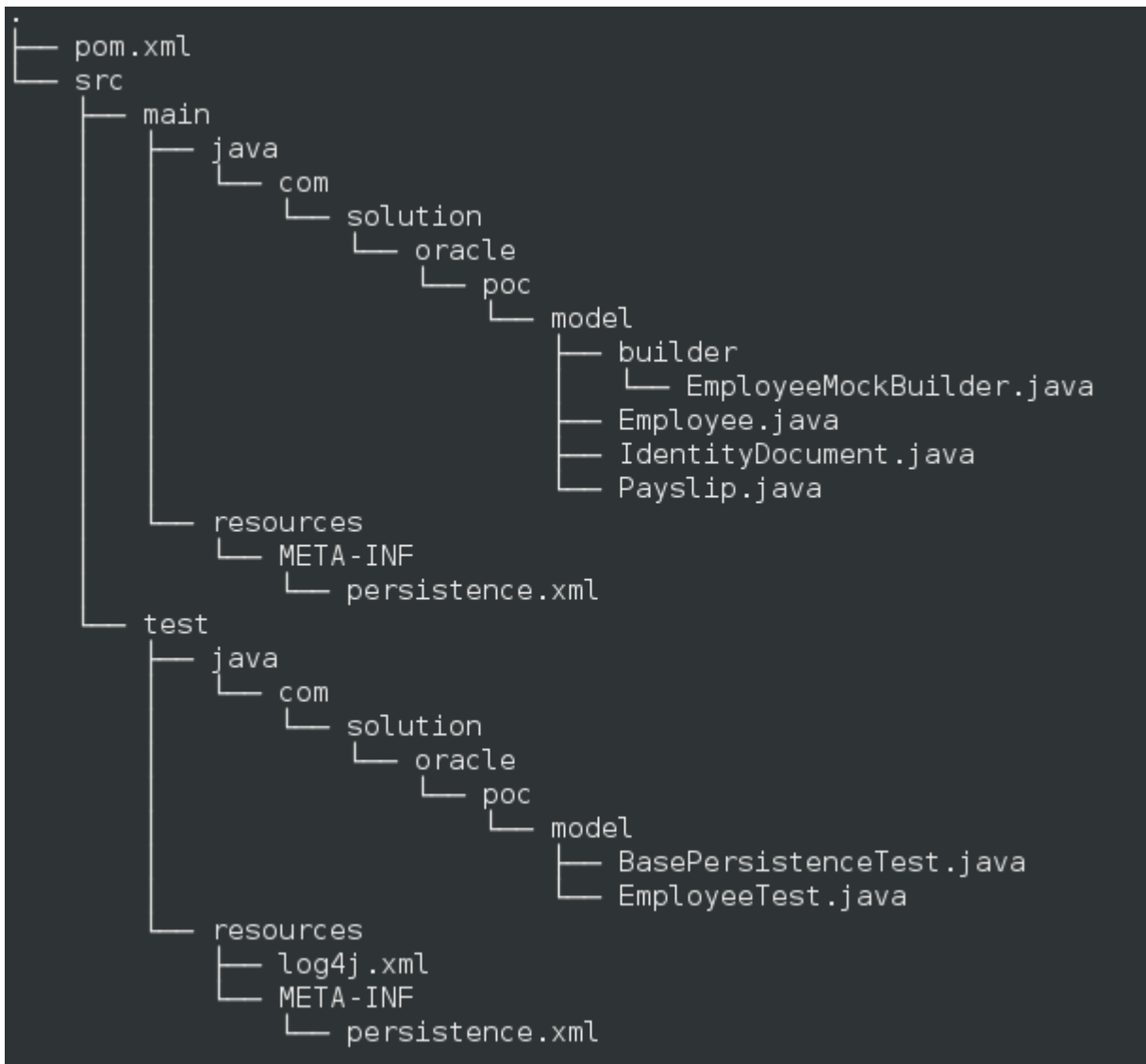
// Cierre.
t.close();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
```

Este servicio nos permite el envío de mail a los destinatarios del tipo Employee enviado por parámetro, que son los que entiende esta aplicación, se puede especificar un mensaje (este se escribe en formato html). Para esto se utiliza la librería de Java Mail en cual proporciona de forma rápida y amigable. JavaMail implementa el protocolo SMTP (Simple Mail Transfer Protocol) así como los distintos tipos de conexión con servidores de correo -TLS, SSL, autenticación con usuario y password, etc. Como se ve se esta configurando las propiedades desde el método, esto es solo conceptual, la forma correcta es exteriorizar dicha información en un archivo externo a la aplicación, ubicado en el classpath del mismo para que pueda ser leído como cualquier recurso.



## javaee-oracle-poc-model

### Estructura de directorios



## Employee

```
package com.solution.oracle.poc.model;

import java.io.Serializable;

import java.util.Date;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Embedded;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToOne;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

@Entity
@NamedQueries({
    @NamedQuery(
        name = "Employee.findAll",
        query="select e from Employee e"),
    @NamedQuery(
        name = "Employee.findByName",
        query="select e FROM Employee e WHERE e.lastName = :lastName"),
    @NamedQuery(
        name = "Employee.findByFileNumber",
        query="select e FROM Employee e WHERE e.fileNumber = :fileNumber")
})
@SequenceGenerator(name = "EMPLOYEE_SEQ", sequenceName = "EMPLOYEE_SEQ", allocationSize = 50,
initialValue = 50)
@Table(name = "EMPLOYEES")
public class Employee implements Serializable {

    @SuppressWarnings("compatibility:-4257518447367921997")
    private static final long serialVersionUID = 7205411068860065902L;
```

```
public static final String EMPLOYEE_FIND_ALL = "Employee.findAll";  
public static final String EMPLOYEE_FIND_BY_LAST_NAME = "Employee.findByName";  
public static final String EMPLOYEE_FIND_BY_FILE_NUMBER = "Employee.findByFileNumber";
```

```
@Id  
@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "EMPLOYEE_SEQ")  
@Column(name = "EMPLOYEE_ID")
```

```
private Integer id;
```

```
@Column(name = "NAME", nullable = false)  
private String name;
```

```
@Column(name = "SECOND_NAME")  
private String secondName;
```

```
@Column(name = "LAST_NAME", nullable = false)  
private String lastName;
```

```
@Column(name = "LAST_MAIL", nullable = false)  
private String mail;
```

```
@Column(name = "BIRTHDAY")  
@Temporal(TemporalType.DATE)  
private Date birthday;
```

```
@Column(name = "FILE_NUMBER", nullable = false, unique = true)  
private Long fileNumber;
```

```
@Embedded  
private IdentityDocument identityDocument;
```

```
@OneToMany(fetch = FetchType.LAZY, cascade = CascadeType.ALL)  
@JoinColumn(name = "EMPLOYEE_ID", referencedColumnName = "EMPLOYEE_ID")  
private List<Payslip> payslip;
```

```
public Employee() {  
}
```

```
public Integer getId() {  
    return id;  
}
```

```
public void setId(Integer id) {  
    this.id = id;  
}
```

```
... getters y setters
```

```
}
```

Este clase pojo, representa una entidad con sus respectivos atributos, como verán se utiliza anotaciones (se antepone con el arroba @), los cuales son necesarios para las configuraciones de JPA y no son intrusivas y restrictivas para el código. Cada entidad es representada con un `@Entity`, por lo que jpa generara su equivalente como tabla en la base de datos, en este caso la tabla se llamara “EMPLOYEES”, y con las columnas representadas por cada atributo, la primary key para este caso es aquella que tenga designada el `@Id`. Otro mapeo característico es la relación de dos entidades, jpa lo resuelve con la anotación `@OneToMany` descripta en el código, generando de forma transparente las relaciones entre las tablas.

## **Payslip**

```
package com.solution.oracle.poc.model;

import [...]

@Entity
@NamedQueries( { @NamedQuery(name = "Payslip.findAll", query = "select o from Payslip o") })
@SequenceGenerator(name = "PAYSLIPS_SEQ", sequenceName = "PAYSLIPS_SEQ", allocationSize = 50,
initialValue = 50)
@Table(name = "PAYSLIPS")
public class Payslip implements Serializable {

    @SuppressWarnings("compatibility:8629665482569191710")
    private static final long serialVersionUID = 2240752066451485170L;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "PAYSLIPS_SEQ")
    @Column(name = "PAYSLIP_ID")
    private Integer id;

    @Column(name = "SALARY", nullable = false)
    private Double salary;

    @Column(name = "PAYMENT_DATE", nullable = false)
    @Temporal(TemporalType.DATE)
    private Date paymentDate;

    public Payslip() {
    }

    public Integer getId() {
        return id;
    }
}
```

```

public void setId(Integer id) {
    this.id = id;
}

... getters y setters
}

```

## Persistence.xml

```

<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
    <persistence-unit name="javaee-oracle-poc" transaction-type="JTA">
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
        <!-- <jta-data-source>javaeeOraclePocMySqlDS</jta-data-source> -->
        <jta-data-source>javaeeOraclePocDerbyDS</jta-data-source>
        <properties>
            <property name="eclipselink.target-server" value="WebLogic_10"/>
            <property name="eclipselink.logging.level" value="FINEST"/>
            <!-- EclipseLink should create the database schema automatically -->
            <property name="eclipselink.ddl-generation" value="create-tables"/>
            <property name="eclipselink.ddl-generation.output-mode" value="database"/>
        </properties>
    </persistence-unit>
</persistence>

```

En este archivo se configura el data sources, permitirle a jpa regenerar las tablas cada ves que se despliega la aplicación en el servidor.