

PROYECTO DE FIN DE CICLO SUPERIOR DE DESARROLLO DE APLIACIONES WEB

TÉCNICO SUPERIOR EN DESARROLLO DE APLICACIONES WEB

Índice general

Capítulo 1.	Introducción y objetivos
Capítulo 2.	Especificación de Requisitos
Capítulo 3.	Planificación Temporal
Capítulo 4.	Evaluación de Costes
Capítulo 5.	Tecnologías Utilizadas
Capítulo 6.	Anotaciones al desarrollo e implementación
Capítulo 7.	Conclusiones y líneas futuras
Capítulo 8.	Bibliografía.

1. Introducción y objetivos

El trabajo de fin de ciclo que he realizado consiste en la creación de una aplicación web de ofertas de las diferentes cadenas de comida rápida que hay en España, ya que, estas tienen diferentes ofertas según el país en el que te encuentres.

Me decidí por hacer esta página web ya que he observado en internet que no hay ninguna dedicada a informar sobre las ofertas de comida rápida. Esto quizás sea debido a que las ofertas suelen estar en las aplicaciones o páginas web de la propia cadena de comida rápida, y ello conlleva a que el usuario tenga que estar cambiando o consultando continuamente diferentes paginas para poder visualizar las ofertas de cada cadena. Considero que es una página que puede llegar a ser muy visitada ya que toda la población tiene la necesidad básica de comer, y hay una alta demanda por parte de la población en estos restaurantes de comida rápida.

En dicha página se ofrece al usuario la posibilidad de registrarse o no. El usuario no registrado únicamente podrá visualizar las ofertas disponibles en las diferentes cadenas de comida rápida, mientras que los usuarios registrados podrán añadir ofertas de alguna cadena, entre las diferentes cadenas de comida rápida que da como opción la página y la posibilidad de eliminar dichas ofertas que han creado ellos mismos. También hay un usuario administrador que tiene un rol diferente y un mayor control en la página web, ya que tendrá la capacidad de eliminar cualquier oferta que hay en la página, es decir, que si no ha sido el creador de la oferta tiene la posibilidad de eliminar la oferta. La página ha sido creada utilizando los lenguajes de HTML, CSS3, SQL y Java. Dichos lenguajes se han impartido en el módulo formativo que he realizado y pretendo, con este proyecto, demostrar los conocimientos que tengo de ellos. También he utilizado Git y GitHub para hacer un repositorio del proyecto.

El nombre elegido para la página web es OnlyFood, pues pienso que es un nombre que aporta información sobre el contenido de la página, de esta forma resultaría fácil llamar la atención del usuario para que acceda a ella y la utilice.

El objetivo principal es ofrecer a los usuarios que accedan a ella información que les permita beneficiarse de las ofertas y un ahorro de dinero que no lograrían si hicieran una compra normal sin apoyo de la página.

2. Especificación de requisitos

El usuario va a necesitar para poder acceder y hacer uso de la página web, un dispositivo electrónico con acceso a internet y un navegador web para poder acceder a la página, como Google Chrome.

3. Planificación Temporal

En esta sección se especifica el tiempo que se ha invertido en la elaboración de cada parte del proyecto.

3.1. Búsqueda de información.

3.1.1. Búsqueda de ofertas de cadenas de comida rápida: 1 hora.

3.1.2. Descarga de imágenes en relación con las ofertas: 30 min.

3.2. Base de datos.

3.2.1. Elaboración del diagrama de la Base de datos con los datos necesarios para cada tabla, en la que se especifican sus claves primarias y foráneas y tipo de datos de cada tabla: 1 hora.

3.2.2. Creación de la base datos conforme se ha elaborado el diagrama de la base de datos: 30 minutos.

3.2.3. Insertar datos de ofertas de las cadenas de comida rápida: 1 hora;

3.3. Desarrollo de la página en entono Java.

3.3.1. Creación del proyecto Java con la faceta de JPA, conectar la base de datos con el proyecto Java, importación de las entidades mediante las tablas con las relaciones correspondientes: 30 min.

3.3.2. Desarrollo de la funcionabilidad de la página: 6 horas.

3.3.3. Corrección de errores de programación y pruebas de funcionamiento correcto: 7 horas.

3.4. Diseño web.

3.4.1. Planificación de la estructura de la página web en relación con el posicionamiento de sus componentes: 30min.

3.4.2. Creación del logo de OnlyFood y CSS de la página: 30 min.

3.4.3. Elaboración de las páginas web con el diseño planificado y asignarle la funcionalidad en relación con el desarrollo en Java: 6 horas.

3.5. Despliegue de la aplicación.

3.5.1. Creación del pool de seguridad en el server (Payara Server): 1 hora.

3.5.2. Despliegue de la aplicación: 10 minutos.

3.6. Git y GitHub.

3.6.1. Creación de la cuenta y repositorio: 15min.

3.6.2. Enlazar y subir los contenidos del proyecto: 15min.

4. Evaluación de Coste

4.1. Base de datos de Oracle: debido a que es una aplicación web con poco número de tablas, y las ofertas de las cadenas dejan de ser válidas (se eliminan) y aparecen nuevas. La aplicación utiliza una base de datos de Oracle de servicio gratuito con 10 CPU y 20 GB de almacenamiento.

4.2. Hosting para aplicaciones. JAVA: HOSTING JVM SMART 14,90€/mes. Este servicio incluye el dominio, Payara Server para el despliegue de la aplicación y la posibilidad de utilizar una base de datos MySQL si se quisiera, pero en este caso he optado para empezar con la base datos anteriormente mencionada.

4.3. Equipo informático para desarrollar la aplicación (ordenador): 600€.

4.4. Conexión a internet. Fibra 100Mb con la compañía finetwork: 20,90€/mes.

5. Tecnologías Utilizadas

La aplicación web OnlyFood tiene una base de datos Oracle que se ha creado, manejado y añadido datos mediante el programa SQLDeveloper. Para el proyecto he utilizado una base de datos utilizada durante el curso borrando los datos que contenía y añadiendo posteriormente los datos de la aplicación web.

Se ha desarrollado su funcionalidad en Java mediante el programa Eclipse creando un proyecto web dinámico añadiéndole la tecnología de JPA.

Su diseño está basado en Bootstrap, HTML y CSS3 para de esta forma obtener un diseño responsivo, es decir, adaptado a cualquier tipo de dispositivo, ya sea ordenador, tablet o móvil.

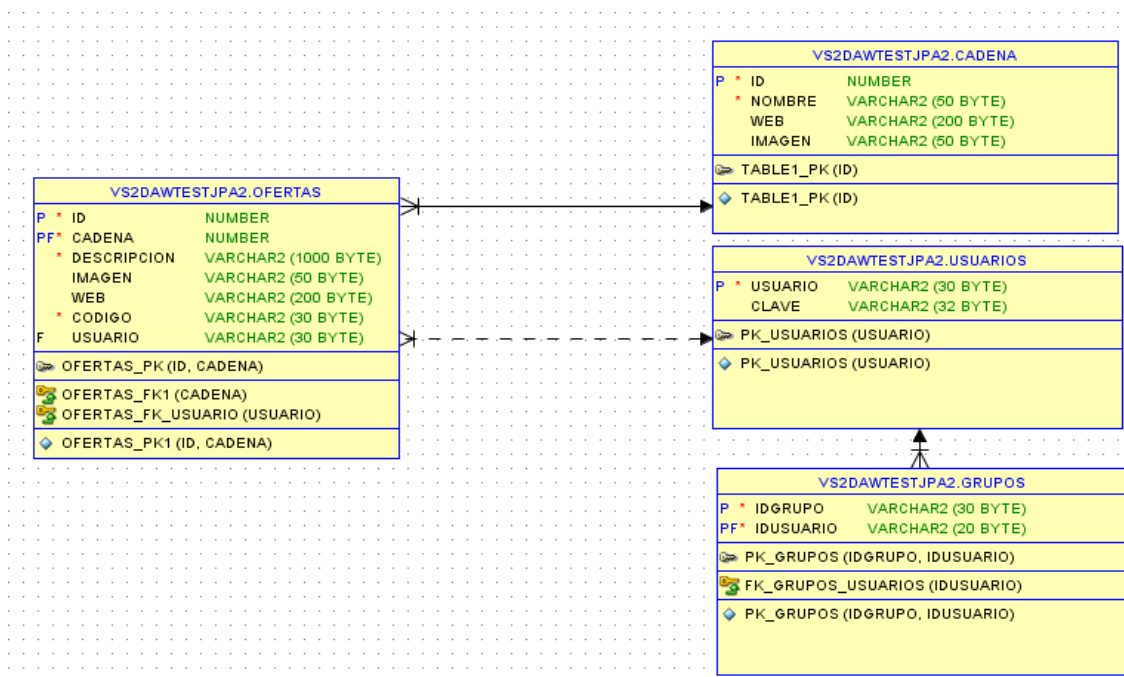
Para el despliegue de la aplicación he utilizado un Payara Server, en el cual se ha desplegado de forma local la aplicación.

Para poder hacer un desarrollo entre varias personas a la vez y tener una copia de seguridad del proyecto he utilizado la tecnología de Git y GitHub, en el que he creado un repositorio en el que se encuentra toda información del proyecto con la documentación incluida.

6. Anotaciones al desarrollo e implementación

6.1. Base de datos.

Diagrama de la base de datos: en él se puede ver el tipo de datos elegidos en la base de datos, las relaciones de las tablas, es decir, sus claves primarias y foráneas.



6.2. Desarrollo en Java.

Al iniciar la aplicación, la página de inicio es la página “index.jsp”, que redirige a la clase “Controller.java” (encargada de hacer la funcionalidad de los usuarios sin registrar) con una operación con el valor “listarCadenas” por parámetro. En ella, se llama al método “getAllCadenas()” de la clase “DaoCadena” que es el encargado de toda la funcionalidad de la entidad Cadena. Dicho método realiza una consulta en la base de datos que como resultado devuelve todas las cadenas que hay en la base de datos.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws !
// TODO Auto-generated method stub
HttpSession sesion=request.getSession();
String operacion=request.getParameter("operacion");
DaoCadena daoCadena=null;
Cadena cadena=null;
switch(operacion) {
    case "listarCadenas":
        try {
            daoCadena=new DaoCadena();
            List<Cadena> listadoCadenas= daoCadena.getAllCadenas();
            request.setAttribute("listadoCadenas", listadoCadenas);
            System.out.println("llega a listarCadenas");
            request.getRequestDispatcher("inicio.jsp").forward(request, response);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        break;
}
```

El método “getAllCadenas()” devuelve un listado de objetos “Cadena” (formada con las propiedades que tiene la tabla Cadena), que los obtiene de la consulta realizada. Posteriormente refresca el listado, debido a que JPA cachea las entidades, por lo tanto, si se hubiera hecho algún cambio en la base de datos desde la última vez que realizo la consulta mostraría el resultado antiguo, pero al realizar el “refresh” evitamos que nos surja ese problema.

```
11
12 public class DaoCadena extends BaseJPADao{
13
14     public DaoCadena() {
15         // TODO Auto-generated constructor stub
16     }
17     public List<Cadena> getAllCadenas(){
18         EntityManager em=getEntityManager();
19         List<Cadena> allCadenas;
20         Query allCadenaQuery =em.createQuery("select a from Cadena a ");
21         allCadenas = (List<Cadena>) allCadenaQuery.getResultList();
22         for(Cadena a:allCadenas)
23             em.refresh(a);
24         em.close();
25         //System.out.println(allCadenas.get(0).getNombre());
26         return allCadenas;
27     }
28 }
```

Se puede observar en la imagen, que esta clase está extendida de “BaseJPADao”. Esta clase es la encargada de crear un “EntityManager”, que a su vez es el encargado de manipular la base de datos en el caso que se lo solicitemos. Todas las clases del proyecto incluidas en la carpeta “dao” están extendidas con ella. En este proyecto la información para que se conecte a la base de datos se encuentra en el archivo de persistencia, se puede observar en la línea 19 de código como obtiene la información de dicho archivo.

```
1 package dao;
2
3 import javax.persistence.EntityManager;
4 import javax.persistence.EntityManagerFactory;
5
6 public class BaseJPADao {
7     /**
8      * Default no-arg constructor
9      */
10    public BaseJPADao() {
11    }
12
13    /**
14     * Returns JPA EntityManager reference.
15     * @return
16     */
17    public EntityManager getEntityManager() {
18        //return JPADaoFactory.createEntityManager();
19        EntityManagerFactory emf = PersistenceManager.getInstance().getEntityManagerFactory();
20        return emf.createEntityManager();
21    }
22 }
23
```

Una vez realizados estos pasos nos redirige a la página “inicio.jsp”, en la que se mostrarían solamente las cadenas en el listado de Cadenas que tengan alguna Oferta, en caso que no tengan no se mostraría la Cadena.

```
52< <c:forEach items="${listadoCadenas}" var="cadena">
53    <c:if test="${cadena.ofertas.size()>0}">
54        <div class="col-4 mb-4">
55            <a
56                href="${pageContext.request.contextPath}/controller?operacion=ofertasPorCadena&id=${cadena.id}">
57                
59            </a>
60        </div>
61    </c:if>
62
```

Al hacer click encima de la imagen de una Cadena nos redirigirá al controlador con dos parámetros, uno es la operación a realizar que en este caso es “ofertasPorCadena” y el otro es el id de la Cadena seleccionada.

En la operación mediante el método “seachCadenaById” devuelve la Cadena con el id introducido por parámetro y redirige a la página “ofertas.jsp”.

```
62         case "ofertasPorCadena":
63             try {
64                 daoCadena=new DaoCadena();
65                 String id=(String) request.getParameter("id");
66                 cadena= daoCadena.seachCadenabyId(id);
67                 request.setAttribute("cadena", cadena);
68                 //cadena.getOfertas().si
69                 System.out.println("llega a listarofertas");
70                 request.getRequestDispatcher("ofertas.jsp").forward(request, response);
71             } catch (Exception e) {
72                 // TODO Auto-generated catch block
73                 e.printStackTrace();
74             }
75             break;
```

El “seachCadenaById” es un método en el que se hace una conversión del parámetro introducido con formato String a long y con un “find” mediante el EntityManager devuelve el objeto “Cadena” con el id introducido por parámetro. Por último, se refresca la entidad antes de devolverla para tenerla actualizada.

```
34     public Cadena seachCadenabyId(String id) {
35         EntityManager em=getEntityManager();
36         System.out.println("el id es"+id);
37         long longId=Long.parseLong(id);
38         System.out.println("el longId es"+longId);
39         Cadena cadena=em.find(Cadena.class, longId);
40         em.refresh(cadena);
41         return cadena;
42     }
```

Una vez devueltos, el objeto “Cadena” con el id correspondiente, lo manda a la página “ofertas.jsp”, en ella se muestran todas las ofertas con el id de la Cadena. Se muestra la imagen asociada a la oferta, un código de oferta si tiene o en su defecto que consulte en el local, una descripción y un enlace a la página donde se ha obtenido la información de la oferta o en su defecto a la página oficial de la cadena de comida rápida.

```

54<
55<
56<
57<
58<
59<
60<
61<
62<
63<
64<
65<
66<
67<
68<
69<
70<
71<
72<
73<
74<
75<

```

```

<c:if test="${cadena.ofertas.size() >0}">
  <c:forEach items="${cadena.ofertas}" var="oferta">
    <a> </a>
    <div class="row col-12">
      <div class="col-2">
        
      </div>
      <div class="col-2">
        <h4>${oferta.codigo}</h4>
      </div>
      <div class="col-7">
        <p>${oferta.descripcion}</p>
      </div>
      <div class="col-1">
        <a href="${oferta.web}" target="_blank">
          <button class="btn btn-danger">Enlace Oferta</button>
        </a>
      </div>
    </div>
  </c:forEach>
</c:if>

```

En el caso que algún usuario quiera acceder a las ofertas de la cadena mediante la URL y cambie el id de la Cadena, y dicha Cadena no tenga ninguna oferta le mostrará un mensaje de que no hay ofertas de la Cadena.

```

51<
52<
53<

```

```

<c:if test="${cadena.ofertas.size() ==0}">
  <h4>No hay ofertas de ${cadena.nombre}</h4>
</c:if>

```

En todas las páginas de la aplicación hay una cabecera que su función es que nos mande al “index.jsp”, que a su vez nos lleva al controlador a la operación de “listarCadenas” para que nos redirija a la página principal que es “inicio.jsp”.

```

13<
14<
15<
16<
17<
18<
19<

```

```

<div class="row cabecera">
  <div class="col-10">
    <a href="${pageContext.request.contextPath}/index.jsp"> 
    </a>
  </div>

```

Dicha cabecera también contiene una imagen o un desplegable de opciones de usuario. En el caso de la imagen de login, haciendo click sobre ella te redirige a la zona de la página que solo podrán tener acceso los usuarios registrados. Concretamente te envía al “ControllerUsuario.java”, encargado de realizar todas las operaciones de los usuarios registrados, la operación que envía por parámetro es “listarOfertasUsuario”. La otra opción que aparezca en este apartado es el desplegable de las diferentes funciones que pueden ejecutar los usuarios registrados. Estas funciones son que le muestren las

ofertas del propio usuario, que inserten una nueva oferta o hacer un logout, es decir, cerrar la sesión de la cuenta de usuario que está en ese momento.

```
21<:if test="${pageContext.request.remoteUser==null}">
22<
23    <a
24        href="${pageContext.request.contextPath}/controllerUsuario?operacion=listarOfertasUsuario">
25        
26    </a>
27</c:if>
28<:if test="${pageContext.request.remoteUser!=null}">
29<div class="btn-group">
30    
33    <div class="dropdown-menu dropdown-menu-right">
34        <a class="dropdown-item"
35            href="${pageContext.request.contextPath}/controllerUsuario?operacion=listarOfertasUsuario">Mis
36            ofertas</a> <a class="dropdown-item"
37            href="${pageContext.request.contextPath}/controllerUsuario?operacion=insertarNuevaOferta">Nueva
38            Oferta</a>
39        <div class="dropdown-divider"></div>
40        <a class="dropdown-item"
41            href="${pageContext.request.contextPath}/controllerUsuario?operacion=logout">Cerrar
42            sesión</a>
43    </div>
44</div>
</c:if>
```

Una vez hemos accedido con nuestro usuario como hemos comentado en el último párrafo, no envía al controlador del usuario a la operación “listarOfertasUsuario”. En ella, recogemos de email del usuario que esta logueado en ese momento para mandarlo por parámetro al método “encontrarUsuario” de la clase “DaoUsuario” (clase que maneja las operaciones de la entidad Usuario). Dicho método se encarga de realizar un find y devolver el objeto de la entidad “Usuario” completo, tal y como hemos visto anteriormente en otros métodos parecidos de otras entidades.

A continuación, se comprueba si el usuario tiene un rol de usuario (usuario básico) o administrador. En el caso que sea un usuario básico devolverá un listado de las ofertas que ha insertado dicho usuario en la aplicación. Por el contrario, si es un usuario con rol de administrador se invoca el método “getAllofertas” de la clase “DaoOferta” que realiza una consulta que devuelve todas las ofertas que se encuentran en la base de datos.

Por último, manda el listado de las ofertas totales o del usuario a la página “ofertasCreadas.jsp”.

```

48     switch(operacion) {
49         case "listarOfertasUsuario":
50             try {
51                 DaoUsuario daoUsuario=new DaoUsuario();
52                 String usuario= request.getUserPrincipal().toString();
53                 Usuario user=daoUsuario.encontrarUsuario(usuario);
54                 request.setAttribute("usuario", user);
55                 boolean isAdmin=request.isUserInRole("administrador");
56                 List<Oferta> listadoOfertas;
57                 if(isAdmin==true) {
58                     DaoOferta daoOferta=new DaoOferta();
59                     listadoOfertas= daoOferta.getAllOfertas();
60                 }else
61                     listadoOfertas=user.getOfertas();
62                 System.out.println("el tamaño del listadoofertasusuario es "+listadoOfertas.size());
63                 request.setAttribute("listadoOfertas", listadoOfertas);
64                 System.out.println("llega a listarOfertasUsuario");
65                 request.getRequestDispatcher("/usuario/ofertasCreadas.jsp").forward(request, response);
66             } catch (Exception e) {
67                 // TODO Auto-generated catch block
68                 e.printStackTrace();
69             }
70             break;

```

En la página “ofertasCreadas.jsp” en el caso que el usuario no haya creado ninguna oferta o haya eliminado las que haya tenido con anterioridad se muestra un mensaje de que no hay ofertas del usuario.

```

77<c:if test="${listadoOfertas.size() ==0}">
78    <h4>No hay ofertas por ti </h4>
79</c:if>

```

En el caso que el usuario si tenga ofertas creadas por él o sea un usuario con rol de administrador, le saldrá el listado total de las ofertas, se mostrará la imagen de la oferta, el código, la descripción, un botón con un enlace a la web y una imagen con un enlace que redirija al controlador del usuario con tres parámetros: uno de la operación a realizar que es “eliminarOferta”, otro con el id de la oferta y el último el id de la cadena que está asociada a esta oferta.

```

80<c:if test="${listadoOfertas.size() >0}">
81    <c:forEach items="${listadoOfertas}" var="oferta">
82        <a> </a>
83        <div class="row col-12">
84            <div class="col-2">
85                
87            </div>
88            <div class="col-2">
89                <h4>${oferta.codigo}</h4>
90            </div>
91            <div class="col-6">
92                <p>${oferta.descripcion}</p>
93            </div>
94            <div class="col-1">
95                <a href="${oferta.web}" target=" blank">
96                    <button class="btn btn-danger">Enlace Oferta</button>
97                </a>
98            </div>
99            <div class="col-1">
100                <a
101                    href="${pageContext.request.contextPath}/controllerUsuario?operacion=eliminarOferta&idoferta=${ofert
102                    
103                </a>

```

Una vez dentro de la operación “eliminarOferta” se vuelve a realizar el proceso de encontrar el usuario, se recogen los parámetros del id de la oferta y del id de la cadena. Con ellos se ejecuta al método “eliminarOferta” de la clase “DaoOferta”.

```
15     case "eliminarOferta":
16         try {
17             System.out.println("entra en eliminar oferta");
18             DaoUsuario daoUsuario=new DaoUsuario();
19             //request.isUserInRole("usuario");
20             String usuario= request.getUserPrincipal().toString();
21             Usuario user=daoUsuario.encontrarUsuario(usuario);|
22             String idcadena=request.getParameter("cadena");
23             String idoferta=request.getParameter("idoferta");
24             DaoOferta daoOferta=new DaoOferta();
25             daoOferta.eliminarOferta(idcadena, idoferta);
26             user=daoUsuario.refrescarUsuario(user);
27             request.setAttribute("usuario", user);
28             System.out.println("llega a final eliminar oferta");
29             request.setAttribute("confirmacion", "Oferta eliminada");
30             request.getRequestDispatcher("/controllerUsuario?operacion=listarOfertasUsuario").forward(request, response);
31         } catch (Exception e) {
32             // TODO Auto-generated catch block
33             e.printStackTrace();
34         }
35     break;
```

En este método se invoca al EntityManager, se convierten los parámetros de id de la oferta y de id de la cadena de formato String a long, para que se pueda realizar el find de la entidad “OfertaPK” que es la entidad encargada para buscar la oferta a borrar, ya que esta entidad es la encargada de ser la clave primaria de la entidad “Oferta” y al ser una clave primaria compuesta por dos datos JPA forma una entidad aparte para que luego sea la entidad por la que va a realizar la búsqueda de la entidad “Oferta”. Una vez obtenemos el objeto de la oferta mediante el find de la entidad “Oferta”, podemos pasar a borrarla mediante el método “remove” del EntityManager. Por último, se hace una confirmación de los cambios realizados en la base de datos mediante un “commit” y se refrescan los objetos manejados para que se actualicen los datos que estaban cacheados.

```
59     public void eliminarOferta(String idcadena, String idoferta) {
60         // TODO Auto-generated method stub
61         System.out.println("entra en eliminarOferta");
62         EntityManager em= getEntityManager();
63         EntityTransaction tx = em.getTransaction();
64         try {
65             Oferta o=new Oferta();
66             Long id=Long.parseLong(idoferta);
67             Long cadena=Long.parseLong(idcadena);
68             OfertaPK oPK=new OfertaPK();
69             oPK.setCadena(cadena);
70             oPK.setId(id);
71             o=em.find(Oferta.class,oPK);
72             Usuario u=em.find(Usuario.class, o.getUsuarioBean().getUsuario());|
73             Cadena c=em.find(Cadena.class, cadena);
74             tx.begin();
75             System.out.println("antes de remove eliminarOferta");
76             em.remove(o);
77             tx.commit();
78             em.refresh(c);
79             em.refresh(u);
80             em.close();
81             System.out.println("commit eliminarOferta");
82         } catch (OptimisticLockException e) {
83             System.out.println("entra en OptimisticLockException eliminarOferta metodo");
84             em.close();
85         }
```

Una vez realizada la eliminación procedemos a refrescar el usuario mediante el método “refrescarUsuario” de la clase “DaoUsuario”. La acción que desempeña este método es el encontrar el usuario mediante un find y a su vez hacerle un refresh para actualizar sus datos y devolverlo.

```
74     public Usuario refrescarUsuario(Usuario user) {  
75         EntityManager em= getEntityManager();  
76         user=em.find(Usuario.class,user.getUsuario());  
77         em.refresh(user);  
78         em.close();  
79         return user;  
80     }
```

Una vez realizados todos los pasos se envía un atributo de confirmación con un mensaje de confirmación. Esta operación finaliza redireccionando a la operación “listarOfertasUsuario” del mismo controlador, para que muestre las ofertas actualizadas en “ofertasCreadas.jsp”.

```
<c:if test="${requestScope.confirmacion != null}">  
    <div id="divconfirmacion"  
        class="centrarTexto mx-auto col-7 mt-2 mb-2">  
        <p>  
            <strong><c:out value="confirmacion" /></strong> <br>  
            <c:out value="${requestScope.confirmacion}" />  
        </p>  
    </div>  
</c:if>  
<c:if test="${requestScope.error != null}">  
    <div id="diverror" class="centrarTexto mx-auto col-7 mt-2 mb-2">  
    <p>  
        <strong><c:out value="Error" /></strong> <br>  
        <c:out value="${requestScope.error}" />  
    </p>  
    </div>  
</c:if>
```

Si hubiera algún error en algún proceso se mostraría en este apartado de las vistas (en este caso es la de la página “ofertasCreadas.jsp”).

En la página “ofertasCreadas.jsp” hay un botón que llama a la operación “irNuevaOferta” del controlador de usuario. También se puede llamar a esta función mediante el desplegable de funciones del usuario en la cabecera.

```
73< a class="btn btn-danger mt-3 mb-3"  
74     href="${pageContext.request.contextPath}/controllerUsuario?operacion=irNuevaOferta">Nueva  
75     oferta</a>
```

En la operación “irNuevaOferta” se llama al método “getAllCadenas” de la clase “DaoCadena”, que se encarga de devolver un listado de todas las Cadenas que hay en la base de datos mediante una consulta. Por último, nos redirige la operación a la página “nuevaOferta.jsp”.

```
76 case "irNuevaOferta":
77     try {
78         daoCadena=new DaoCadena();
79         List<Cadena> listadoCadenas= daoCadena.getAllCadenas();
80         request.setAttribute("listadoCadenas", listadoCadenas);
81         System.out.println("llega a listarcadeanas");
82         request.getRequestDispatcher("/usuario/nuevaOferta.jsp").forward(request, response);
83     } catch (Exception e) {
84         // TODO Auto-generated catch block
85         e.printStackTrace();
86     }
87     break;

17 public List<Cadena> getAllCadenas(){
18     EntityManager em=getEntityManager();
19     List<Cadena> allCadenas;
20     Query allCadenaQuery =em.createQuery("select a from Cadena a ");
21     allCadenas = (List<Cadena>) allCadenaQuery.getResultList();
22     for(Cadena a:allCadenas)
23         em.refresh(a);
24     em.close();
25     //System.out.println(allCadenas.get(0).getNombre());
26     return allCadenas;
27 }
```

En la página “nuevaOferta.jsp” muestra un formulario para insertar una nueva oferta. El formulario pide los datos que forman la entidad Oferta excepto el id. La cadena de comida a la que le quieres añadir la oferta, se mostrara su nombre por un desplegable select, que contendrá el listado que hemos recibido de la operación anterior y dependiendo de la cadena seleccionada cambiara el valor del desplegable select. El campo de descripción es un textarea debido a que posee una mayor capacidad de introducir información en el caso que lo de desee el usuario en su oferta a crear. El resto de campos son input de tipo texto.

```
52< form action="${pageContext.request.contextPath}/controllerUsuario"
53    method="POST" class="col-12">
54    <div class="form-group">
55        <label for="cadena">Cadena de comida:</label> <select
56            name="cadena" class="form-control" id="cadena"
57            value="${listadoCadenas[0].id}" required>
58            <c:forEach items="${listadoCadenas}" var="cadena">
59                <option value="${cadena.id}">${cadena.nombre}</option>
60            </c:forEach>
61            <option></option>
62        </select>
63    </div>
64    <div class="form-group">
65        <label for="codigo">Código oferta:</label> <input type="text"
66            class="form-control" id="codigo" name="codigo"
67            placeholder="Introduce el código de oferta">
68    </div>
```

Como se puede observar en la imagen el formulario tiene un atributo action que se encarga realizar la acción que hay contenida en él, en este caso es redirigir al controlador del usuario con la operación “insertarOferta” que se encuentra en un input hidden (que no se muestra para el usuario). Esta redirección se producirá una vez se haga click en el botón de tipo submit del formulario.

También posee un enlace con apariencia de botón con la URL que te envía al controlador de usuario con la operación “listarOfertasUsuario” para que posteriormente nos muestre la página “ofertasCreadas.jsp” como hemos visto anteriormente.

```
92         <input type="hidden" name="operacion" id="operacion"
93             value="insertarOferta" />
94         <button type="submit" name="submit" class="btn btn-primary">Aceptar</button>
95         <a class="btn btn-primary"
96             href="{pageContext.request.contextPath}/controllerUsuario?operacion=ListarOfertasUsuario">Cancelar</a>
97     </form>
98 />
```

Una vez se llega a la operación “insertarOferta” del controlador del usuario, se llama al método “encontrarUsuario” para manejar el usuario que esta logueado. Después recogemos los valores enviados con el formulario. En el caso que el código de la oferta no se haya insertado ninguno se la da por defecto “Consulta en el local.”. Los valores recogidos se le asignan a un objeto de la entidad “Oferta”. En el caso del id de la cadena recogido procederemos a llamar al método “seachCadenaById” de la clase “DaoCadena” que realizará un find del objeto mediante en EntityManager y nos lo devolverá una vez encontrado. Seguidamente, se procederá a llamar al método “nuevaOferta” de la clase “DaoOferta”.

```
90     case "insertarOferta":
91         try {
92             System.out.println("entra en nueva oferta");
93             DaoUsuario daoUsuario=new DaoUsuario();
94             //request.isUserInRole("usuario");
95             String usuario= request.getUserPrincipal().toString();
96             Usuario user=daoUsuario.encontrarUsuario(usuario);
97             Oferta oferta=new Oferta();
98             System.out.println("-----"+request.getParameter("codigo"));
99             if(request.getParameter("codigo").equalsIgnoreCase("")==false || request.getParameter("codigo")==null)
100                 oferta.setCodigo(request.getParameter("codigo"));
101             else
102                 oferta.setCodigo("Consultar en local.");
103             //oferta.setCodigo(request.getParameter("codigo"));
104             oferta.setDescripcion(request.getParameter("descripcion"));
105             oferta.setWeb(request.getParameter("web"));
106             oferta.setUsuarioBean(user);
107             cadena=new Cadena();
108             daoCadena=new DaoCadena();
109             System.out.println("cadena"+request.getParameter("cadena"));
110             cadena=daoCadena.seachCadenabyId(request.getParameter("cadena"));
```



```

111      /*
112      * if(request.getParameter("imagen")=="" ||
113      * request.getParameter("imagen")==null)
114      * oferta.setImagen(request.getParameter("imagen")); else
115      */
116      oferta.setImagen(cadena.getImagen());
117      DaoOferta daoOferta=new DaoOferta();
118      daoOferta.nuevaOferta(oferta, cadena);
119      user=daoUsuario.refrescarUsuario(user);
120      request.setAttribute("usuario", user);
121      request.setAttribute("confirmacion", "Oferta añadida correctamente");
122      System.out.println("llega a final nueva oferta");
123      request.getRequestDispatcher("/controllerUsuario?operacion=listarOfertasUsuario").forward(request,
124  ) catch (Exception e) {
125      // TODO Auto-generated catch block
126      e.printStackTrace();
127  }
128  break;

```

El método “nuevaOferta” recibe dos parámetros: un objeto de la entidad “Oferta” y otro de “Cadena”. Seguidamente creamos un objeto de la entidad “OfertaPK” (funcionalidad de esta entidad explicada en la página 13), que se la añadiremos a nuestro objeto “Oferta” para poder insertarla en la base de datos mediante un método “persist” que su función es insertar el objeto que se le introduce por parámetro en la base de datos y para que estos cambios sean definitivos se realiza un “commit” (realizar una confirmación en la base de datos). Antes de salir del método se refrescan los objetos “Cadena”.

```

23  public void nuevaOferta(Oferta o,Cadena cadena) {
24      System.out.println("entra en nuevaOferta");
25      EntityManager em= getEntityManager();
26      EntityTransaction tx = em.getTransaction();
27      try {
28          Long idOferta=(long) cadena.getOfertas().size()+1;
29          OfertaPK oPK=new OfertaPK();
30          oPK.setCadena(cadena.getId());
31          oPK.setId(idOferta);
32          o.setId(oPK);
33          o.setCadenaBean(cadena);
34          //Usuario u=em.find(Usuario.class, o.getUsuarioBean().getUsuario());
35          Cadena c=em.find(Cadena.class, cadena.getId());
36          tx.begin();
37          em.persist(o);
38          System.out.println("antes de persistir oferta nueva");
39          tx.commit();
40          em.refresh(c);
41          //em.refresh(u);
42          //em.refresh(o);
43          //em.refresh(o.getUsuarioBean());
44          em.close();
45          System.out.println("commit nuevaOferta");
46      } catch (OptimisticLockException e) {
47          System.out.println("entra en OptimisticLockException nuevaOferta metodo");
48          em.close();
49          throw e;
50      } catch (RollbackException e) {
51          System.out.println("entra en RollbackException nuevaOferta metodo");
52          if (tx.isActive())
53              tx.rollback();
54          em.close();
55          throw e;
56      }
57  }

```

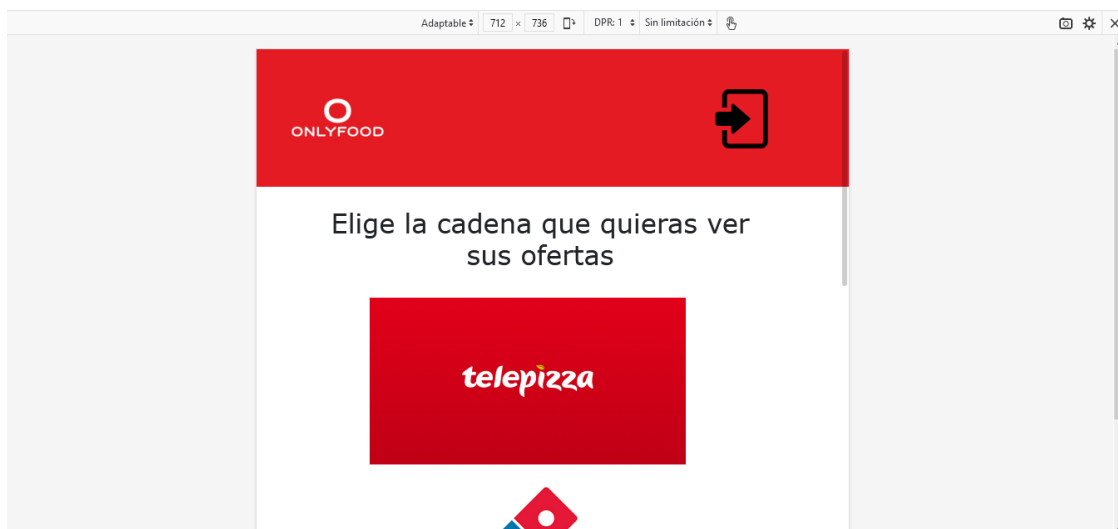
Por último, en la operación “insertarOferta” del controlador usuario cuando volvemos de añadir la oferta en la base de datos llamamos al método de refrescar el usuario logueado para que se actualice el objeto con los cambios realizados y redirigimos a la

operación del controlador del usuario que a su vez redirige a “ofertasCreadas.jsp” donde podremos visualizar la oferta añadida y su mensaje de confirmación.

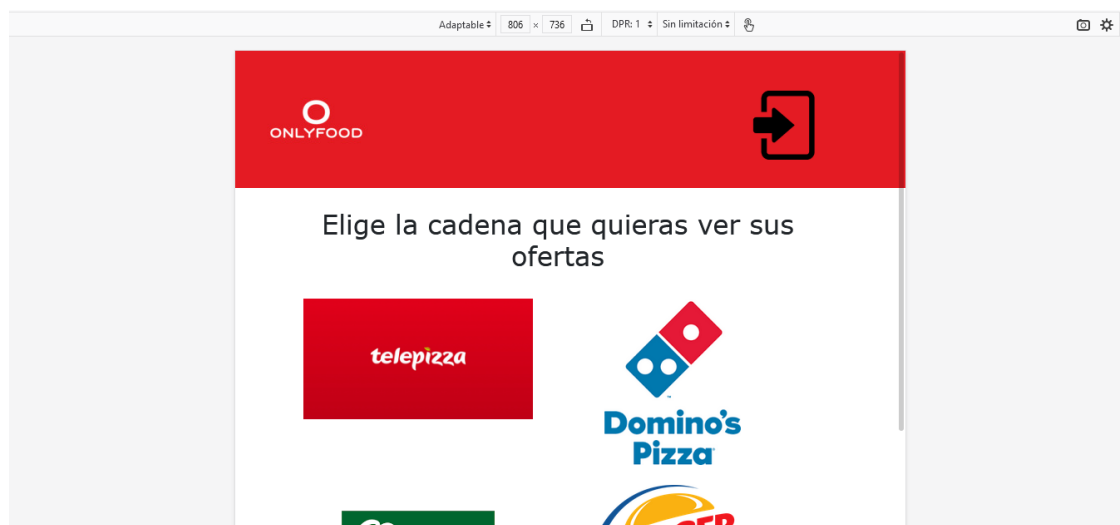
6.3. Diseño web con Bootstrap.

El diseño de la aplicación web es un diseño responsivo, es decir, la colocación de los elementos de cada página se adapta al tamaño de pantalla desde el que se está visualizando la aplicación web.

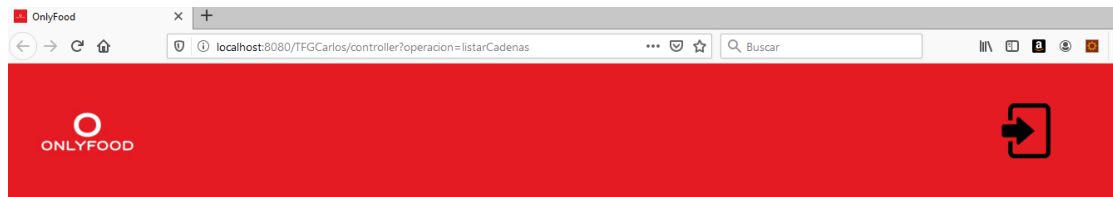
En este caso la colocación de los elementos de la página “inicio.jsp” que estamos visualizando está adaptado a pantallas pequeñas como las de los teléfonos móviles.



En este caso está adaptada la misma página a tamaños de pantallas de tamaño intermedio o medianas como las de las tablets.



En el caso de las pantallas de tamaño grande como las de los ordenadores se visualizaría el siguiente diseño.

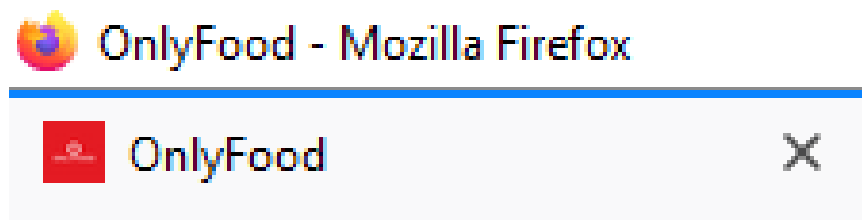


Elige la cadena que quieras ver sus ofertas



Todas estas modalidades tienen las mismas funciones, solo cambiaría la colocación y tamaños de los elementos de la página.

En la pestaña de nuestro explorador se puede apreciar que aparece el nombre de "OnlyFood" (el de la aplicación web) y el logo de la aplicación.



Que aparezca el nombre se consigue solamente con añadir a las páginas en el parte del diseño de "head" la etiqueta de "title".

```
6 <head>
7 <meta charset="UTF-8">
8 <title>OnlyFood</title>
```

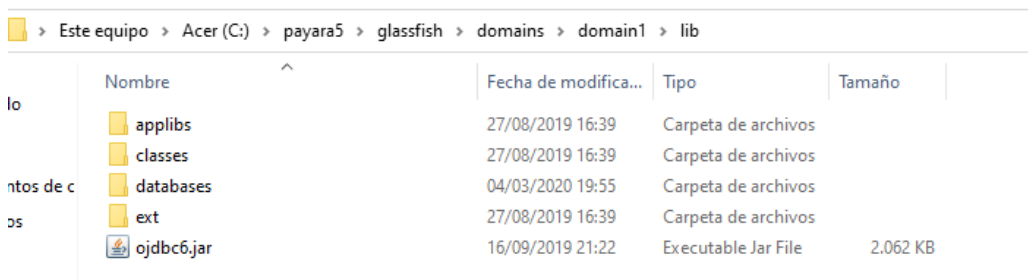
Y el icono del logo se añade mediante la siguiente línea de código, en la que importa la imagen de la carpeta de imágenes que contiene el proyecto.

```
1 <link rel="shortcut icon" type="image/x-icon"
2 href="${pageContext.request.contextPath}/resources/img/logo_only_r.jpg" />
```

6.4. Seguridad.

La aplicación posee un dominio de seguridad con el fin de los usuarios que accedan a la aplicación sin estar registrados no puedan tener acceso mediante la URL a los apartados que únicamente tienen acceso los usuarios que están registrados.

Para poder llevar a cabo este proceso tiene que disponer el Payara Server de un driver que se situara en la ubicación que muestra en la imagen para que pueda conectar futuramente con la base de datos.



Nombre	Fecha de modifica...	Tipo	Tamaño
applibs	27/08/2019 16:39	Carpeta de archivos	
classes	27/08/2019 16:39	Carpeta de archivos	
databases	04/03/2020 19:55	Carpeta de archivos	
ext	27/08/2019 16:39	Carpeta de archivos	
ojdbc6.jar	16/09/2019 21:22	Executable Jar File	2.062 KB

En nuestra base de datos disponemos de dos tablas:

- Usuarios: que almacena a información del usuario.
- Grupos: que almacena el rol que tiene que cada usuario.

Para aportarle una mayor seguridad la contraseña esta encriptada con la encriptación “MD5”. En el proyecto hay una clase para poder encriptar un dato de tipo String en “MD5”, esta clase se encuentra dentro del paquete “útil” y se llama “Hash.java”.

```
10 public static String getHash(String txt, String hashType) {
11     try {
12         java.security.MessageDigest md = java.security.MessageDigest.getInstance(hashType);
13         byte[] array = md.digest(txt.getBytes());
14         StringBuffer sb = new StringBuffer();
15         for (int i = 0; i < array.length; ++i) {
16             sb.append(Integer.toHexString((array[i] & 0xFF) | 0x100).substring(1,3));
17         }
18         return sb.toString();
19     } catch (java.security.NoSuchAlgorithmException e) {
20         //error action
21     }
22     return null;
23 }
24
25 public static String md5(String txt) {
26     return Hash.getHash(txt, "MD5");
27 }
28
```

El server debe tener creado un pool de conexiones para que le permita acceder al base de datos donde se alojan los usuarios y grupos. Debe tener las siguientes propiedades correspondientes de la base de datos.

The screenshot shows the 'onlyfoodpool' configuration page. The 'Pool Name' is 'onlyfoodpool'. Below the 'Additional Properties (5)' section, there is a table with the following data:

Select	Name	Value	Description
<input type="checkbox"/>	Password	VS2DAWTESTJPA2	
<input type="checkbox"/>	DataSourceName	TFGCarlos	
<input type="checkbox"/>	URL	jdbc:oracle:thin:@80.28.158.14:1521:oradai	
<input type="checkbox"/>	User	VS2DAWTESTJPA2	
<input type="checkbox"/>	PortNumber	1521	

A ese pool se le debe asociar un recurso JDBC.

The screenshot shows the 'jdbc/onlyfood' configuration page. The 'JNDI Name' is 'jdbc/onlyfood'. The 'Pool Name' is 'onlyfoodpool'. The 'Deployment Order' is '100'. The 'Status' is 'Enabled'. Below the 'Additional Properties (0)' section, there is a table with the following data:

Select	Name	Value	Description
--------	------	-------	-------------

Este recurso JDBC tiene asociado el dominio de seguridad en el que se especifican los datos de dominio con los de la tabla de la base de datos.

The screenshot shows the 'onlyfoodjdbc' configuration page. The 'JNDI' is 'jdbc/onlyfood'. The 'User Table' is 'USUARIOS'. The 'User Name Column' is 'USUARIO'. The 'Password Column' is 'CLAVE'. The 'Group Table' is 'GRUPOS'. The 'Group Table User Name Column' is 'IDUSUARIO'.

Property	Value	Description
JNDI *	jdbc/onlyfood	JNDI name of the JDBC resource used by this realm
User Table *	USUARIOS	Name of the database table that contains the list of authorized users for this realm
User Name Column *	USUARIO	Name of the column in the user table that contains the list of user names
Password Column *	CLAVE	Name of the column in the user table that contains the user passwords
Group Table *	GRUPOS	Name of the database table that contains the list of groups for this realm
Group Table User Name Column *	IDUSUARIO	Name of the column in the user group table that contains the list of groups for this realm

The screenshot shows the 'Security' configuration page in GlassFish. The left sidebar lists various configuration areas, with 'Security' expanded and 'onlyfoodjdbc' selected. The main panel shows the configuration for the 'onlyfoodjdbc' realm. The 'Group Name Column' is set to 'IDGRUPO'. The 'Assign Groups' field is empty. The 'Database User' and 'Database Password' fields are empty. The 'Digest Algorithm' is set to 'MD5'. The 'Encoding' field is empty.

Group Name Column: *	IDGRUPO
Assign Groups:	
Database User:	
Database Password:	
Digest Algorithm:	MD5
Encoding:	

Con esto el dominio de seguridad de nuestro server está configurado, pero falta configurar en el entorno java. En el archivo “web.xml” está definido el método de autenticación de los usuarios que se conecten a la aplicación.

```

4< welcome-file-list>
5  <welcome-file>index.jsp</welcome-file>
6< /welcome-file-list>
7< login-config>
8  <auth-method>FORM</auth-method>
9  <realm-name>onlyfoodjdbc</realm-name>
10< form-login-config>
11  <form-login-page>/seguridad/identificate.jsp</form-login-page>
12  <form-error-page>/seguridad/credencialesnovalidas.jsp</form-error-page>
13< /form-login-config>
14< /login-config>
15< security-role>
16  <description>Rol de usuarios registrados para la aplicacion</description>
17  <role-name>usuario</role-name>
18< /security-role>
19< security-role>
20  <description>Rol de administrativos para la aplicacion</description>
21  <role-name>administrador</role-name>
22< /security-role>
23< security-constraint>
24  <display-name>Consulta Usuario</display-name>
25  <web-resource-collection>
26    <web-resource-name>Consulta Usuario</web-resource-name>
27    <description></description>
28    <url-pattern>/usuario/*</url-pattern>
29  </web-resource-collection>
30  <auth-constraint>
31    <description></description>
32    <role-name>usuario</role-name>
33    <role-name>administrador</role-name>
34  </auth-constraint>
35< /security-constraint>
36< security-constraint>
37  <display-name>Acceso al controlador de operaciones de Usuario</display-name>
38  <web-resource-collection>
39    <web-resource-name>Acceso al controlador de operaciones de Usuario</web-resource-name>
40    <description></description>
41    <url-pattern>/controllerusuario</url-pattern>
42    <url-pattern>/controllerUsuario</url-pattern>
43  </web-resource-collection>
44  <auth-constraint>
45    <description></description>
46    <role-name>usuario</role-name>
47    <role-name>administrador</role-name>

```

El servidor GlassFish dispone de usuarios autenticados y grupos de usuarios autenticados. Están asignados los roles que tienen los usuarios de nuestra aplicación en el archivo “glassfish-web.xml”.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!DOCTYPE glassfish-web-app PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1 S
4 <glassfish-web-app>
5   <context-root>/TFGCarlos</context-root>
6   <security-role-mapping>
7     <role-name>usuario</role-name>
8     <group-name>usuario</group-name>
9   </security-role-mapping>
10  <security-role-mapping>
11    <role-name>administrador</role-name>
12    <group-name>administrador</group-name>
13  </security-role-mapping>
14  <parameter-encoding default-charset="UTF-8"/>
15 </glassfish-web-app>
```

Esta es la página que te muestra el servidor de dominio si un usuario sin estar autenticado intenta acceder a una página o función que no tiene acceso. Esta página es “identificate.jsp”, que es un formulario para loguearse en la aplicación y de esta forma conseguir el acceso a esas funciones.

```
21 <div class="row">
22   <div class="col-1"></div>
23   <div class="row contenido col-10 centrarTexto">
24     <div class="col-12 mt-4 mb-4">
25       <h2>Inicia sesión</h2>
26     </div>
27     <div class="row col-6 mx-auto">
28       <form action="j_security_check" method="POST" class="col-12">
29         <div class="form-group">
30           <label for="Email">Correo electrónico:</label> <input type="text"
31             class="form-control" id="Email" name="j_username"
32             aria-describedby="emailHelp" placeholder="Introduce el email">
33           <small id="emailHelp" class="form-text text-muted">Ejemplo:usuario@gmail.com</small>
34         </div>
35         <div class="form-group">
36           <label for="Password">Contraseña</label> <input type="password"
37             class="form-control" name="j_password" id="Password"
38             placeholder="contraseña">
39         </div>
40         <button type="submit" name="Login" class="btn btn-primary">Entrar</button>
41         <a href="/TFGCarlos/altausuario.jsp">Regístrate</a>
42       </form>
43     </div>
44   </div>
```

En el caso que el inicio de sesión haya sido incorrecto nos muestra la página “credencialesnovalidas.jsp” que tiene el mismo formato que la página “identificate.jsp” pero muestra un mensaje de error del inicio de sesión.

```
45 <div id="diverror" class="centrarTexto mx-auto col-7 mt-2 mb-2">
46   <p>
47     <strong><c:out value="Error" /></strong> <br>
48     <c:out value="Usuario o password incorrectos" />
49   </p>
50 </div>
```

Inicia sesión

Correo electrónico:

Introduce el email

Ejemplo:usuario@gmail.com

Contraseña

contraseña

Entrar

Regístrate

Error

Usuario o password incorrectos

OnlyFood

Conectar

En el caso que sea un usuario nuevo puede registrarse haciendo click en el enlace de “Regístrate”, que le redirige a la página “altaUsuario.jsp” que tiene un formulario de registro. Este formulario de registro tiene una seguridad añadida utilizando un recaptcha.

```

6<=<head>
7 <meta charset="UTF-8">
8 <title>OnlyFood</title>
9 <jsp:directive.include file="/WEB-INF/includes/includefiles.jspf" />
10<script
11   src="https://www.google.com/recaptcha/api.js?render=6LeKMsEUAAAAELRmoYEOzACgQJqHjhnLMw7z7"></script>
12<script>
13   grecaptcha.ready(function() {
14     grecaptcha.execute('6LeKMsEUAAAAELRmoYEOzACgQJqHjhnLMw7z7', {
15       action : 'altaUsuario'
16     }).then(
17       function(token) {
18         var recaptchaResponse = document
19           .getElementById('g-recaptcha-response');
20         recaptchaResponse.value = token;
21       });
22   });
23 </script>
24 </head>

62<= <div class="col-12 mt-4 mb-4">
63   <h2>Datos del nuevo usuario</h2>
64 </div>
65<= <div class="row col-6 mx-auto">
66<=   <form action="{pageContext.request.contextPath}/controller"
67     method="POST" class="col-12">
68<=     <div class="form-group">
69       <label for="Email">Correo electrónico:</label> <input type="email"
70         class="form-control" id="Email" name="email"
71         aria-describedby="emailHelp" placeholder="Introduce el email">
72       <small id="emailHelp" class="form-text text-muted">Ejemplo:usuario@gmail.com</small>
73     </div>
74<=     <div class="form-group">
75       <label for="Password">Contraseña</label> <input type="password"
76         class="form-control" name="clave" id="Password"
77         placeholder="contraseña">
78     </div>
79     <input type="hidden" name="operacion" id="operacion"
80       value="altaUsuario" />
81     <button type="submit" name="submit" class="btn btn-primary">Aceptar</button>
82     <a class="btn btn-primary" href="/TFGCarlos/index.jsp">Cancelar</a>
83     <input type="hidden" id="g-recaptcha-response"
84       name="g-recaptcha-response">
85   </form>

```


Una vez rellanados estos campos y enviado el formulario, te envía al controlador a la operación “altaUsuario”. En ella realiza verifica si la validación del recaptcha es válida, si no lo fuera enviaría un mensaje error de vuelta a la propia página. Seguidamente comprueba de que no hay un usuario registrado con esos datos mediante el método “encontrarUsuario” de la clase “DaoUsuario” (este método ya ha sido explicado en apartados anteriores), si no hay ninguno invoca al método “insertaUsuario” de la clase “DaoUsuario” que se encarga persistir el objeto de la entidad “Usuario” para insertarlo en la base de datos. Si todo el proceso se ha realizado correctamente redirige a la página “identificate.jsp” con un mensaje de confirmación de que el usuario ha sido registrado correctamente.

```
76         case "altaUsuario":
77             System.out.println("-----entra en altaUsuario-----");
78             //aqui se ve la forma de recoger una fecha y darle formato para posteriormente introducirla en un
79             DaoUsuario daoUsuario=new DaoUsuario();
80             Usuario usuario=new Usuario();
81             usuario.setUsuario(request.getParameter("email"));
82             usuario.setClave(request.getParameter("clave"));
83             String clave=request.getParameter("clave");
84             try {
85                 String grecaptcharesponse = request.getParameter("g-recaptcha-response");
86                 //request.setAttribute("confirmacion", confirmacion);
87                 request.setAttribute("email", (String) request.getParameter("email"));
88                 request.setAttribute("clave", clave);
89                 if(VerificarRecaptcha.validate(grecaptcharesponse)) {
90                     System.out.println("recaptcha valido");
91                     if(daoUsuario.encontrarUsuario(usuario.getUsuario())!=null)
92                         throw new UsuarioException("Ya existe un usuario con esas credenciales.");
93                     daoUsuario.insertaUsuario(usuario);//llamo al metodo que realiza la operacion
94                     System.out.println("termina a insertar usuario");
95                     String confirmacion="Socio "+usuario.getUsuario()+" dado de alta.";
96                     //usuario=daoUsuario.encontrarUsuario((String) request.getParameter("email"));
97                     //request.setAttribute("usuario",usuario);
98                     request.setAttribute("confirmacion",confirmacion);
99                     System.out.println("redirige a identificate");
100                     request.getRequestDispatcher("/seguridad/identificate.jsp").forward(request, response);
101                 }else{
102                     request.setAttribute("error", "Validación de captcha erronea");
103                     request.getRequestDispatcher("/altausuario.jsp").forward(request, response);
104                 }
105             }catch(UsuarioException e){
106                 String error=e.getMessage();
107                 request.setAttribute("error", error);
108                 request.getRequestDispatcher("/altausuario.jsp").forward(request, response);
109             }
110             }catch (RollbackException e) {
111                 String error="Error en la base de datos, escriba un correo con la incidencia a incidenciasOn";
112                 request.setAttribute("error", error);
113                 request.getRequestDispatcher("/altausuario.jsp").forward(request, response);
114             }catch (Exception e) {
115                 String error="Error: "+e.getMessage()+" escriba un correo con la incidencia a incidenciasOn";
116                 request.setAttribute("error", error);
117                 request.getRequestDispatcher("/altausuario.jsp").forward(request, response);
118             }
119         break;
```

6.5. Repositorio de GitHub.

En él se encuentra toda la información del proyecto, este repositorio posee dos ramas, una rama “master” que es la rama que posee toda la información de la aplicación que se ha comprobado que su funcionamiento es correcto y una rama “devel” que es en la que los desarrolladores subirán los cambios realizados para comprobar que su funcionamiento es correcto y si fuera correcto se actualizaría la rama “master” con la “devel”. Con él podemos ir viendo el seguimiento de los cambios que se han realizado en la aplicación.

```
MINGW64/c/Users/carlo/Desktop/TFG/GitHub-TFGCarlos
Carlos@LAPTOP-7LLAI93Q MINGW64 ~/Desktop/TFG/GitHub-TFGCarlos
$ git clone https://github.com/CCampayo-Azarquiel/OnlyFood-TFGCarlos.git
Cloning into 'OnlyFood-TFGCarlos'...
remote: Repository not found.
fatal: repository 'https://github.com/CCampayo-Azarquiel/OnlyFood-TFGCarlos.git/'
not found

Carlos@LAPTOP-7LLAI93Q MINGW64 ~/Desktop/TFG/GitHub-TFGCarlos
$ git init
Initialized empty Git repository in C:/Users/carlo/Desktop/TFG/GitHub-TFGCarlos/.git/

Carlos@LAPTOP-7LLAI93Q MINGW64 ~/Desktop/TFG/GitHub-TFGCarlos (master)
$ git remote add origin https://github.com/CCampayo-Azarquiel/OnlyFood-TFGCarlos.git

Carlos@LAPTOP-7LLAI93Q MINGW64 ~/Desktop/TFG/GitHub-TFGCarlos (master)
$ git checkout -b devel
Switched to a new branch 'devel'

Carlos@LAPTOP-7LLAI93Q MINGW64 ~/Desktop/TFG/GitHub-TFGCarlos (devel)
$ git status
## No commits yet on devel
?? TFGCarlos/
?? ojdbc6.jar
?? onlyFood.sql
?? payara5/
```

En este caso estoy inicializando el Git en la carpeta donde tengo guardados los archivos del proyecto. Le añado el acceso al repositorio con el comando “git remote add origin rutaRepositorio”, en este caso “origin” es como llamaríamos al remoto del repositorio. Después creo una rama devel y le añado los archivos con “git add .” y realizo una confirmación para que se queden guardados los archivos añadido mediante un “git commit -m mensaje”.

```
A payara5/mq/lib/props/broker/default.properties
A payara5/mq/lib/props/broker/install.properties
A payara5/mq/lib/tyrus-standalone-client.jar

Carlos@LAPTOP-7LLAI93Q MINGW64 ~/Desktop/TFG/GitHub-TFGCarlos (devel)
$ git commit -m "Añadiendo proyecto java, payara server y onlyfood.sql"
```

Y para subir los archivos al repositorio ejecuto el comando “git push origin devel”. Ya están subidos en esa rama, al ser correcto me cambio a la rama master realizo el comando “git merge devel” para que se actualicen la rama y subo rama master mediante el “push”.

Link del repositorio: <https://github.com/CCampayo-Azarquiel/OnlyFood-TFGCarlos>

7. Conclusiones y líneas futuras

Con este proyecto ha sido la primera vez que he realizado una aplicación web por completo por mí mismo con los conocimientos que he adquiriendo con los profesores del ciclo, y de esta forma afianzar esos conocimientos.

Al realizarlo me he dado cuenta de la gran complejidad que conlleva realizar cualquier aplicación web, ya que en esta ocasión a mi aplicación se le puede hacer muchas mejoras lo que conllevaría un mayor tiempo de desarrollo. Durante el desarrollo también he tenido que adquirir conocimientos nuevos que no se han impartido en el curso como Git y GitHub.

Las líneas futuras de mi aplicación son bastantes debido a que veo muchas formas de mejorar la aplicación. Algunas de estas mejoras son:

- Hacer una paginación cuando se muestran las ofertas de los usuarios o de las cadenas, en la que se elijan el número de ofertas que se muestren por página.
- Añadir una función para los usuarios administradores para poder crear un usuario administrador.
- Añadir una función para usuarios administradores de poder borrar un usuario.
- Añadir un mapa de geolocalización que muestre las cadenas de comida rápida que se encuentren en la página su restaurante más cercano. Y si entras en una cadena en concreto que muestre los de la cadena únicamente.
- Añadir una opción de poder modificar la imagen de la oferta.
- Añadir una opción de votación de las ofertas para saber las ofertas que más gustan y las que menos, de esa forma se muestren en orden por mayor número de votaciones positivas.
- Añadir una fecha de duración de la oferta para que cuando llegue a su fecha limite no se muestre la oferta.
- Añadir un filtro para que muestre las ofertas dependiendo de la opción que elijas como más nuevas, mejor valoradas, peor valoradas...
- Añadir una sección en la que el usuario pueda guardar sus ofertas favoritas.
- Añadir un servidor de correo para en el caso que un usuario haya olvidado su contraseña la pueda restablecer.

8. Bibliografía

- <https://www.oracle.com/es/cloud/free/>
- <https://www.dondominio.com/buscar/>
- <https://www.anw.es/alojamiento-web/alojamiento-hosting-java.html>
- https://finetwork.altas-internet.com/Finetwork_Fibra100Mb_Internet_2_0?encid=ClkTGRiAbptPsxe9G33dnw%253d%253d&c-sig=1589105937-6393672
- <https://getbootstrap.com/>
- <https://www.w3schools.com/>