



Java Academy Xideral 2022

Alumno: Moisés Vidal Hernández

Maestro: Miguel Angel Rugerio

Java Bootcamp Training

Examen: Semana 3

Diciembre 9 del 2022

1. Realiza un programa con pruebas unitarias

Para este ejercicio tomaremos el código que hemos estado usando en algunos ejercicios anteriores, el cual consiste en una caseta de peaje.

Tenemos una clase vehículo que contendrá las características de un vehículo terrestre.

```
1 package com.ejerciciUnitTest;
2
3 public class Vehiculo {
4
5     private String marca;
6     private String modelo;
7     private String color;
8     private int ejes;
9
10    public Vehiculo (String marca, String modelo, String color, int ejes) {
11        this.marca = marca;
12        this.modelo = modelo;
13        this.color = color;
14        this.ejes = ejes;
15    }
16
17    public String getMarca() {
18        return marca;
19    }
20
21    public void setMarca(String marca) {
22        this.marca = marca;
23    }
24
25    public String getModelo() {
26        return modelo;
27    }
28
29    public void setModelo(String modelo) {
30        this.modelo = modelo;
31    }
32
33    public String getColor() {
34        return color;
35    }
36
37    public void setColor(String color) {
38        this.color = color;
39    }
40
41    public int getEjes() {
42        return ejes;
43    }
44
45    public void setEjes(int ejes) {
46        this.ejes = ejes;
47    }
```

También tenemos una clase Casetas, en la que tenemos 3 métodos que los pondremos a prueba por medio de las pruebas unitarias.

```
8⊕ public String cobrarPeaje(int ejes) {
9     String cadena = "";
10    if (ejes <= 0) {
11        cadena = "El vehiculo no se escaneo de manera correcta";
12    } else {
13        switch (ejes) {
14            case 1:
15                cadena = "El vehiculo es una motocicleta y paga 49.50 pesos";
16                break;
17            case 2:
18                cadena = "El vehiculo es un automovil y paga 89.50 pesos";
19                break;
20            case 3:
21                cadena = "El vehiculo es un autovbus y paga 149.50 pesos";
22                break;
23            case 4:
24                cadena = "El vehiculo es un camion de carga y paga 209.50 pesos";
25                break;
26            default:
27                cadena = "No se proceso su respuesta";
28                break;
29        }
30    }
31    return cadena;
32 }
33
34⊕ public ArrayList<Vehiculo> muestraVehiculosErroneos(List<Vehiculo> lvh) {
35     ArrayList<Vehiculo> vehiculosErroneos = null;
36     for (Vehiculo vh : lvh) {
37         if(vh.getEjes() <= 0 || vh.getEjes() > 4) {
38             System.out.println("entre en vehiculo "+ vh);
39             vehiculosErroneos.add(vh);
40         }
41     }
42     return vehiculosErroneos;
43 }
44
45⊕ public boolean compruebaVehiculosErroneos(List<Vehiculo> lvh) {
46     boolean vehiculosErroneos = false;
47     for (Vehiculo vh : lvh) {
48         if(vh.getEjes() <= 0 || vh.getEjes() > 4) {
49             vehiculosErroneos = true;
50         }
51     }
52     return vehiculosErroneos;
53 }
```

Posteriormente tenemos nuestra clase Test que lleva por nombre TestCaseta.

A continuación describimos cada test

- El primer test, comprueba que la función de cobrarPeaje, funcione de manera corrector, ya que al mandar que este vehículo tiene 2 ejes el resultado nos debe mandar un string diciendo el tipo de vehículo y la tarifa que debe pagar.

```
12@  @Test
13  void testCobrarPeaje() {
14      Vehiculo v1 = new Vehiculo("Ford", "Lobo", "Azul", 2);
15      Caseta caseta = new Caseta();
16      String cad1 = caseta.cobrarPeaje(v1.getEjes());
17      assertEquals(cad1, "El vehiculo es un automovil y paga 89.50 pesos");
18  }
```

- El siguiente test, evalua que la función cobrarPeaje(), nos retorne un String indicando que no se pudo procesar una respuesta ya que el número de ejes asignado rebasa el rango aceptado.

```
20@  @Test
21  void testCobrarPeaje1() {
22      Vehiculo v1 = new Vehiculo("Ford", "Lobo", "Azul", 2);
23      Caseta caseta = new Caseta();
24      v1.setEjes(9);
25      String cad1 = caseta.cobrarPeaje(v1.getEjes());
26      assertEquals(cad1, "No se proceso su respuesta");
27  }
```

- El tercer test comprueba que la respuesta del método cobrarPeaje(), nos retorne un string indicando que es un vehículo del tipo motocicleta.

```
29@  @Test
30  void testCobrarPeaje2() {
31      Vehiculo v1 = new Vehiculo("Ducati", "Scrambler", "Negro mate", 1);
32      Caseta caseta = new Caseta();
33      String cad1 = caseta.cobrarPeaje(v1.getEjes());
34      assertEquals(cad1, "El vehiculo es una motocicleta y paga 49.50 pesos")
35  }
```

- El cuarto test comprueba que el método cobrarPeaje(), nos devuelva un string indicando un mensaje de error, ya que si mandamos un número de ejes se determinara que es un vehículo con errores al escanear.

```
37@  @Test
38  void testCobrarPeaje3() {
39      Vehiculo v1 = new Vehiculo("Ducati", "Scrambler", "Negro mate", -34);
40      Caseta caseta = new Caseta();
41      String cad1 = caseta.cobrarPeaje(v1.getEjes());
42      assertEquals(cad1, "El vehiculo no se escaneo de manera correcta");
43  }
```

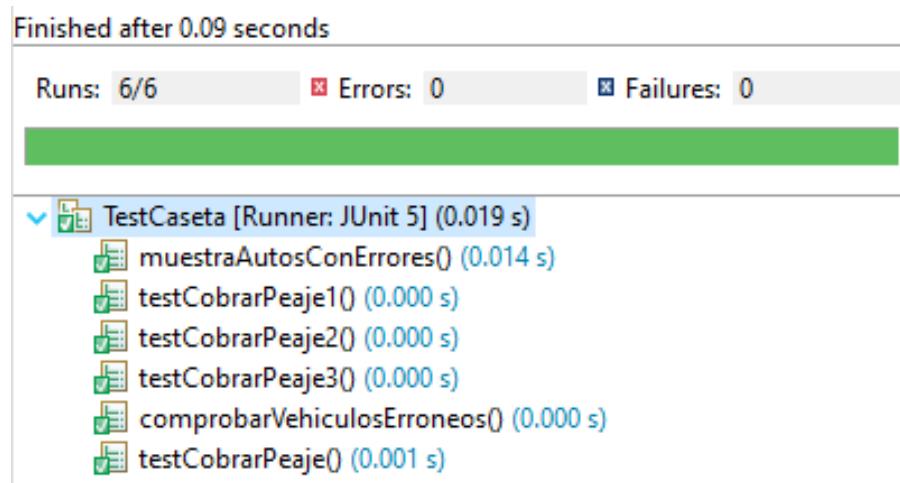
- El último comprueba si en la lista de vehículos existen vehículos con errores al escanear, en caso de que se encuentren se retornara un true entonces este test se cumplirá

```

    @Test
    void comprobarVehiculosErroneos() {
        Casetta caseta = new Casetta();
        List<Vehiculo> listaVehiculos = new ArrayList<>();
        listaVehiculos.add(new Vehiculo("Dodge", "Charger 400", "Negro", 2));
        listaVehiculos.add(new Vehiculo("Volkswagen", "Gol", "Rojo", 2));
        listaVehiculos.add(new Vehiculo("Scannia", "K360C B4x2NB", "Azul", 3));
        listaVehiculos.add(new Vehiculo("Ford", "Fiesta 2012", "Verde", 2));
        listaVehiculos.add(new Vehiculo("Yamaha", "R6", "Azul c Gris", 1));
        listaVehiculos.add(new Vehiculo("Chevrolet", "Silverado", "Blanco", 2));
        listaVehiculos.add(new Vehiculo("Kenworth", "T680", "Amarillo", 4));
        listaVehiculos.add(new Vehiculo("Man", "MAN TGS 33.540", "Negro", 3));
        listaVehiculos.add(new Vehiculo("Kawasaki", "Ninja 400", "Verde", 1));
        listaVehiculos.add(new Vehiculo("Ford", "Lobo 2021", "Negra", 2));
        boolean vehiculosErroneos = caseta.compruebaVehiculosErroneos(listaVehiculos);
        assertEquals(true, true);
    }
}

```

Entonces una vez ejecutados nuestros test, todos pasaran justo como se muestra a continuación.



2. Realiza un resumen de la implementación de SCRUM (Apendice)

Para desarrollar un proyecto aplicando la metodología SCRUM, requerimos de 11 puntos clave e importantes, los cuales presentamos a continuación.

1. Necesitamos a un responsable de producto, al cual también se suele nombrar “Product Owner”, la persona que asuma este rol tendrá la responsabilidad de evaluar riesgos y beneficios que se obtendrán al llevar a cabo ciertas acciones en el desarrollo del proyecto.
2. Requerimos un grupo de personas de alrededor unas 9 personas, las cuales tendrán la responsabilidad de realizar el trabajo de manera eficiente y efectiva, las personas que integran este equipo deben poseer ciertas habilidades específicas.
3. Necesitamos a una persona que ayude y capacite al equipo en torno al enfoque Scrum, esta persona tomará el rol de “Scrum Master” y su función primordial es eliminar todas las posibles trabas que presente el equipo.
4. Es necesario crear una bitácora en la cual están listadas todas las actividades que engloban el desarrollo del proyecto. Esta bitácora sufriría cambios y modificaciones conforme se progrese en el desarrollo. Llevará por nombre “Bitácora del producto”, solo debe de existir una, y quien actúa en función de la importancia de las actividades enlistadas es el Product Owner también se puede auxiliar a los involucrados en el desarrollo para asignar prioridad a cada tarea.
5. Este punto va muy de la mano con el anterior, ya que en este se estima la importancia de cada actividad, se determina si la información y los recursos que se proporcionan son realmente suficientes para llevar a cabo dichas tareas o actividades, justo como lo dijimos en el punto anterior aquí es preferible que todos los involucrados participen para que todos estén de acuerdo y no existan incongruencias.
6. Debemos hacer reuniones o meets, con una duración fija, habitualmente se hacen 2 a la semana, estas reuniones llevan por nombre “Sprints”, en cada reunión que se efectúe, se irán sumando puntos y esos son considerados como la velocidad del equipo. Por medio de este puntaje se demuestra que todos los implicados tienen la misma visión y cumplirán los objetivos en común. En cada Sprint se plantea una meta que debe ser cumplida dentro del Sprint que está transcurriendo.
7. Debemos hacer visible y organizadas todas las actividades a desarrollar, por lo regular se organizan en una tabla con 3 columnas en la que cada una lleva su título respectivamente como se muestra a continuación, To do, doing y done.
 - En la columna “to do”, se encuentran aquellas actividades que aún no se están desarrollando.
 - En la columna “doing”, se encuentran las actividades que se están llevando a cabo en ese momento.
 - En la columna “done”, se encuentran aquellas actividades que ya han sido realizadas y están a la espera de la liberación o evaluación.
8. Scrums diarios, son pequeñas reuniones que no rebasan los 15 minutos, en estas reuniones cada miembro del equipo contesta estas 3 preguntas.
 - ¿Qué hiciste el día de ayer?

- ¿Qué harás el día de hoy?
- ¿Tienes algún obstáculo o algo que no te permita continuar?

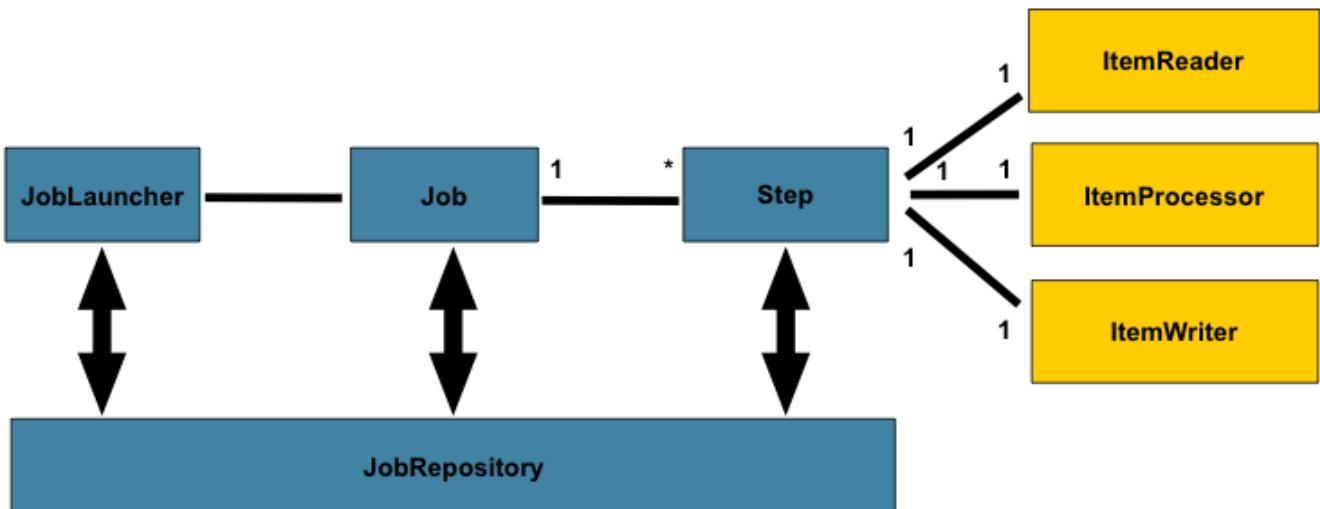
En caso de que algún miembro del equipo tenga un obstáculo, los demás miembros tienen la responsabilidad de ayudarlo, para que este pueda continuar con sus actividades.

9. Se debe concretar y llevar a cabo una reunión en la cual todos los interesados en el desarrollo, deberán participar: Product Owner, Scrum Master, el equipo y clientes, esta reunión lleva por nombre “Revisión de sprint”. Durante esta reunión se muestran los avances en calidad de terminado con el objetivo de mostrar el valor del desarrollo al cliente.
10. Una vez mostrados los avances al cliente, se hace una nueva reunión que lleva por nombre “Retrospectiva del sprint”, en esta reunión se realiza un análisis y un juicio del proceso sobre el cual se desarrollaron las actividades, con el objetivo de evaluar que se hizo bien y que se puede mejorar.
11. Comienza el ciclo del siguiente sprint a partir del retrospective sprint, con la finalidad de aplicar una mejora continua.

3. Explicar el siguiente diagrama Spring Batch

Spring batch es un framework open source desarrollado por SpringSource (Pivotal) y Accenture, dicho framework permite el procesamiento por lotes (manejar información, transacciones y estadísticas en grandes volúmenes sin la intervención humana).

En la siguiente imagen podemos apreciar los componentes de Spring Batch que explicaremos a continuación.



JobRepository

- Se trata de un repositorio que funciona de manera persistente, cuya función principal es realizar la escritura y verificar si un fichero se ha procesado previamente.
- Otra de sus funciones radica en re-procesar un job fallido y no todo un fichero.
- Específicamente el JobRepository escribe y consulta sobre las tablas de una base de datos transaccional, la responsabilidad del JobRepository es almacenar la información sobre cada job o step que se produzca, los parámetros del job y los errores que surjan en cada uno.

JobLauncher

- Este se encarga de lanzar los procesos suministrando los parámetros de entrada deseados o requeridos.

Job

- Un job (proceso) es un bloque de trabajo compuesto por uno o varios step (pasos).
- Una vez ejecutados todos los pasos el job es considerado como completado.

Step

- Es un elemento independiente dentro de un Job, el cual representa una de las fases de las que está compuesto un Job
- Un Job debe tener al menos un step.
- Igualmente un step puede estar compuesto de tres elementos los cuales se describen a continuación:
 - **ItemReader:** Elemento responsable de leer datos desde una fuente de datos como lo puede ser una base de datos o un fichero.
 - **ItemProcessor:** Elemento responsable de tratar la información leída por el reader.
 - **ItemWriter:** Elemento responsable de guardar la información leída por el reader o tratada por el processor (si existe un reader obligatoriamente debe existir un writer).

4. Desarrolla un crud web con Servlets y JSP

Tomando de referencia el proyecto proporcionado, comenzamos con el desarrollo y actualización del proyecto web aplicando servlets, jsp's y jdbc en java.

Este proyecto originalmente consiste en un crud web de alumnos, nosotros decidimos adaptarlo a candidatos que se postulan a ofertas de trabajo.

A continuación describimos cada parte destacable que permiten la actualización de este proyecto.

- Una vez comprendido el contexto o tema en el que gira nuestro proyecto, primero nos disponemos a crear los script en lenguaje de base de datos SQL, para posteriormente ejecutarlos, por medio de estos scripts realizaremos la creación del usuario, base de datos, tabla e inserción inicial de datos. De tal forma que nuestro script SQL queda de la siguiente forma.

```
CREATE USER 'recruiter' '@localhost' IDENTIFIED BY 'recruiter';

GRANT ALL PRIVILEGES ON *.* TO 'recruiter' '@localhost';
S
CREATE DATABASE IF NOT EXISTS `web_candidates`;

USE `web_candidates`;

DROP TABLE IF EXISTS `candidates`;

CREATE TABLE `candidates` (
    `id` INT(11) NOT NULL AUTO_INCREMENT,
    `name` VARCHAR(45) DEFAULT NULL,
    `last_name` VARCHAR(45) DEFAULT NULL,
    `email` VARCHAR(45) DEFAULT NULL,
    `phone` VARCHAR(10) DEFAULT NULL,
    `department` VARCHAR(45) DEFAULT NULL,
    `degree` VARCHAR(45) DEFAULT NULL,
    PRIMARY KEY(`id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=latin1;

INSERT INTO `candidates` VALUES (1, 'Mery', 'Jane', 'meryjein@mail.com', '1234567890', 'Human Resources', 'psychologist'),
                                (2, 'Piter', 'Parker', 'spiderman@mail.com', '1234567890', 'Daily news', 'Photographer'),
                                (3, 'Clark', 'Kent', 'superman@mail.com', '1234567890', 'Daily Planet', 'Reporter'),
                                (4, 'Matt', 'Murdock', 'ddevil@mail.com', '1234567890', 'Legales', 'Lawyer');
```

- Para comprobar nuestra base de datos podemos ir a nuestro gestor y veremos que existe una base de datos con el nombre “web_candidates”, que a su vez aloja la tabla “candidates” y dentro de esta existen los registros:

The screenshot shows the MySQL Workbench interface. On the left, the database tree displays 'information_schema', 'mysql', 'performance_schema', 'phpmyadmin', 'test', and 'web_candidates'. Under 'web_candidates', there is a 'Nueva' folder containing a 'candidates' table. The main pane shows a results grid for the 'candidates' table with the following data:

	id	name	last_name	email	phone	department	degree
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	2	Piter	Parker	spiderman@mail.com	1234567890	Daily news	Photographer
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	3	Clark	Kent	superman@mail.com	1234567890	Daily Planet	Reporter
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	4	Matt	Murdock	ddevil@mail.com	1234567890	Legales	Lawyer
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	7	Luisa	Lane	lane@mail.com	83736357	Daily Planet	Reporter
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	8	Diana	Prince	wonderw@gmail.com	84847463	Justice League	Amazona
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	9	Artur	Curri	arr@gmail.com	8823724364	Justice League	Heroe
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	10	Linus	Torvalds	linux@linuxmail.com	88373653	Developer	Programmer
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	14	Victor	Stone	cycborg@jgmail.com	737363534	Justice League	Heroe
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	15	Alex	Gel	memr@gmail.com	084287243	Product Owner	Enginner
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	17	Eliot	Alderson	elior_alderson@evil.corp.com	8383764	Security	Security Analyst
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	19	Mr	Robot	mrobovoto@mail.com	3838336	Hacker	Hacker
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	20	Mark	Zukaritas	km@gmail.com	837464465	Systemaas	Facebook CEO

- Una vez que nuestra base de datos ha sido construida de manera exitosa, procedemos a configurar el contexto, el cual se encuentra en el archivo context.xml, que esta alojado en la carpeta src/main/webapp/META-INF/context.xml.
- Dentro de este archivo configuraremos la conexión, la forma de hacerlo es especificando el driver, nombre de la conexión, usuario, contraseña y el url.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <Context path="/web_candidates">
3   <Resource auth="Container"
4     driverClassName="com.mysql.cj.jdbc.Driver"
5     maxActive="20" maxIdle="5" maxWait="10000"
6     name="jdbc/web_candidates"
7     password="recruiter"
8     type="javax.sql.DataSource"
9     url="jdbc:mysql://localhost:3306/web_candidates?useSSL=false&serverTimezone=UTC"
10    username="recruiter"/>
11 </Context>
12

```

- Una vez configurado el contexto, procedemos a la creación de nuestro objeto Candidate que tendrá sus atributos y sus métodos de acceso, los cuales nos servirán para el manejo de la información con la base de datos.

```

3 public class Candidate {
4
5   private int id;
6   private String name;
7   private String lastName;
8   private String email;
9   private String phone;
10  private String department;
11  private String degree;
12
13  public Candidate(int id, String name, String lastName, String email, String phone, String department,
14                    String degree) {
15    this.id = id;
16    this.name = name;
17    this.lastName = lastName;
18    this.email = email;
19    this.phone = phone;
20    this.department = department;
21    this.degree = degree;
22  }
23
24  public Candidate(String name, String lastName, String email, String phone, String department, String degree) {
25    this.name = name;
26    this.lastName = lastName;
27    this.email = email;
28    this.phone = phone;
29    this.department = department;
30    this.degree = degree;
31  }
32
33
34
35  public int getId() {
36    return id;
37  }
38
39  public void setId(int id) {
40    this.id = id;
41  }
42
43  public String getName() {
44    return name;
45  }
46
47  public void setName(String name) {
48    this.name = name;
49  }

```

- Luego actualizamos el controller servlet cambiando el nombre, nombre de métodos valores por parámetro y retorno, pero orientado a candidatos.

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

    private static final long serialVersionUID = 1L;

    private CandidateDbUtil candidateDbUtil;

    @Resource(name = "jdbc/web_candidates") //SE COMENTO PARA HACER USO DE JNDI
    private DataSource dataSource;

    @Override
    public void init() throws ServletException {
        super.init();
        // create our student db util ... and pass in the conn pool / datasource
        try {
            //https://www.digitalocean.com/community/tutorials/tomcat-datasource-jndi-example-java
            /*Context ctx = new InitialContext(); //USO DE JNDI
            dataSource = (DataSource) ctx.lookup("java:/comp/env/jdbc/javatechie"); //USO DE JNDI
            System.out.println("Demo con JNDI, Datasource: "+dataSource);*/
            candidateDbUtil = new CandidateDbUtil(dataSource);
        } catch (Exception exc) {
            throw new ServletException(exc);
        }
    }
}
```

- En el doGet, mandamos el comando y de acuerdo a este comando se realizará cierta acción ya se listar a los candidatos, mostrar el formulario para eliminar o actualizar y eliminar un candidato.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    try {
        // read the "command" parameter
        String theCommand = request.getParameter("command");

        // if the command is missing, then default to listing students
        if (theCommand == null) {
            theCommand = "LIST";
        }

        // route to the appropriate method
        switch (theCommand) {

            case "LIST":
                listCandidates(request, response);
                break;

            case "ADD":
                addCandidate(request, response);
                break;

            case "LOAD":
                loadCandidate(request, response);
                break;

            case "UPDATE":
                updateCandidate(request, response);
                break;

            case "DELETE":
                deleteCandidate(request, response);
                break;

            default:
                listCandidates(request, response);
        }
    } catch (Exception exc) {
        throw new ServletException(exc);
    }
}
```

- En el método deleteCandidate, hacemos la búsqueda del registro a eliminar mediante el id, si este es encontrado se eliminará dicho registro.
- El método updateCandidate, hace la búsqueda de la misma forma como el deleteCandidate, busca al candidato mediante su id de registro, si es encontrado se construyen variables que tomarán la información que se extrae de la base de datos y captura los cambios una vez capturada toda la información se crea un nuevo objeto con la información actualizada y se realiza el update directo en la Bd.

```

private void deleteCandidate(HttpServletRequest request, HttpServletResponse response)
    throws Exception {

    // read student id from form data
    String candidateId = request.getParameter("candidateId");

    // delete student from database
    candidateDbUtil.deleteCandidate(candidateId);

    // send them back to "list students" page
    listCandidates(request, response);
}

private void updateCandidate(HttpServletRequest request, HttpServletResponse response)
    throws Exception {

    // read student info from form data
    int id = Integer.parseInt(request.getParameter("candidateId"));
    String firstName = request.getParameter("name");
    String lastName = request.getParameter("lastName");
    String email = request.getParameter("email");
    String phone = request.getParameter("phone");
    String department = request.getParameter("department");
    String degree = request.getParameter("degree");

    // create a new student object
    Candidate theCandidate = new Candidate(id, firstName, lastName, email, phone, department, degree);
    System.out.println("En update" + theCandidate);

    // perform update on database
    candidateDbUtil.updateCandidate(theCandidate);

    // send them back to the "list students" page
    listCandidates(request, response);
}

```

- El método loadCandidate, nos sirve para mandar la información del candidato seleccionado desde la tabla del jsp y mandarla a la siguiente ruta que es la actualización de la información del candidato (la jsp “update-candidate-form”).

```

private void loadCandidates(HttpServletRequest request, HttpServletResponse response)
    throws Exception {

    // read student id from form data
    String theCandidateId = request.getParameter("candidateId");

    // get student from database (db util)
    Candidate theCandidate = candidateDbUtil.getCandidate(theCandidateId);

    // place student in the request attribute
    request.setAttribute("THE_CANDIDATE", theCandidate);

    // send to jsp page: update-student-form.jsp
    RequestDispatcher dispatcher
        = request.getRequestDispatcher("/update-candidate-form.jsp");
    dispatcher.forward(request, response);
}

```

- El método addCandidate, permite dar de alta nuevos registro dentro de la bd, se hace mediante la creación de un nuevo objeto donde se especifica la información a insertar, la información recibe desde la vista (el jsp “add-candidate-form”)

```

private void addCandidate(HttpServletRequest request, HttpServletResponse response) throws Exception {

    // read student info from form data
    String firstName = request.getParameter("name");
    String lastName = request.getParameter("lastName");
    String email = request.getParameter("email");
    String phone = request.getParameter("phone");
    String department = request.getParameter("department");
    String degree = request.getParameter("degree");

    // create a new student object
    Candidate theCandidate = new Candidate(firstName, lastName, email, phone, department, degree);
    System.out.println("theCandidate = " + theCandidate);
    // add the student to the database
    System.out.println("theCandidate = " + theCandidate);
    candidateDbUtil.addCandidate(theCandidate);

    // send back to main page (the student list)
    listCandidates(request, response);
}

```

- El método listCandidates, obtiene todos los registro de la BD y los manda al jsp “list-candidates”, dentro de este método se especifica la información que se desea consultar.

```

private void listCandidates(
    HttpServletRequest request, HttpServletResponse response)
    throws Exception {

    // get students from db util
    List<Candidate> candidates = candidateDbUtil.get Candidates();

    for (Candidate c : candidates) {
        System.out.println(c);
    }

    candidates.add(new Candidate(999, "Name", "LastName", "Email", "Phone", "Department", "Degree"));
    // add students to the request
    request.setAttribute("LISTA_CANDIDATOS", candidates);

    // send to JSP page (view)
    RequestDispatcher dispatcher = request.getRequestDispatcher("/list-candidates.jsp");
    dispatcher.forward(request, response);
}

```

- e

- En la clase CandidateDbUtil, se concentran los métodos que pre procesan las instrucciones en SQL, estas instrucciones nos permitirán manejar la información de acuerdo a lo que necesitamos (agregar, visualizar, modificar o eliminar).

```

public class CandidateDbUtil {

    private DataSource dataSource;

    public CandidateDbUtil(DataSource theDataSource) {
        dataSource = theDataSource;
    }

    public List<Candidate> getCandidates() throws Exception {
        List<Candidate> candidates = new ArrayList<>();

        Connection myConn = null;
        Statement myStmt = null;
        ResultSet myRs = null;

        try {
            // get a connection
            myConn = dataSource.getConnection();

            // create sql statement
            String sql = "select * from candidates order by last_name";

            myStmt = myConn.createStatement();

            // execute query
            myRs = myStmt.executeQuery(sql);

            // process result set
            while (myRs.next()) {

                // retrieve data from result set row
                int id = myRs.getInt("id");
                String firstName = myRs.getString("name");
                String lastName = myRs.getString("last_name");
                String email = myRs.getString("email");
                String phone = myRs.getString("phone");
                String department = myRs.getString("department");
                String degree = myRs.getString("degree");

                // create new student object
                Candidate tempCandidate = new Candidate(id, firstName, lastName, email, phone, department, degree);

                // add it to the list of students
                candidates.add(tempCandidate);
            }
        }
    }
}

```

- Ahora pasando a nuestra JSP, dentro de esta tenemos código HTML y código Java, este jsp tiene un refactoring ya que se pensó darle más estilo y hacer una presentación más atractiva, además se hicieron modificaciones para el cómo mostrar la información.

```

CandidateDbUtil.java  list-candidates.jsp ×
1 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
2 <!DOCTYPE html>
3<html>
4<head>
5 <title>Candidates Tracker App</title>
6 <link rel="icon" type="image/x-icon"
7   href="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTfnBVPMPfBILqQFAmEdcAkJGwzQUFEmDxXJA&usqp=CAU">
8 <link type="text/css" rel="stylesheet" href="css/style.css">
9 </head>
10<body>
11    <div class="container-title">
12      <h2 class="title-app">Candidates for job</h2>
13    </div>
14    <div class="container">
15      <div class="content center">
16        <!-- put new button: Add Student -->
17        <input class="btn-add" type="button" value="Add Candidate"
18          onclick="window.location.href='add-candidate-form.html'; return false;" />
19      <table class="table-content center">
20        <tr>
21          <th>First Name</th>
22          <th>Last Name</th>
23          <th>Email</th>
24          <th></th>
25          <th></th>
26        </tr>
27        <c:forEach var="tempCandidate" items="${LISTA_CANDIDATOS}">
28          <!-- set up a link for each candidate -->
29          <c:url var="tempLink" value="CandidateControllerServlet">
30            <c:param name="command" value="LOAD" />
31            <c:param name="candidateId" value="${tempCandidate.id}" />
32          </c:url>
33          <!-- set up a link to delete a student -->
34          <c:url var="deleteLink" value="CandidateControllerServlet">
35            <c:param name="command" value="DELETE" />
36            <c:param name="candidateId" value="${tempCandidate.id}" />
37          </c:url>
38          <tr>
39            <td>${tempCandidate.name}</td>
40            <td>${tempCandidate.lastName}</td>
41            <td>${tempCandidate.email}</td>
42            <td><a class="btn-update" href="${tempLink}">Show More</a></td>
43            <td><a class="btn-delete" href="${deleteLink}"
44              onclick="if (!confirm('Are you sure you want to delete this candidate?')) return false">
45                Delete</a></td>
46          </tr>
47        </c:forEach>

```

- La vista final de nuestro JSP, sería la siguiente

Candidates for job				
First Name		Last Name	Email	
Eliot	Alderson		elior_alderson@evil.corp.com	Show More Delete
Artur	Curri		arq@gmail.com	Show More Delete
Alex	Gel		memr@gmail.com	Show More Delete
Clark	Kent		superman@mail.com	Show More Delete
Luisa	Lane		lane@mail.com	Show More Delete
Matt	Murdock		ddevil@mail.com	Show More Delete
Piter	Parker		spaiderman@mail.com	Show More Delete
Diana	Prince		wonderw@gmail.com	Show More Delete
Mr	Robot		mrrobovoto@mail.com	Show More Delete
Miguel	Rugerio		mg@gmail.com	Show More Delete
Victor	Stone		cyborg@jlmail.com	Show More Delete
Linus	Torvalds		linux@linuxmail.com	Show More Delete
Mark	Zukaritas		km@gmail.com	Show More Delete
Name	LastName	Email		Show More Delete

- Posteriormente el JSP para agregar candidatos también sufrió modificaciones para darle un mejor estilo, además que en este se solicita la información completa del candidato.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Add Student</title>
5  <link type="text/css" rel="stylesheet" href="css/style.css">
6  <link type="text/css" rel="stylesheet" href="css/add-student-style.css">
7  </head>
8  <body>
9  <div class="container-title">
10 <h2 class="title-app">Candidates for job</h2>
11 </div>
12 <div class="container">
13 <form class="form center" action="CandidateControllerServlet"
14 method="GET">
15 <h3 class="title-form">Add Candidate</h3>
16 <input class="text-field" type="hidden" name="command" value="ADD" />
17 <input class="text-field" type="text" name="name" placeholder="First Name" />
18 <input class="text-field" type="text" name="lastName" placeholder="Last Name" />
19 <input class="text-field" type="text" name="email" placeholder="E-mail" />
20 <input class="text-field" type="text" name="phone" placeholder="Phone" />
21 <input class="text-field" type="text" name="department" placeholder="Department of job" />
22 <input class="text-field" type="text" name="degree" placeholder="Degreeel" />
23 <input class="save center" type="submit" value="Save" />
24 <a class="btn-back center" href="CandidateControllerServlet">Back to List</a>
25 </form>
26 </div>
27 </body>
28 </html>
```

- La vista final del JSP para agregar candidatos sería la siguiente.

Candidates for job

Add Candidate

First Name
Last Name
E-mail
Phone
Department of job
Degree

[Back to List](#) [Save](#)

accenture

- Por último tenemos el JSP para actualizar información de candidatos, este archivo también tuvo que sufrir un refactoring para aplicar estilo y presentar la información detallada de cada candidato.

Candidates for job

Update Candidate

First Name
Last Name
E-mail
Phone
Department
Degree

Eliot
Alderson
elior_alderson@evil.corp.com
8383764
Security
Security Analyst

[Back to List](#) [Save](#)

accenture

5. Desarrolla un crud web con Spring e Hibernate.

Al igual que el proyecto anterior de Servlets y JSP, Tomamos de referencia un proyecto proporcionado, comenzamos con el desarrollo y actualización del proyecto web aplicando spring e hibernate.

Este proyecto originalmente consiste en un crud web de alumnos, nosotros decidimos adaptarlo ahora a un inventario de productos de una tienda.

A continuación describimos cada parte destacable que permiten la actualización de este proyecto.

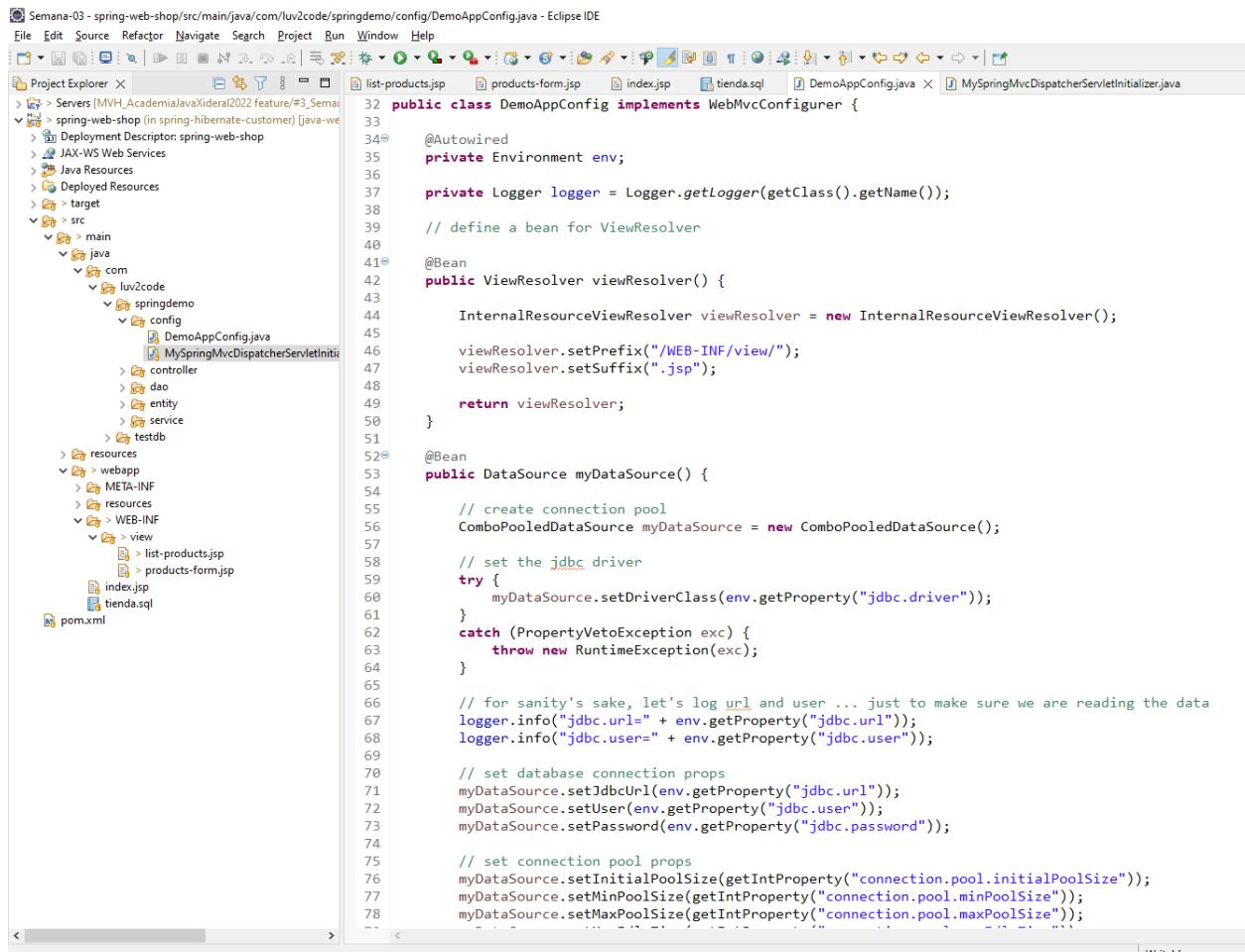
- Una vez comprendido el contexto o tema en el que gira nuestro proyecto, primero nos disponemos a crear los script en lenguaje de base de datos SQL, para posteriormente ejecutarlos, por medio de estos scripts realizaremos la creación del usuario, base de datos, tabla e inserción inicial de datos. De tal forma que nuestro script SQL queda de la siguiente forma.

```
1 CREATE DATABASE IF NOT EXISTS `web_shop` /*!40100 DEFAULT CHARACTER SET latin1 */;
2
3 USE `web_shop`;
4
5 CREATE TABLE `products` (
6     `id` INT(11) NOT NULL AUTO_INCREMENT,
7     `name` VARCHAR(45) DEFAULT NULL,
8     `description` VARCHAR(120) DEFAULT NULL,
9     `content` VARCHAR(45) DEFAULT NULL,
10    `price` VARCHAR(45) DEFAULT NULL,
11    `date_register` DATE DEFAULT NULL,
12    `date_expiration` DATE DEFAULT NULL,
13    PRIMARY KEY (`id`)
14 ) ENGINE = InnoDB AUTO_INCREMENT = 6 DEFAULT CHARSET = latin1;
15
16 /*INSERT VALUES IN TABLE products*/
17 INSERT INTO `products`
18     (name, description, content, price, date_register, date_expiration)
19     VALUES
20     ('Chetos','flaming hot','50g','$5.00','2008-7-04','2008-7-04'),
21     ('computadora','HP pavillion','1pza','$17,000.00','2008-7-04','2008-7-04'),
22     ('Café','nescafe molido','300g','$80.00','2008-7-04','2008-7-04'),
23     ('pantalon','sahara mezclilla','50g','$5.00','2008-7-04','2008-7-04'),
24     ('guitarra','fender acustica','50g','$5.00','2008-7-04','2008-7-04'),
25     ('Zucaritas','Cerial xD','250gr.', '50.0', '2022-12-08', '2022-12-22'),
26     ('Takis', 'Taquitos picantes con chile del que si pica xD', '67gr.', '15', '2022-12-08', '2023-01-21');
```

- Para comprobar nuestra base de datos podemos ir a nuestro gestor y veremos que existe una base de datos con el nombre “web_shop”, que a su vez aloja la tabla “products” y dentro de esta existen los registros:

	↓	id	name	description	content	price	date_register	date_expiration	
<input type="checkbox"/>		Editar		Copiar		Borrar	6 Chetos	flaming hot	50g \$5.00 2008-07-04 2008-07-04
<input type="checkbox"/>		Editar		Copiar		Borrar	7 computadora	HP pavillion	1pza \$17,000.00 2008-07-04 2008-07-04
<input type="checkbox"/>		Editar		Copiar		Borrar	9 pantalon	sahara mezclilla	50g \$5.00 2008-07-04 2008-07-04
<input type="checkbox"/>		Editar		Copiar		Borrar	11 Zucaritas	Cerial xD	250gr. 50.0 2022-12-08 2022-12-22
<input type="checkbox"/>		Editar		Copiar		Borrar	12 Takis	Taquitos picantes con chile del que si pica xD	67gr. 15 2022-12-08 2023-01-21
<input type="checkbox"/>		Editar		Copiar		Borrar	13 Sabritas	Frituras papas	35g. 12 2022-12-06 2022-12-13
<input type="checkbox"/>		Editar		Copiar		Borrar	14 Doritos	Frituras	34g 15 2022-12-06 2022-12-12
<input type="checkbox"/>		Editar		Copiar		Borrar	15 Sopa Maruchan Camaron	Sopa instantanea sabor camaron	23gr 34 2022-12-12 2023-04-10

- En el paquete config, vienen todos los métodos que establecen la conexión con la base de datos, parámetros y credenciales de acceso.



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure for "Semana-03 - spring-web-shop". It includes:
 - Servers [MWH_AcademiaJavaXideral2022 feature#_3_Semana]
 - Deployment Descriptor: spring-web-shop [java-ws]
 - JAX-WS Web Services
 - Java Resources
 - Deployed Resources
 - target
 - src
 - main
 - java
 - luv2code
 - springdemo
 - config
 - DemoAppConfig.java
 - MySpringMvcDispatcherServletInitializer.java
 - controller
 - dao
 - entity
 - service
 - testdb
 - resources
 - webapp
 - META-INF
 - resources
 - WEB-INF
 - view
 - list-products.jsp
 - products-form.jsp
 - index.jsp
 - tienda.sql
- pom.xml

- Code Editor:** Displays the content of `DemoAppConfig.java`. The code implements `WebMvcConfigurer` and defines a `ViewResolver` and a `DataSource`.

```

32 public class DemoAppConfig implements WebMvcConfigurer {
33
34     @Autowired
35     private Environment env;
36
37     private Logger logger = Logger.getLogger(getClass().getName());
38
39     // define a bean for ViewResolver
40
41     @Bean
42     public ViewResolver viewResolver() {
43
44         InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();
45
46         viewResolver.setPrefix("/WEB-INF/view/");
47         viewResolver.setSuffix(".jsp");
48
49         return viewResolver;
50     }
51
52     @Bean
53     public DataSource myDataSource() {
54
55         // create connection pool
56         ComboPooledDataSource myDataSource = new ComboPooledDataSource();
57
58         // set the jdbc driver
59         try {
60             myDataSource.setDriverClass(env.getProperty("jdbc.driver"));
61         } catch (PropertyVetoException exc) {
62             throw new RuntimeException(exc);
63         }
64
65         // for sanity's sake, let's log url and user ... just to make sure we are reading the data
66         logger.info("jdbc.url=" + env.getProperty("jdbc.url"));
67         logger.info("jdbc.user=" + env.getProperty("jdbc.user"));
68
69         // set database connection props
70         myDataSource.setJdbcUrl(env.getProperty("jdbc.url"));
71         myDataSource.setUser(env.getProperty("jdbc.user"));
72         myDataSource.setPassword(env.getProperty("jdbc.password"));
73
74         // set connection pool props
75         myDataSource.setInitialPoolSize(getIntProperty("connection.pool.initialPoolSize"));
76         myDataSource.setMinPoolSize(getIntProperty("connection.pool.minPoolSize"));
77         myDataSource.setMaxPoolSize(getIntProperty("connection.pool.maxPoolSize"));
78     }
    
```

- En la clase de productController, tenemos todos los métodos que utiliza hibernate para realizar las diferentes operaciones (agregar, modificar, consultar y eliminar), antes de cada método se asigna una notación que delimita el tipo petición y especifica la ruta a donde se dirige la página.

```

public class ProductsController {

    // need to inject our customer service
    @Autowired
    private ProductsService customerService;

    @GetMapping("/list")
    public String listCustomers(Model theModel) {
        // get customers from the service
        List<Products> theProducts = customerService.getCustomers();
        // add the customers to the model
        theModel.addAttribute("products", theProducts);
        return "list-products";
    }

    @GetMapping("/showFormForAdd")
    public String showFormForAdd(Model theModel) {
        // create model attribute to bind form data
        Products theProducts = new Products();
        theModel.addAttribute("products", theProducts);
        return "products-form";
    }

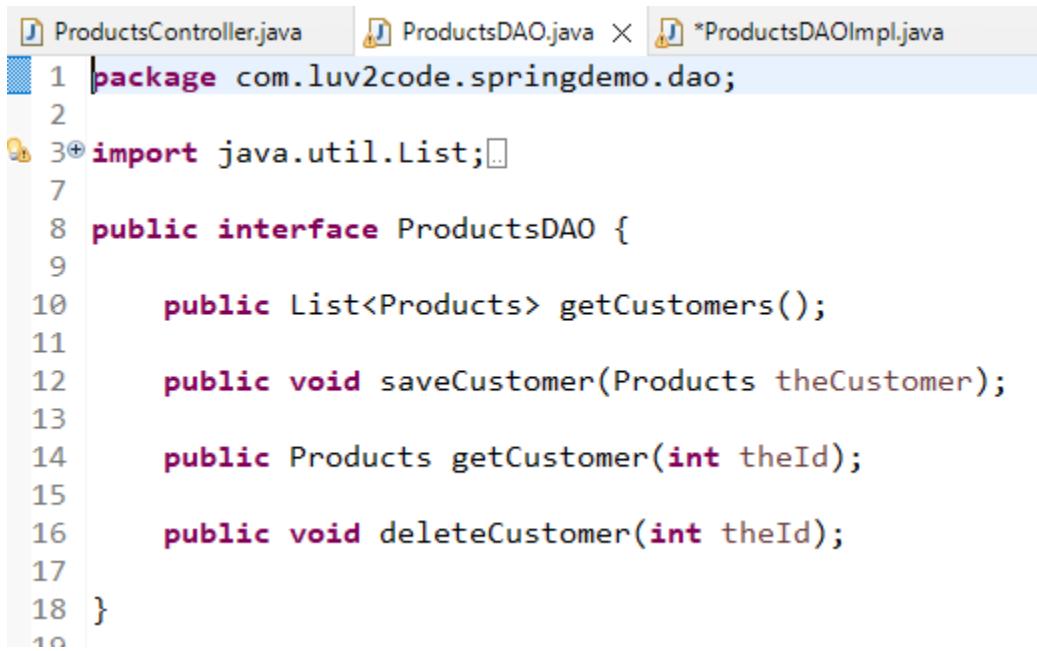
    @PostMapping("/saveCustomer")
    public String saveCustomer(@ModelAttribute("products") Products theProduct) {
        // save the customer using our service
        customerService.saveCustomer(theProduct);
        return "redirect:/products/list";
    }

    @GetMapping("/showFormForUpdate")
    public String showFormForUpdate(@RequestParam("productsId") int theId,
                                   Model theModel) {
        // get the customer from our service
        Products theCustomer = customerService.getCustomer(theId);
        // set customer as a model attribute to pre-populate the form
        theModel.addAttribute("products", theCustomer);
        // send over to our form
        return "products-form";
    }

    @GetMapping("/delete")
    public String deleteCustomer(@RequestParam("productsId") int theId) {
        // delete the customer
        customerService.deleteCustomer(theId);
        return "redirect:/products/list";
    }
}

```

- Dentro del paquete dao, tenemos.
- Una interface que lleva por nombre Producto Dao, al ser una interfaz con métodos abstractos, estos no llevan una lógica, ya que esta se desarrollará dentro de las clases que implementan dicha interface.



The screenshot shows a Java code editor with three tabs at the top: 'ProductsController.java', 'ProductsDAO.java', and '*ProductsDAOImpl.java'. The code in 'ProductsDAO.java' is as follows:

```
1 package com.luv2code.springdemo.dao;
2
3+import java.util.List;
4
5 public interface ProductsDAO {
6
7     public List<Products> getCustomers();
8
9     public void saveCustomer(Products theCustomer);
10
11    public Products getCustomer(int theId);
12
13    public void deleteCustomer(int theId);
14
15 }
16
17 }
```

-

- Despu s tenemos la clase ProductDAOImp, que es una clase que implementa a la interface antes descrita, obviamente hereda los m todos de la interface, dentro de estos se desarrolla la l gica de acuerdo al como se manejara la informaci n que se obtiene desde la BD.
- Como podemos ver en los m todos getCustomers() y deleteCustomer(), hacemos las consultas a la entidad y no a la base de datos, esto se puede notar ya que tanto la base de datos y la entidad tienen nombres diferentes.

```

17 // need to inject the session factory
18 @Autowired
19 private SessionFactory sessionFactory;
20
21 @Override
22 public List<Products> getCustomers() {
23     // get the current hibernate session
24     Session currentSession = sessionFactory.getCurrentSession();
25     // create a query ... sort by last name
26     Query<Products> theQuery =
27         currentSession.createQuery("from Products order by name",
28                                  Products.class);
29     System.out.println("theQuery = " + theQuery);
30     // execute query and get result list
31     List<Products> products = theQuery.getResultList();
32     // return the results
33     return products;
34 }
35
36 @Override
37 public void saveCustomer(Products theProduct) {
38     // get current hibernate session
39     Session currentSession = sessionFactory.getCurrentSession();
40     // save/update the customer ... finally LOL
41     currentSession.saveOrUpdate(theProduct);
42 }
43
44 @Override
45 public Products getCustomer(int theId) {
46     // get the current hibernate session
47     Session currentSession = sessionFactory.getCurrentSession();
48     // now retrieve/read from database using the primary key
49     Products theCustomer = currentSession.get(Products.class, theId);
50     return theCustomer;
51 }
52
53 @Override
54 public void deleteCustomer(int theId) {
55     // get the current hibernate session
56     Session currentSession = sessionFactory.getCurrentSession();
57     // delete object with primary key
58     Query theQuery =
59         currentSession.createQuery("delete from Products where id=:productsId");
60     theQuery.setParameter("productsId", theId);
61
62     theQuery.executeUpdate();

```

- Dentro del paquete entity, tenemos a la entidad, la cual es una clase que lleva por nombre Products, en esta clase se establece el nombre de la base de datos y se le establece un alias a la tabla que almacena la información por lo tanto esta será la entidad por la cual se llamará para manejar la información de la BD.
- Además esta clase tiene métodos de acceso para obtener y modificar los atributos de la tabla, a estos también se les especifica un alias, justo como se le hace a la entidad.

```
J Products.java X
12 @Entity
13 @Table(name = "products")
14 public class Products {
15
16     @Id
17     @GeneratedValue(strategy = GenerationType.IDENTITY)
18     @Column(name = "id")
19     private int id;
20
21     @Column(name = "name")
22     private String name;
23
24     @Column(name = "description")
25     private String description;
26
27     @Column(name = "content")
28     private String content;
29
30     @Column(name = "price")
31     private String price;
32
33     @Column(name = "date_register")
34     private Date dateRegister;
35
36     @Column(name = "date_expiration")
37     private Date dateExpiration;
38
39     public Products() {
40
41 }
42
43     public int getId() {
44         return id;
45     }
46
47     public void setId(int id) {
48         this.id = id;
49     }
50
51     public String getName() {
52         return name;
53     }
54
55     public void setName(String name) {
56         this.name = name;
57     }
58 }
```

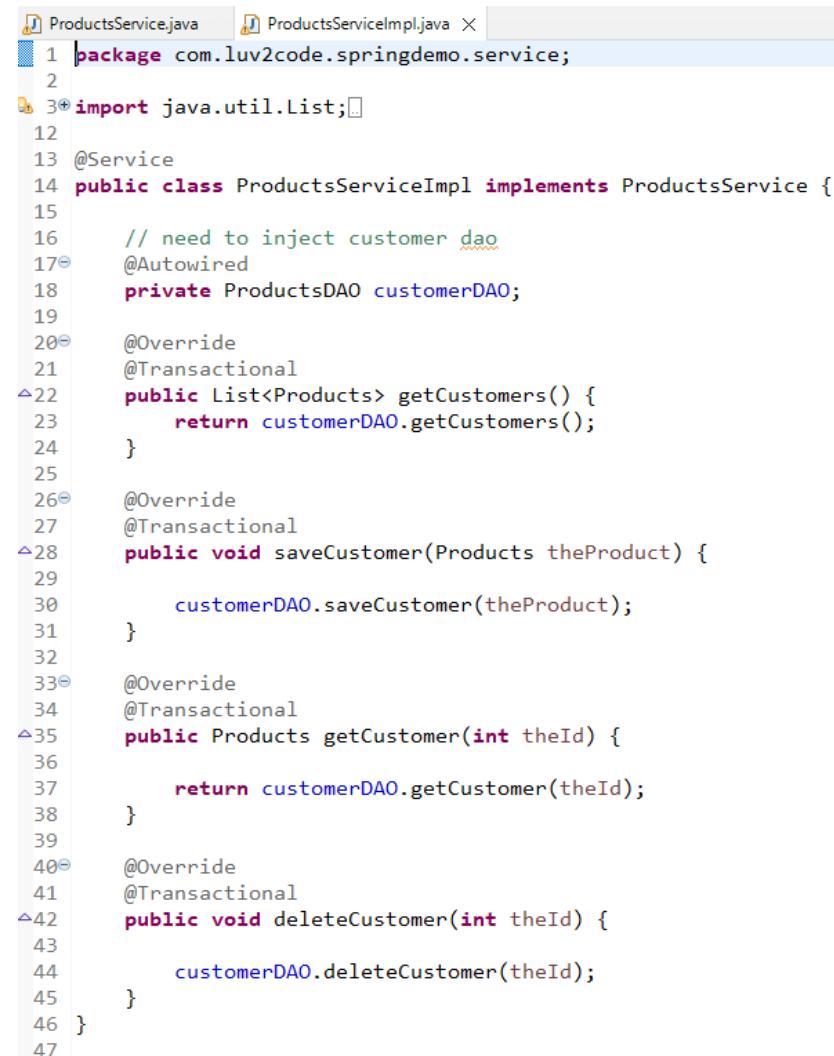
- En el paquete de service, tenemos una interface que contiene métodos abstractos que posteriormente serán usados en otra clase que se encuentra en el mismo paquete y lleva por nombre ProductServiceImp.

```

8  public interface ProductsService {
9
10     public List<Products> getCustomers();
11
12     public void saveCustomer(Products theProduct);
13
14     public Products getCustomer(int theId);
15
16     public void deleteCustomer(int theId);
17
18 }

```

- En la clase productServiceImp, implementamos la interface descrita en el punto anterior.
- Dentro de esta clase desarrollamos la logica, que internamente cada método invoca los métodos de la clase CustomerDAO, la cual invoca o llama a las acciones y vistas que permiten modificar y manejar la información.



```

1 package com.luv2code.springdemo.service;
2
3 import java.util.List;
4
5 @Service
6 public class ProductsServiceImpl implements ProductsService {
7
8     // need to inject customer dao
9     @Autowired
10    private ProductsDAO customerDAO;
11
12    @Override
13    @Transactional
14    public List<Products> getCustomers() {
15        return customerDAO.getCustomers();
16    }
17
18    @Override
19    @Transactional
20    public void saveCustomer(Products theProduct) {
21
22        customerDAO.saveCustomer(theProduct);
23    }
24
25    @Override
26    @Transactional
27    public Products getCustomer(int theId) {
28
29        return customerDAO.getCustomer(theId);
30    }
31
32    @Override
33    @Transactional
34    public void deleteCustomer(int theId) {
35
36        customerDAO.deleteCustomer(theId);
37    }
38
39
40    @Override
41    @Transactional
42    public void deleteCustomer(int theId) {
43
44        customerDAO.deleteCustomer(theId);
45    }
46}
47

```

- Pasando ahora las JSP, tenemos una que lleva por nombre list-product, esta jsp también fue refactorizar para aplicar un mejor estilo y mostrar información adicional y detallada de los productos.
- Cada producto se muestra en una fila de una tabla, la información se maneja mediante código HTML y Java.

```

10<body>
11    <div class="container">
12        <div class="container-title center">
13            <h2 class="title-page">Web Shop</h2>
14        </div>
15        <div class="container-content center">
16            <!-- put new button: Add Customer -->
17            <input class="btn-add" type="button" value="Add Product"
18                onclick="window.location.href='showFormForAdd'; return false;" 
19                class="add-button" />
20            <!-- add our html table here -->
21            <table>
22                <tr>
23                    <th>Name</th>
24                    <th>Description</th>
25                    <th>Content</th>
26                    <th>Price</th>
27                    <th>Date Register</th>
28                    <th>Date Expiratio</th>
29                    <th>Action</th>
30                </tr>
31                <!-- loop over and print our customers -->
32                <c:forEach var="tempProducts" items="${products}">
33                    <!-- construct an "update" link with customer id -->
34                    <c:url var="updateLink" value="/products/showFormForUpdate">
35                        <c:param name="productsId" value="${tempProducts.id}" />
36                    </c:url>
37                    <!-- construct an "delete" link with customer id -->
38                    <c:url var="deleteLink" value="/products/delete">
39                        <c:param name="productsId" value="${tempProducts.id}" />
40                    </c:url>
41                    <tr>
42                        <td>${tempProducts.name}</td>
43                        <td>${tempProducts.description}</td>
44                        <td>${tempProducts.content}</td>
45                        <td>${tempProducts.price}</td>
46                        <td>${tempProducts.dateRegister}</td>
47                        <td>${tempProducts.dateExpiration}</td>
48                        <td>
49                            <!-- display the update link --> <a class="btn-update" href="${updateLink}">Update</a>
50                            | <a class="btn-delete" href="${deleteLink}" 
51                                onclick="if (!confirm('Are you sure you want to delete this customer?')) return false">Delete</a>
52                        </td>
53                    </tr>
54                </c:forEach>
55            </table>

```

- La vista final de esta JSP es la siguiente.

Web Shop							
Name	Description	Content	Price	Date Register	Date Expiratio	Action	
Chetos	flaming hot	50g	\$5.00	2008-07-03	2008-07-03	Update Delete	
computadora	HP pavillion	1pza	\$17,000.00	2008-07-03	2008-07-03	Update Delete	
Doritos	Frituras	34g	15	2022-12-05	2022-12-11	Update Delete	
pantalon	sahara mezclilla	50g	\$5.00	2008-07-03	2008-07-03	Update Delete	
Sabritas	Frituras papas	35g,	12	2022-12-05	2022-12-12	Update Delete	
Sopa Maruchan Camaron	Sopa Instantanea sabor camaron	23gr	34	2022-12-11	2023-04-09	Update Delete	
Takis	Taquitos picantes con chile del que si pica xD	67gr.	15	2022-12-07	2023-01-20	Update Delete	
Zucaritas	Cerial xD	250gr.	50.0	2022-12-07	2022-12-21	Update Delete	

- Por último tenemos otra JSP , que lleva por nombre, “products-form”, dentro de esta se tiene un formulario, que permite la creación o modificación de un producto.

```

1 <%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
2 <!DOCTYPE html>
3 <html>
4 <head>
5   <title>Save Customer</title>
6   <link type="text/css" rel="stylesheet"
7     href="${pageContext.request.contextPath}/resources/css/style.css">
8
9   <link type="text/css" rel="stylesheet"
10    href="${pageContext.request.contextPath}/resources/css/add-customer-style.css">
11 </head>
12 <body>
13   <div class="container">
14     <div class="container-title">
15       <h2 class="title-page">Web Shop</h2>
16     </div>
17     <form:form class="form center" action="saveCustomer"
18       modelAttribute="products" method="POST">
19       <!-- need to associate this data with customer id -->
20       <form:hidden path="id" />
21       <h3 class="title-form">Save Product</h3>
22       <form:input path="name" placeholder="Name" />
23       <form:input path="description" placeholder="Description" />
24       <form:input path="content" placeholder="Content" />
25       <form:input path="price" placeholder="Price" />
26       <form:input path="dateRegister" placeholder="Date Register" />
27       <form:input path="dateExpiration" placeholder="Date Expiration" />
28       <input type="submit" value="Save" class="save" />
29     <div class="container-back center">
30       <a class="btn-back"
31         href="${pageContext.request.contextPath}/products/list">Back to
32         List</a>
33     </div>
34   </form:form>
35 </div>
36 </body>
37 </html>
```

- La vista final de esta JSP, sería la siguiente.

The screenshot shows a 'Save Product' form within a 'Web Shop' application. The background features a dark purple gradient with colorful diagonal stripes in yellow, cyan, and magenta. The form itself has a dark blue header with the title 'Save Product'. Below the title are six input fields, each containing a piece of product information: 'Chetos', 'flaming hot', '50g', '\$5.00', '2008-07-03', and '2008-07-03'. At the bottom of the form are two buttons: a blue 'Save' button and a red 'Back to List' button.

Field	Value
Name	Chetos
Description	flaming hot
Weight	50g
Price	\$5.00
Expiry Date	2008-07-03
Manufacture Date	2008-07-03

Save

Back to List