

Cerca de La Huerta

Desarrollo de Aplicaciones Web – CIPFP Ausiàs March

Cerca de La Huerta

Contenido

Abstract	4
Introducción.....	5
Objetivos	6
Estudio de viabilidad.....	6
Descripción de la empresa	6
Oportunidad de negocio	6
La empresa	6
El servicio ofrecido	7
Estudio de mercado.....	7
Análisis externo	7
Análisis interno	7
Datos recogidos en el estudio de mercado.....	8
Análisis DAFO	9
Desarrollo del negocio	9
Producción.....	10
Valoración final	10
Tramites de inicio de actividad.....	10
Análisis del proyecto	11
El modelo de datos	11
Restricciones	12
La interfaz de usuario	12
Tecnologías 'frontend' utilizadas	12

Mapa de navegación.....	12
Interfaz web	13
Guía de estilos	17
Usabilidad Web	18
Implantación de accesibilidad y estándares.....	19
Revisión de la normativa aplicable	20
Desarrollo del proyecto	21
Aplicación Cliente.....	22
Funcionalidad de la aplicación cliente	23
Aplicación Servidor	30
Funcionalidad de la aplicación servidor	31
Futuras ampliaciones y mejoras	42
Bibliografía y enlaces	42
Licencia.....	43
Agradecimientos	43

Abstract

Cerca de la Huerta is a Web application, which aims to bring the orchard, its culture and its products closer to the city. It is both a tool for dissemination and communication, and an opportunity to market the crops, which are so difficult to sell today, although there is a great demand for them. Producers may offer their harvest to potential customers, without going through any intermediary.

For the development of the application, different technologies have been used, among which AngularJS, JavaEE, HTML and CSS stand out.

In this report the different sections of the Web application are broken down, as well as the tools used for its creation and the motivations and objectives to carry it out.

Cerca de la Huerta es una aplicación Web, que pretende acercar la huerta, su cultura y sus productos, a la ciudad. Es tanto una herramienta de difusión y comunicación, como una oportunidad de comercializar los productos del campo, que tan difícil salida tienen hoy en día, aunque paradójicamente haya una gran demanda de éstos. Los productores podrán ofertar sus cultivos a potenciales clientes, sin pasar por ningún intermediario.

Para el desarrollo de la aplicación, se han utilizado diferentes tecnologías, entre las que destacan AngularJS, JavaEE, HTML y CSS.

En esta memoria se desglosan los diferentes apartados de la aplicación Web, así como las herramientas usadas para su creación y las motivaciones y objetivos para llevarla a cabo.

Introducción

Cada temporada veo la misma situación; campos sin recoger, con la cosecha echándose a perder, porque el producto local no tiene una salida competitiva en este mercado globalizado. Por otra parte, en las ciudades, hay gente dispuesta a pagar un precio justo por obtener productos de calidad, frescos y saludables. El problema está en que al mercado intermediario no le interesa la producción local, ya que no ofrece los beneficios que esperan.

El resultado, además de lo ya mencionado, es que nuestros mercados están en la mayoría de los casos, abastecidos con productos de baja calidad, ya sea por modificaciones transgénicas o por los métodos de distribución. Métodos éstos que al mismo tiempo generan un problema medioambiental y un sobrecoste para el consumidor final.

Cerca de La Huerta, nace con la intención de dar a conocer el mundo de la huerta, de ser un punto de encuentro para gente del campo y de la ciudad. Por medio de la publicación de noticias de interés, en las que los/as lectores/as podrán conocer la vida en la huerta, o compartir experiencias con otros/as agricultores/as.

Además, se tendrá la posibilidad de vender, comprar (a través de la tienda) o intercambiar (a través del blog/sección de noticias), los productos que no renta vender al mercado mayorista, en la cantidad que se deseé.

La aplicación consta de 3 partes bien definidas, La parte cliente, con una interfaz clara y sencilla con la que interactuarán los usuarios, se encarga de mostrar y enviar la información. La parte servidor, que se encarga de procesar la información, comunicar con la base de datos y enviar los resultados. Finalmente, la base de datos, almacena la información de manera permanente sobre usuarios, productos y noticias.

Tanto la aplicación de cliente (AngularJS) como la de servidor (JavaEE), utilizan el patrón MVC (Modelo – Vista – Controlador), separando las diferentes partes de la aplicación, haciéndola más sencilla de mantener y dotándola de mayor escalabilidad. Se utiliza una base de datos relacional gestionada con el SGBD de código abierto MariaDB.

Objetivos

Los objetivos principales son claros y ya han sido expuestos: Se pretende realizar una plataforma en la que la cultura y la forma de vivir en la huerta, puedan llegar a todas partes. Al mismo tiempo, se facilitará una salida al mercado de la producción local y una forma de obtener estos productos directamente de las personas que los cultivan, sin intermediarios de ningún tipo. Para lograrlo, se tendrán en cuenta estas premisas:

- > Sitio Web fácil de utilizar y con una estructura sencilla.
- > Sección de noticias para comunicarse e informar.
- > Sección tienda para la venta de productos de la huerta.
- > Sección de administración para el mantenimiento.

Como alumno, el proyecto es la herramienta perfecta para afianzar y ampliar los conocimientos adquiridos a lo largo del ciclo.

Estudio de viabilidad

Si bien el proyecto está planteado sin ánimo de lucro y el beneficio que se pudiera obtener sería simplemente para cubrir los gastos de producción y mantenimiento, a continuación se detalla el estudio de viabilidad realizado.

Descripción de la empresa

Oportunidad de negocio

La necesidad que cubre la aplicación tiene dos direcciones, por un lado los proveedores/productores, con mercancía a la que no pueden dar salida y, por otro, la creciente demanda de productos de calidad del campo (frutas, verduras...), que desde las ciudades tienen muchos potenciales clientes.

La empresa

El domicilio social de la empresa se sitúa en Valencia capital, aunque dada la naturaleza del negocio, podrá operar tanto a nivel nacional como internacional, siempre que exista tanto una demanda del producto ofrecido, como productores locales dispuestos a darse a conocer y ofrecer sus mercancías.

Inicialmente, se constituirá como empresa individual/autónomo, ya que no se espera un volumen de negocio excesivo ni una gestión del mismo que requiera más personal que el propio desarrollador. En un futuro y de ser necesario, pasaría a formarse una empresa con una forma jurídica de responsabilidad limitada.

El nombre de la empresa es 'Cerca de La Huerta'.

El servicio ofrecido

Desde Cerca de La Huerta, se facilitará la comunicación y venta directa de los productos del campo. Con espacios dedicados para la comunicación y la venta.

Estudio de mercado

Análisis externo

Si bien se pueden encontrar algunos sitios Webs con finalidad similar, son generalmente tiendas de distribuidores para captar venta directa. Cerca de La Huerta no pretende ser la competencia de otras tiendas, ni mucho menos una alternativa a los grandes distribuidores, sino más bien un espacio de comunicación y obtención de presencia para el pequeño productor.

A pesar de buscar el mercado cercano, hoy por hoy es cada vez más imprescindible el uso de las nuevas tecnologías para la comunicación, independientemente del sector en el que nos encontremos.

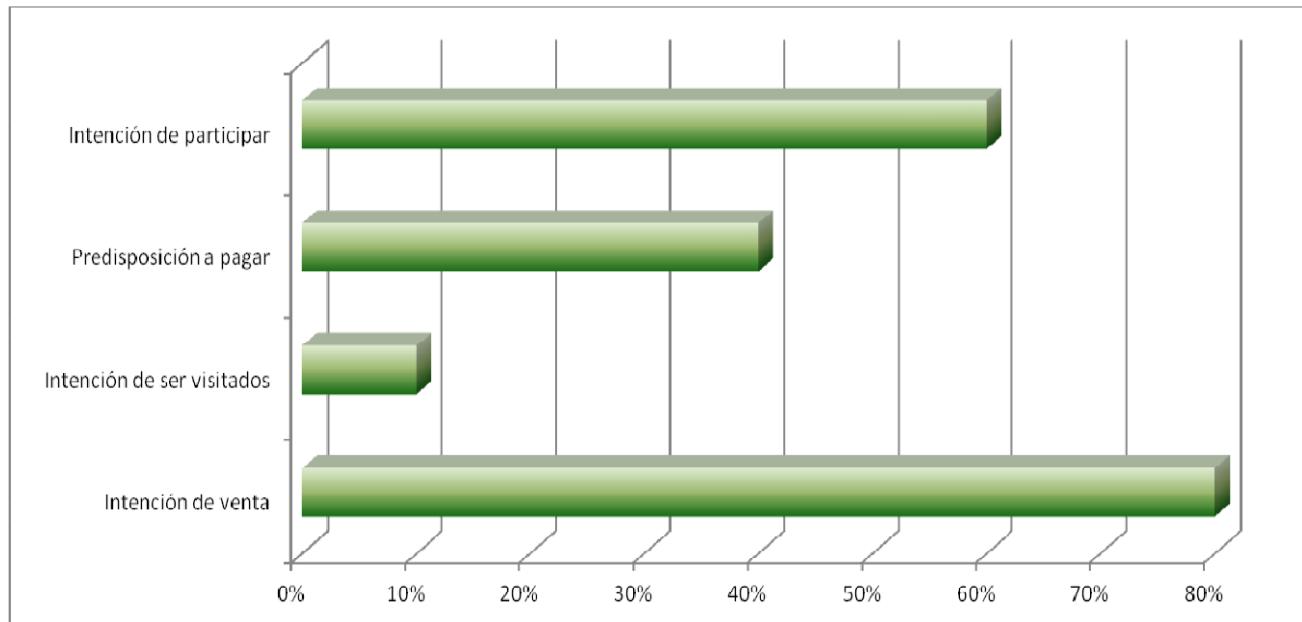
La empresa Cerca de La Huerta, con el uso de la aplicación, podrá llegar a sus clientes con mayor facilidad. Estos clientes serán tanto los compradores como los vendedores que deseen participar en el sitio Web, que se servirán de esta herramienta para ponerse en contacto.

Análisis interno

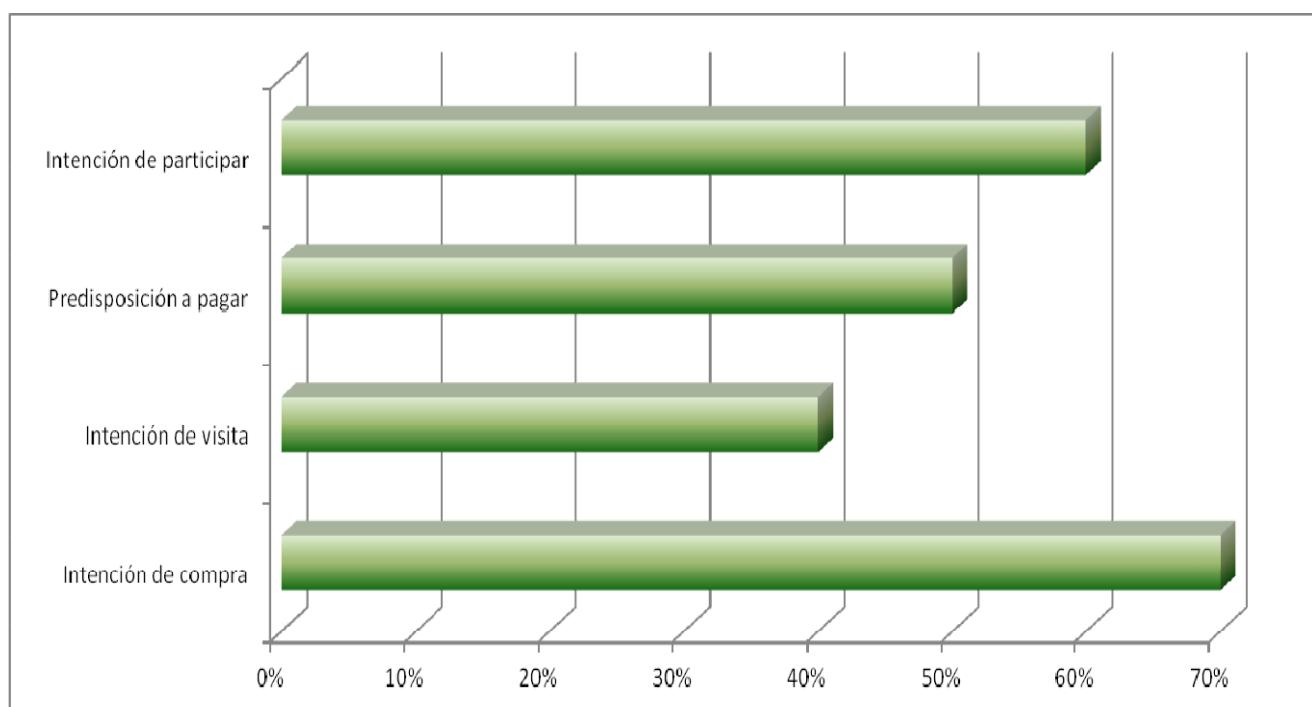
El desarrollador, una vez desplegado el sitio Web, será el encargado de administrarlo y mantenerlo aunque, con el tiempo, se le ofrecerá a otros usuarios la posibilidad de participar como colaboradores, especialmente en la sección de noticias.

Datos recogidos en el estudio de mercado

Productores



Compradores



Análisis DAFO



Desarrollo del negocio

El Servicio

El servicio ofrecido consiste en el diseño, desarrollo y mantenimiento de la aplicación Web, así como el despliegue y alojamiento en servidor y la contratación del dominio.

Precio

La producción y puesta en marcha del servicio, tendrá un coste de 1500 €, asumido por la empresa, que es quien realiza el desarrollo.

Para el mantenimiento de la página, en un principio se calculará sobre un porcentaje de las ventas y compras, siendo este el mínimo necesario para hacer los pagos requeridos. De este modo, tanto vendedores como compradores, aportarán una mínima cantidad, siempre y cuando se realicen operaciones mercantiles a través del sitio Web. En un futuro, cabe la posibilidad de incluir publicidad no invasiva, por medio de Google Add Sense o similar, exclusivamente para cubrir los costes de mantenimiento. Se calcula que los ingresos mensuales deberán ser de unos 250 € inicialmente, cantidad que se incrementará a medida que el negocio se expanda y crezca la cantidad de usuarios.

Producción

El desarrollo y puesta en marcha de la aplicación Web -> 1500 €

Mantenimiento y administración (10 horas mensuales) -> 100 € mensuales

Dada la naturaleza del negocio, los gastos se limitarían al alojamiento Web, pago del dominio, comunicaciones (Conexión a Internet y línea móvil). Siendo el gasto en comunicaciones no exclusivo para el negocio y al no necesitar un plan de alojamiento Web muy exigente inicialmente, se calcula que el gasto podría ser de 100 € mensuales aproximadamente. Así mismo, El trabajo se realizará en las actuales instalaciones de la empresa (domicilio del desarrollador), por lo que los costes de local, electricidad y desgaste de equipo informático, serán estimados como parte proporcional del coste total de las instalaciones. Este cálculo nos da un coste de unos 50 € mensuales.

Valoración final

La empresa cuenta con un capital inicial de 10.000 €, por lo que si bien la estimación de gastos e ingresos es de mínimos, se cuenta con solvencia para poner el negocio en marcha. Dado que el mercado potencial es muy amplio, una vez funcionando el negocio, se han desarrollado estrategias para la monetización del mismo. Además, casi el 100% del código de la aplicación es re-utilizable para futuros proyectos, por lo que el coste de desarrollo y mantenimiento, son compartidos por la actividad general del desarrollador/la empresa.

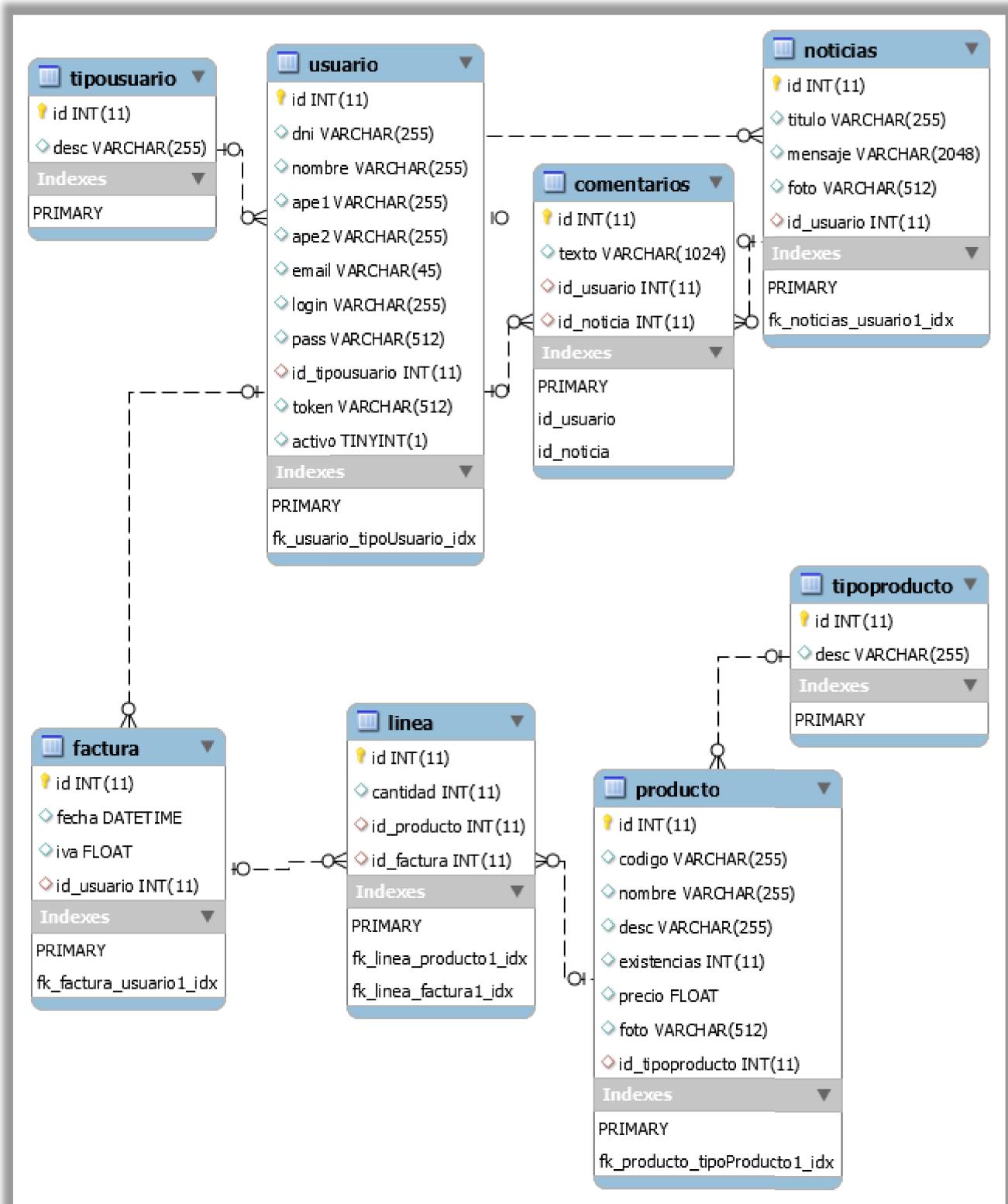
Tramites de inicio de actividad

Como empresario individual, los trámites a realizar son los siguientes:

- > Alta en el censo de obligados tributarios
- > Alta en el impuesto de actividades económicas
- > Diligenciar los libros contables
- > Inscripción de la empresa en la Seguridad Social
- > Alta en autónomos
- > Alta de los trabajadores (a futuro)
- > Adquisición y sellado del libro de visitas
- > Comunicación de apertura

Análisis del proyecto

El modelo de datos



Restricciones

En todas las tablas con claves ajenas, existen las mismas restricciones:

'on delete set null, on update cascade'

Lo que significa que si se elimina el registro al que hace referencia la clave ajena, el campo de la tabla se configura como 'null' (nulo), mientras que si se actualiza ese registro, la actualización se produce también en la tabla que posee esa clave ajena.

La interfaz de usuario

Tecnologías 'frontend' utilizadas

Para el desarrollo de la interfaz se ha utilizado Bootstrap 4, Material Design Bootstrap, CSS (apoyado en ocasiones con JavaScript) y se han incluido elementos de CCS GRID y CCS Flexbox.

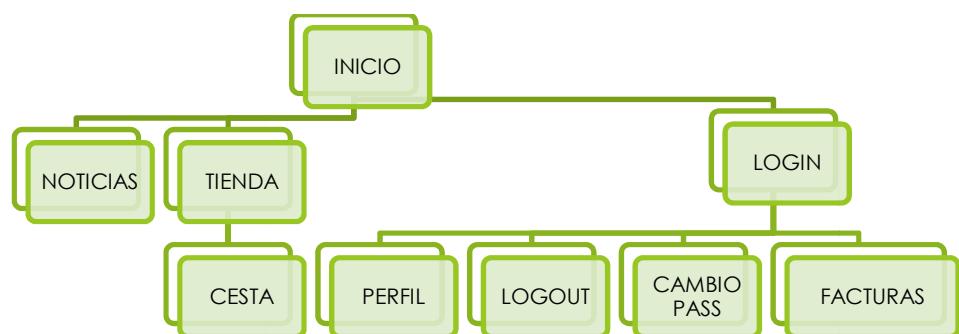
Mapa de navegación

En todo momento se puede volver a la página de inicio o a la anterior. La navegación se realiza desde el encabezado, que está siempre visible. Dependiendo del tipo de usuario, la navegación es la siguiente:

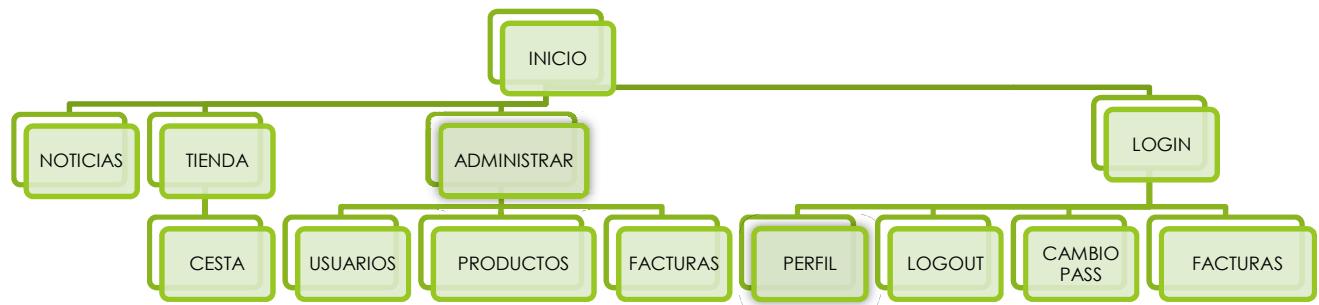
Usuario no registrado:



Usuario registrado:



Usuario administrador:



Interfaz web

El objetivo de la interfaz es que la navegación por la aplicación sea lo más sencilla posible, sin abrumar al usuario con demasiadas opciones. Por tanto se ha huido de interfaces recargadas y se ha puesto el énfasis en dejar las opciones disponibles claras y simples de utilizar.

La página de inicio, presenta las partes principales del sitio Web y se intuye el diseño de las diferentes secciones del sitio:

- Encabezado: Incluye el logo y nombre de la empresa (identificación), la barra de navegación por la diferentes secciones (navegación), y el botón/desplegable para el login, logout y opciones del usuario (interacción).
- Imagen de bienvenida con nombre de la empresa y slogan (identificación).
- Noticias: Incluye las noticias más actuales y tiene acceso directo a la sección de noticias (contenido).
- 'Call To Action' de registro (interacción)
- Productos; incluye los productos con más stock y enlaza con la sección de compras (contenido/interacción)
- Pie de página: Información del proyecto (identificación).



Quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Descubre con nosotros el día a día de la Huerta



El campo este año
Carlos Garcia

Parturient signiferumque adhuc porro voluptatum varius. Quotnam doctus eirmod interesseret quem voluptatibus iudicabit etiam scripta adversarium velit ...



El clima este año
Lucia Gonzalez

Delectus condimentum oporteat affert aliquip autem constituto platea. Torquent minim alia tibique quaque sapiente populo nominavi condimentum saepe ...

Forma parte de Cerca de la Huerta

[¡REGISTRATE!](#)

Directamente de la huerta, recibelo en tu casa o ven a por ello



tomates de temporada

Consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam.



tomates maduros

Consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam.



setas maduros

Consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam.

Proyecto final - DAW CPIFP Ausiás March
Alejandro Llamas

Listado de usuarios, con acceso a las acciones de ver facturas, ver, editar y eliminar:

ID	DNI	Nombre	Primer apellido	Segundo apellido	Tipo de usuario	Acciones
51	98741236L	Manuela	Perez	Gonzalez	cliente	[Delete]
52	98741236L	Lucia	Gonzalez	Perez	administrador	[Delete]
53	96325612H	Carlos	Marquez	Perez	administrador	[Delete]
54	96325612H	Carlos	Garcia	Martinez	administrador	[Delete]
55	96325612H	Lucia	Garcia	Marquez	administrador	[Delete]
56	98741236L	Maria	Gonzalez	Perez	administrador	[Delete]
57	96325612H	Marta	Perez	Gonzalez	administrador	[Delete]
58	25896321P	Carlos	Marquez	Perez	administrador	[Delete]
59	9654123F	Lucia	Gonzalez	Martinez	cliente	[Delete]
60	96325612H	Victor	Perez	Gonzalez	administrador	[Delete]

Proyecto final - DAW CPIFP Ausiás March
Alejandro Llamas

Formulario editar usuario:

Editar usuario

DNI: 98741236L

Nombre: Manuela

Primer apellido: Perez

Segundo apellido: Gonzalez

Correo electronico: bemu@bemu.com

Login: bemu

Tipo de usuario:

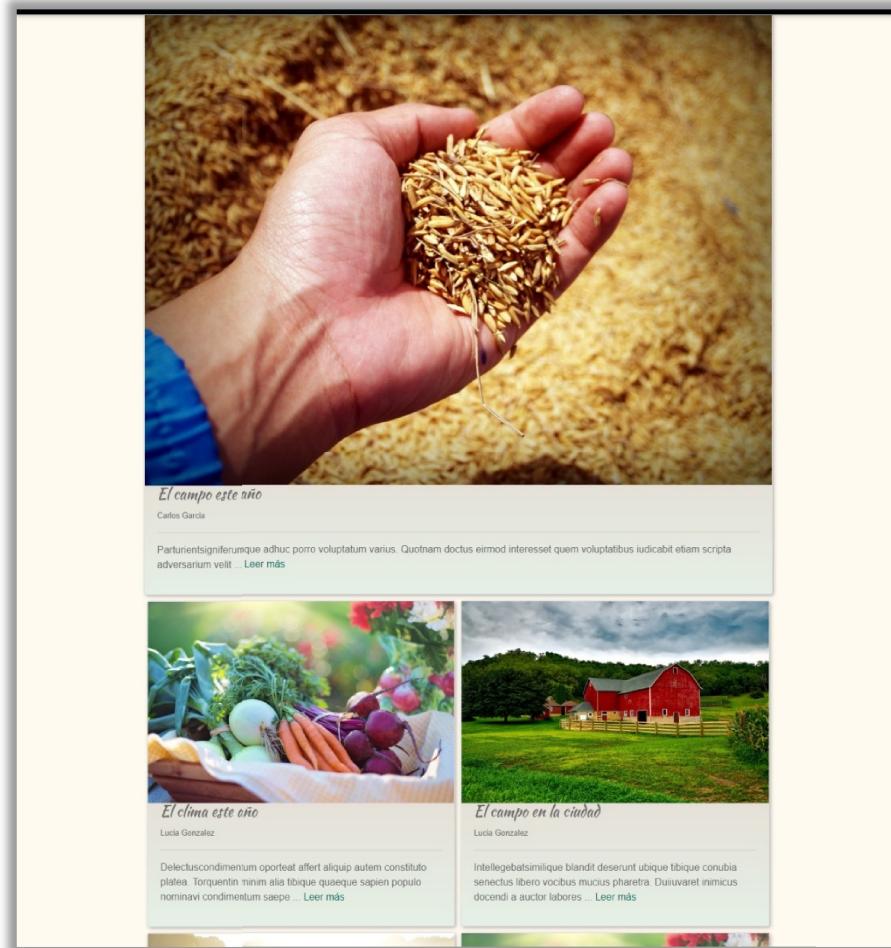
2 Tipo de usuario: cliente

EDITAR

VOLVER

Proyecto final - DAW CPIFP Ausiás March
Alejandro Llamas

Noticias/blog:



Zona de compras:

La cosecha

tomates de temporada	tomates de temporada	espárragos grandes	tomates frescos
Consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam. Consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam.	Consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam.	Consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam.	Consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam.
0	4	2	0

Guía de estilos

Clasificación de elementos

Los elementos de los que consta esta Web son:

> Elementos de identificación

Constarán en un principio del título de la página y de la cabecera, que es el nombre de la empresa.

> Elementos de navegación

El menú de la cabecera y los botones de 'volver', así como los enlaces a las diferentes zonas de la aplicación.

> Elementos de contenido

Se compone de todo el texto, imágenes y animaciones que muestran y explican tanto las noticias y los productos, como los diferentes elementos de administración.

> Elementos de interacción

Los botones de acciones y los formularios.

Estilos utilizados

Colores

Los colores principales (corporativos), de Cerca de la Huerta son:

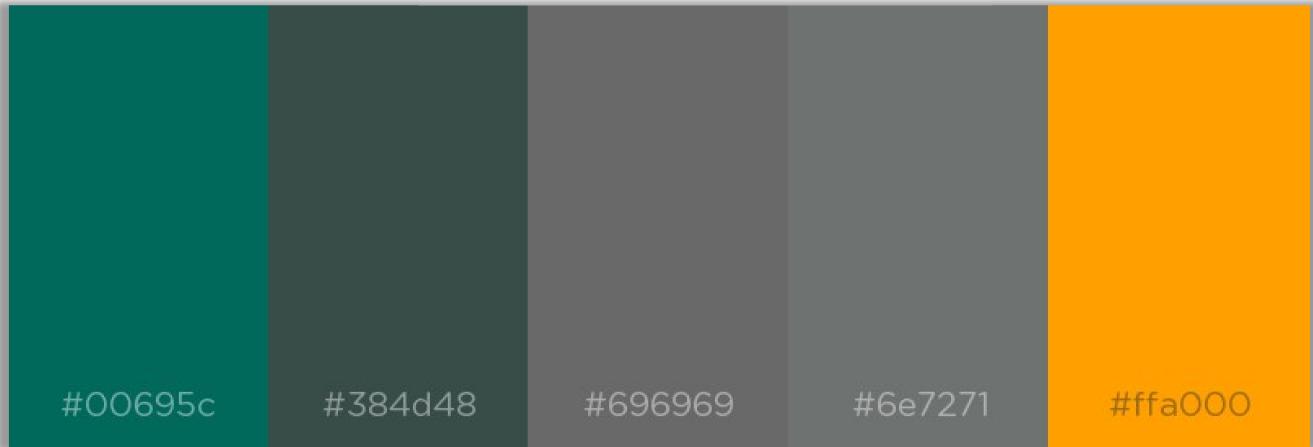
- > Dark green (#00695c) -> Verde oscuro
- > Light orange (#ffa000) -> Naranja

Se han utilizado en el logo, encabezado, pie de página y botones.

En listados, formularios, las tarjetas de noticias y productos, se juega con degradados y opacidad, para dar uniformidad y coherencia a la Web, y al mismo tiempo resaltar el contenido (texto e imágenes).

De forma genérica, en el texto se ha utilizado el color dimgray (#696969), resaltando en blanco, negro o en los colores principales de la Web, textos como menús, títulos, botones o texto sobre imágenes, según el fondo para dar mayor contraste.

La gama de colores propuesta a seguir es la siguiente:



Fuentes

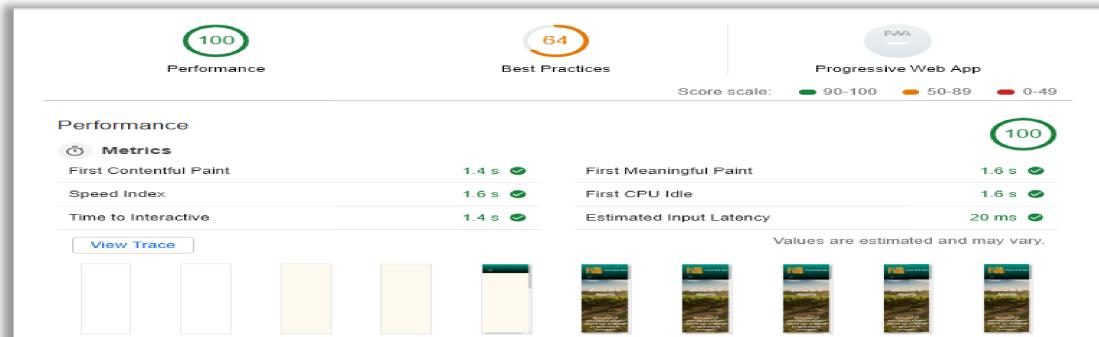
La familia de fuentes utilizada en el proyecto en general es: 'Arial, Helvetica, sans-serif', lo que asegura una buena legibilidad y compatibilidad.

Al igual que con los colores, se ha decidido resaltar el menú de navegación y los títulos principales con otra fuente. En este caso se utiliza la fuente 'Kaushan Script' (de Google Fonts) con 'cursive' de respaldo. Esta fuente está embebida en el proyecto, mediante enlace a 'googleapis'. Del mismo modo se ha embebido 'fontawesome' para utilizar sus iconos gratuitos.

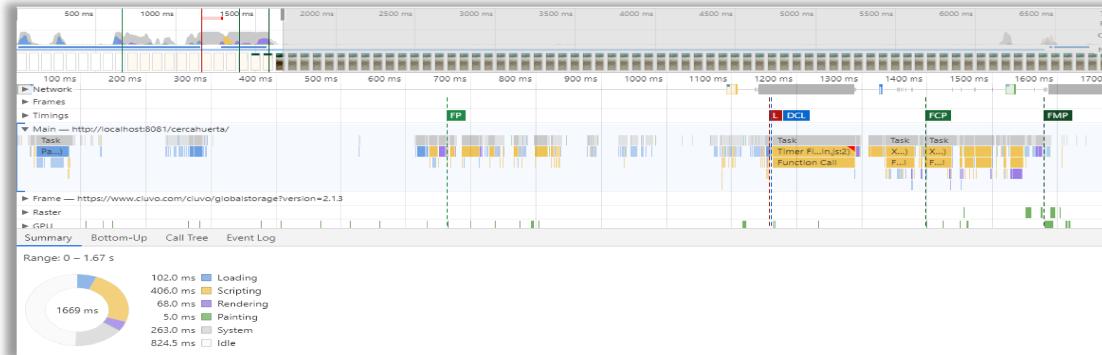
Usabilidad Web

La aplicación se ha diseñado desde el principio, con la intención de que sea lo más sencilla posible, con menús claros y siempre disponibles, sin opciones o contenido innecesario que pueda despistar al usuario. Es una interfaz intuitiva en la que se pretende que el usuario sepa siempre dónde está lo que busca. Siempre que ha sido posible, se ha huido de recursos innecesarios para que la Web cargue lo más rápidamente posible y su manejo sea ligero y sin retardos injustificados.

Herramienta de auditoría de Google Chrome (usabilidad) I:



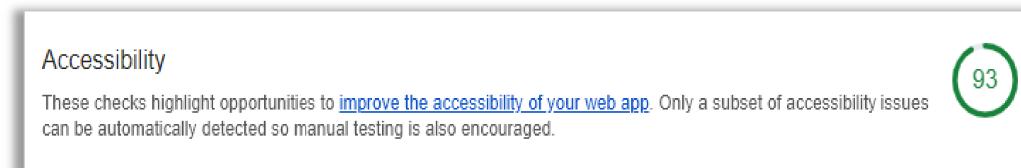
Herramienta de auditoría de Google Chrome (usabilidad) II:



Implantación de accesibilidad y estándares

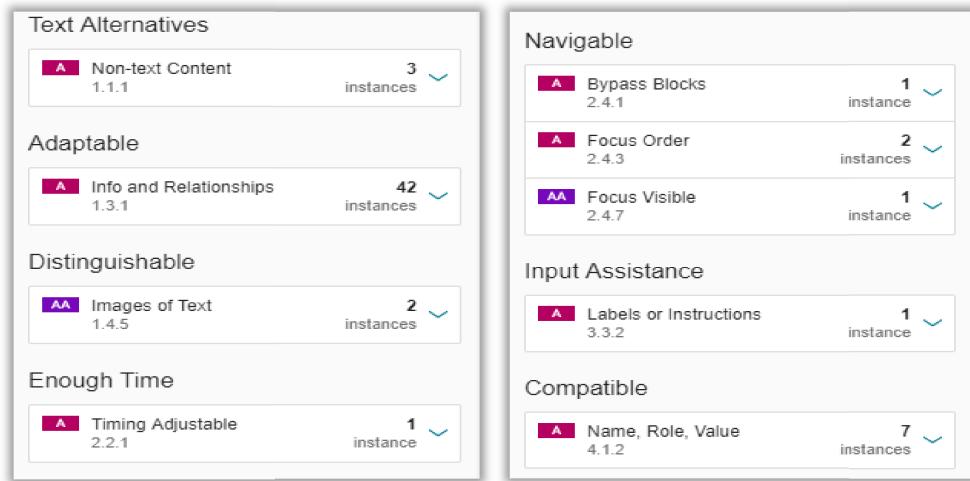
En todo momento, se ha cuidado que el texto tenga un contraste que, sin ser molesto o poco atractivo, facilite la lectura del mismo. Además se han utilizado textos de ayuda, como 'titles' y 'placeholders' tanto en campos de formulario como en imágenes, para ayudar a la comprensión de la Web. En la disposición del contenido, se ha cuidado que el desplazamiento por tabulación sea coherente con la disposición de los diferentes elementos.

Herramienta de auditoría de Google Chrome (accesibilidad):



Passed audits		13 audits
1	[aria-*] attributes match their roles	✓ ▾
2	[aria-*] attributes have valid values	✓ ▾
3	[aria-*] attributes are valid and not misspelled	✓ ▾
4	Buttons have an accessible name	✓ ▾
5	The page contains a heading, skip link, or landmark region	✓ ▾
6	Background and foreground colors have a sufficient contrast ratio	✓ ▾
7	Document has a <title> element	✓ ▾
8	[id] attributes on the page are unique	✓ ▾
9	<html> element has a [lang] attribute	✓ ▾
10	<html> element has a valid value for its [lang] attribute	✓ ▾
11	Image elements have [alt] attributes	✓ ▾
12	Links have a discernible name	✓ ▾
13	[user-scalable="no"] is not used in the <meta name="viewport"> element and the [maximum-scale] attribute is not less than 5.	✓ ▾

Siteimprove Accesibility checker:



Revisión de la normativa aplicable

Se cumplirá rigurosamente con el Reglamento Europeo de protección de datos (RGPD), aprobado en mayo de 2016 y en vigor desde 2018. Para ello se seguirán los siguientes principios:

- > Principio de licitud, lealtad y transparencia
 - o Los datos serán recogidos de manera lícita, leal y transparente.
- > Principio de limitación de la finalidad
 - o Los datos serán recogidos con fines determinados.
- > Principio de minimización de datos
 - o Se recogerán los datos estrictamente necesarios en relación con los fines para los que son tratados.
- > Principio de limitación del plazo de conservación
 - o Los datos serán mantenidos durante no más tiempo del necesario para los fines del tratamiento.
- > Principio de integridad y confidencialidad
 - o Los datos serán tratados de tal manera que se garantice una seguridad adecuada de los datos personales.
- > Principio de responsabilidad proactiva
 - o El responsable del tratamiento será también responsable del cumplimiento de lo dispuesto en el Reglamento.

En cuanto a la inscripción de ficheros, se seguirá la normativa vigente:

"La obligación de notificar ficheros se sustituye a partir del 25 de mayo de 2018 por elaborar un registro de actividades de tratamiento que deberá contener la información señalada en el artículo 30 del [...] Reglamento." (<https://www.aepd.es/herramientas/inscripcion-ficheros.html>)

Desarrollo del proyecto



NetBeans

IDE (Entorno de desarrollo)

NetBeans es un entorno de desarrollo integrado basado en Java (IDE). El término también se refiere al marco de la plataforma de aplicación subyacente del IDE.

El IDE está diseñado para limitar los errores de codificación y facilitar la corrección de errores con herramientas como NetBeans FindBugs para localizar y solucionar problemas comunes de codificación de Java y Debugger para administrar código complejo con vigilancia de campo, puntos de interrupción y monitoreo de ejecución.



Tomcat

Servidor de la aplicación

Tomcat es un servidor de aplicaciones de Apache Software Foundation que ejecuta servlets de Java y representa páginas web que incluyen la codificación de la página del servidor Java.

Es una "implementación de referencia" del Servlet de Java y las especificaciones de la página del Servidor de Java, Tomcat es el resultado de una colaboración abierta de desarrolladores y está disponible en el sitio web de Apache en versiones binarias y de origen.



Base de datos

MariaDB es un sistema de gestión de base de datos relacional de código abierto (SGBD) que es un reemplazo compatible para la tecnología de base de datos MySQL ampliamente utilizada.

Fue creado como una bifurcación de software de MySQL por los desarrolladores que jugaron roles clave en la construcción de la base de datos original; idearon MariaDB en 2009 en respuesta a la adquisición de MySQL por Oracle Corp.



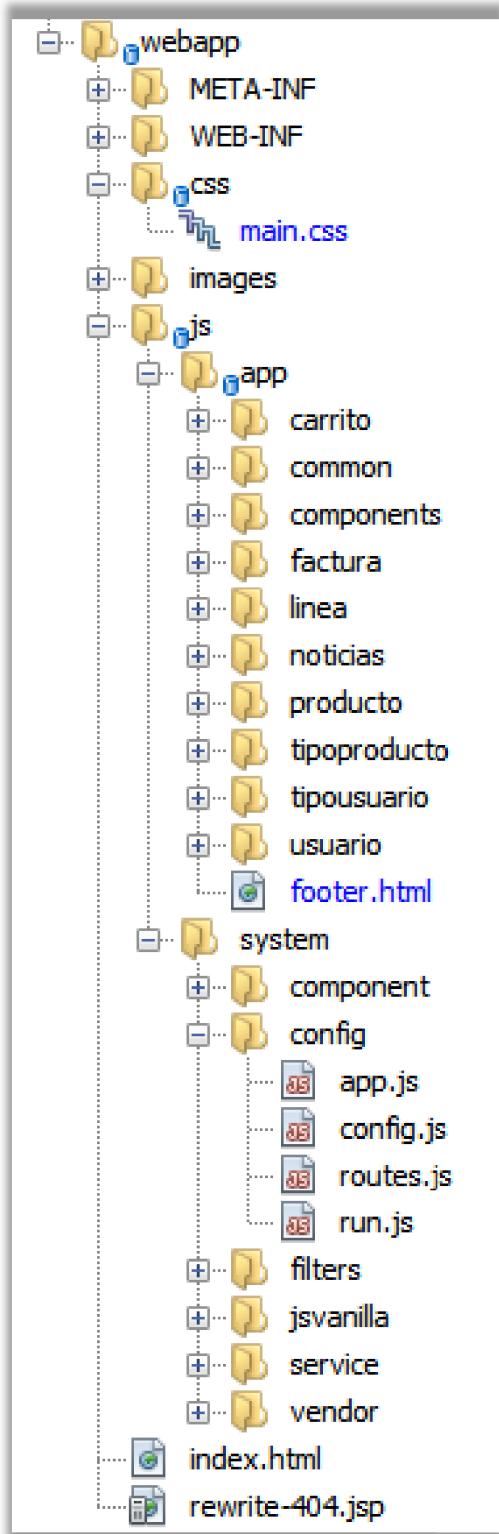
APACHE

Servidor web

Apache es un servidor web que se distribuye bajo una licencia de "código abierto".

Es rápido, confiable y seguro. Puede ser altamente personalizado para satisfacer las necesidades de muchos entornos diferentes mediante el uso de extensiones y módulos.

Aplicación Cliente



AngularJS

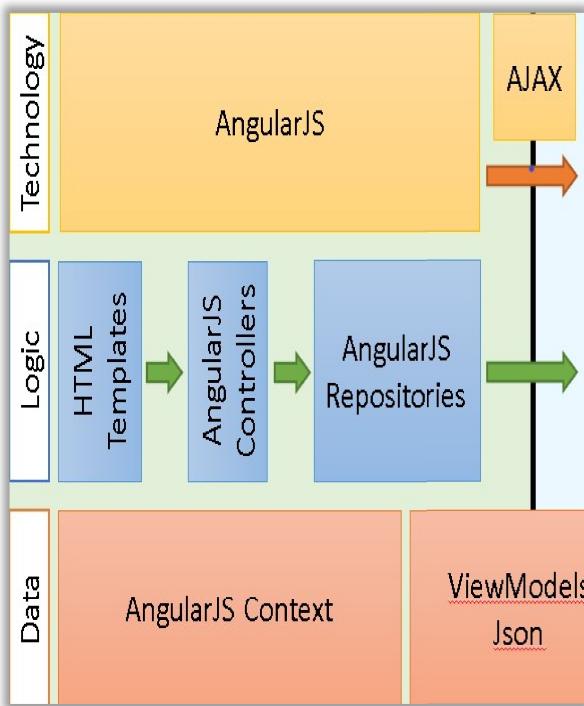
En el lado cliente, se ha utilizado AngularJS, un framework de JavaScript de código abierto. Es un framework muy popular entre los desarrolladores. Para aplicaciones de una sola página, el framework AngularJS crea funciones interactivas para una experiencia en tiempo real. Es sencillo para los desarrolladores y tiene una comunidad muy activa. Sigue el patrón MVVM (Model View View-Model) de ingeniería de software y alienta la articulación flexible entre la presentación, datos y componentes lógicos. Con el uso de la inyección de dependencias, Angular lleva servicios tradicionales del lado del servidor, tales como controladores dependientes de la vista, a las aplicaciones web del lado del cliente. En consecuencia, gran parte de la carga en el backend se reduce, lo que conlleva a aplicaciones web mucho más ligeras.

AJAX

Para la comunicación con el servidor, se ha optado por utilizar AJAX (Asynchronous JavaScript And XML; JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

Funcionalidad de la aplicación cliente

Single Page Application



Single page application (SPA) o aplicación de página única, consiste en una aplicación web o sitio web **que cabe en una sola página**. En otras palabras, la página web sólo se “carga” una vez con el propósito de dar una experiencia más fluida. En una SPA todos los códigos de HTML, JavaScript, y CSS se cargan de una vez.

Ventajas:

- > Experiencia de usuario mejorada.
- > Las páginas web se actualizan más rápido y se usa menos ancho de banda.
- > La implementación de la aplicación en la producción es más fácil.
- > La división de código se puede hacer para dividir los paquetes en varias partes.

Desarrollo de la SPA

Creación del modulo

Siguiendo la arquitectura MVC, lo primero es crear un modulo para albergar controladores, servicios y demás:

```
var cercahuerta = angular.module('MyApp', [
  'ngRoute',
  'Filters',
  'services',
  'components',
  'commonControllers',
  'tipousuarioControllers',
  'usuarioControllers',
  'tipoproductoControllers',
  'facturaControllers',
  'productoControllers',
  'lineaControllers',
  'carritoControllers',
  'noticiasControllers',
  'ngMaterial',
  'Directives'
```

```
var moduleCommon = angular.module('commonControllers', []);
var moduleService = angular.module('services', []);
var moduloFiltros = angular.module('Filters', []);
var moduleComponent = angular.module('components', []);
var moduleTipousuario = angular.module('tipousuarioControllers', []);
var moduleUsuario = angular.module('usuarioControllers', []);
var moduleProducto = angular.module('productoControllers', []);
var moduleFactura = angular.module('facturaControllers', []);
var moduleTipoproducto = angular.module('tipoproductoControllers', []);
var moduleLinea = angular.module('lineaControllers', []);
var moduleCarrito = angular.module('carritoControllers', []);
var moduleNoticias = angular.module('noticiasControllers', []);
var moduloDirectivas = angular.module('Directives', []);
```

Definición de controladores

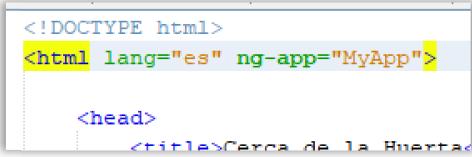
Los controladores que se encargarán de las diferentes vistas. En esta imagen se puede ver parte del controlador para la vista listado de usuarios:

```
moduleUsuario.controller('usuarioPlistController', ['$scope', '$http', '$location', 'toolService', '$routeParams',
  function ($scope, $http, $location, toolService, $routeParams) {
    $scope奥巴 = "usuario";
    $scope.totalPages = 1;

    $scope.isActive = toolService.isActive;
    if (!$routeParams.order) {
```

La vista principal o padre

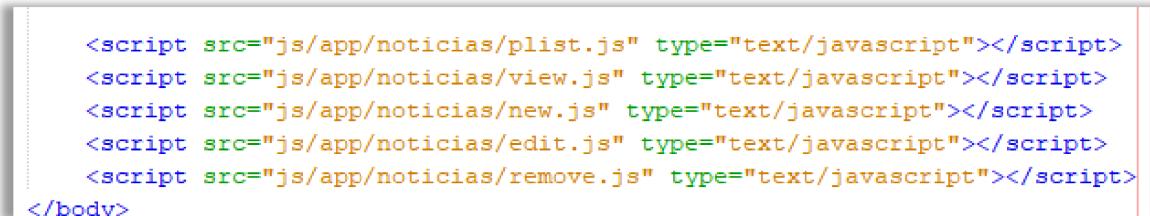
En esta vista, además de cargarse los diferentes archivos de estilos (.css) y scripts (.js) de la aplicación, se carga el script de AngularJS, el de angular-route (para el manejo del tandem 'vista-controlador') y la directiva ng-app que arranca la aplicación estableciendo el elemento raíz de la misma.



```
<!DOCTYPE html>
<html lang="es" ng-app="MyApp">

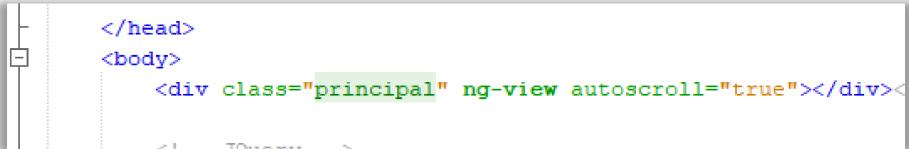
  <head>
    <title>Cerca de la Huerta</title>

  <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/mdbbootstrap/4.7.0/js/mdb.min.js"></script>
  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.5.6/angular.min.js"></script>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.6/angular-route.min.js"></script>
  <script src="https://unpkg.com/jspdf@latest/dist/jspdf.min.js"></script>
  <script src="js/system/config/app.js" type="text/javascript"></script>
  <script src="js/system/vendor/moment.js" type="text/javascript"></script>
```



```
<script src="js/app/noticias/plist.js" type="text/javascript"></script>
<script src="js/app/noticias/view.js" type="text/javascript"></script>
<script src="js/app/noticias/new.js" type="text/javascript"></script>
<script src="js/app/noticias/edit.js" type="text/javascript"></script>
<script src="js/app/noticias/remove.js" type="text/javascript"></script>
</body>
```

En la vista principal, se utiliza la directiva 'ngView', para indicar a la aplicación donde se cargarán las diferentes vistas que se irán llamado con el sistema de rutas, es decir, en una sola página, la de la vista principal, se irán recargando las diferentes vistas de la aplicación.



```
</head>
<body>
  <div class="principal" ng-view autoscroll="true"></div>
  <!-- jQuery -->
```

Vista secundaria o hija

Cada controlador maneja una vista con la que el usuario interactúa. Aquí se muestra parte de la plantilla que genera la vista de listado de usuarios:

```
<tr ng-repeat="fila in ajaxDataUsuarios">
    <td>{{fila.id}}</td>
    <td>{{fila.dni}}</td>
    <td>{{fila.nombre}}</td>
    <td>{{fila.apel1}}</td>
    <td>{{fila.apel2}}</td>
    <td><a href="tipousuario/view/{{fila.obj_tipoUsuario.id}}">{{fila.obj_tipoUsuario.desc}}

```

Con la directiva ngRepeat, se construye un bucle que repetirá el elemento, tantas veces como registros tenga la respuesta recibida del servidor.

Definición de rutas

Con el servicio 'routeProvider' del modulo 'ngRoute', se especifica un 'templateUrl' (plantilla que genera la vista) y un controlador para cada ruta que se agrega:

```
cercahuerta.config(['$routeProvider', function ($routeProvider) {
    //HOME
    $routeProvider.when('/', {templateUrl: 'js/app/common/home.html', controller: 'homeController', resolve: {auth: autenticacionCualquiera}});

    //USUARIO
    $routeProvider.when('/usuario/plist/:rpp?:page?:order?', {templateUrl: 'js/app/usuario/plist.html', controller: 'usuarioPlistController',
    $routeProvider.when('/usuario/view/:id', {templateUrl: 'js/app/usuario/vview.html', controller: 'usuarioViewController', resolve: {auth: autenticacionCualquiera}},
    $routeProvider.when('/usuario/new', {templateUrl: 'js/app/usuario/new.html', controller: 'usuarioNewController', resolve: {auth: autenticacionCualquiera}},
    $routeProvider.when('/usuario/edit/:id', {templateUrl: 'js/app/usuario/edit.html', controller: 'usuarioEditController', resolve: {auth: autenticacionCualquiera}},
    $routeProvider.when('/usuario/remove/:id', {templateUrl: 'js/app/usuario/remove.html', controller: 'usuarioRemoveController', resolve: {auth: autenticacionCualquiera}});

    //NOTICIAS
    $routeProvider.when('/noticias/edit/:id', {templateUrl: 'js/app/noticias/edit.html', controller: 'noticiasEditController', resolve: {auth: autenticacionCualquiera}});
    $routeProvider.when('/noticias/remove/:id?:page?', {templateUrl: 'js/app/noticias/remove.html', controller: 'noticiasRemoveController', resolve: {auth: autenticacionCualquiera}});
    $routeProvider.otherwise({redirectTo: '/', resolve: {auth: autenticacionCualquiera}});
}]);
```

El manejo de excepciones cuando un usuario intenta navegar a una ruta que no existe o a la que no tiene acceso, se hace escribiendo una función 'de lo contrario' ('otherwise'), para redirigir al usuario a la ruta '/' (raíz de la aplicación):

```
$routeProvider.when('/noticias/edit/:id', {templateUrl: 'js/app/noticias/edit.html', controller: 'noticiasEditController', resolve: {auth: autenticacionCualquiera}});
$routeProvider.when('/noticias/remove/:id?:page?', {templateUrl: 'js/app/noticias/remove.html', controller: 'noticiasRemoveController', resolve: {auth: autenticacionCualquiera}});
$routeProvider.otherwise({redirectTo: '/', resolve: {auth: autenticacionCualquiera}});
```

Seguridad

Dependiendo del usuario activo, se tendrá acceso a según qué partes de la aplicación.

La seguridad desde la aplicación cliente es llevada por el servicio de sesión ('sessionService'), creado a tal efecto.

Fragmento del servicio de sesión:

```
moduleService.service('sessionService', ['$location', function ($location) {
    var isSessionActive = false;
    var userName = "";
    var idUserLogged = "";
    var tipoUsuarioID = "";
    var sesion = "";

    return {
        getUserName: function () {
            return userName;
        },
        setId: function (id) {
            idUserLogged = id;
        },
        getId: function () {
            return idUserLogged;
        }
    };
}]);
```

El controlador de la vista de inicio de sesión, se encarga de hacer la petición al servidor, si se tiene éxito, el servidor envía los datos de usuario, que se almacenan en las variables del sessionService, si falla, se muestra un mensaje de error en la autenticación.

Fragmento del controlador de inicio de sesión:

```
$http({
    method: 'GET',
    header: {
        'Content-Type': 'application/json;charset=utf-8'
    },
    url: 'json?ob.usuario&op=login&user=' + login + '&pass=' + pass
}).then(function (response) {
    if (response.data.message.id !== 0) {

        $scope.logged = true;
        $scope.failedlogin = false;
        sessionService.setSessionActive();
        sessionService.setUserName(response.data.message.nombre + " " + response.data.message.apel);
        $scope.loggeduserid = sessionService.getUserName();
        $scope.loggeduserid = sessionService.setId(response.data.message.id);
        sessionService.setTypeUserID(response.data.message.obj_tipoUsuario.id);
        $location.url('/home');

    } else {
        $scope.failedlogin = true;
        if(response.data.message != null){
            $scope.error = ":" + response.data.message;
        }
    }
}, function (response) {
    $scope.failedlogin = true;
    $scope.logged = false;
    $scope.error = ":" + response.data.message;
});
```

Por otro lado desde el archivo de configuración de rutas, se comprueban las credenciales del usuario mediante promesas (promises), funciones que se ejecutan cuando se cumple una condición, en este caso, el tipo de usuario 'logueado'. Estas promesas, llaman al método check del servidor, que comprueba si hay un usuario en sesión. Hay 4 tipos de situaciones y por tanto 4 promesas repartidas en las diferentes rutas. Si la promesa no se cumple, no se puede acceder a esa ruta y, por tanto, se redirige al inicio de la aplicación, mediante el método anteriormente comentado 'otherwise'.

Fragmento del archivo de configuración de rutas:

```

var autenticacionAdministrador = function ($q, $location, $http, sessionService) {
    var deferred = $q.defer();
    $http({
        method: 'GET',
        url: 'json?ob=usuario&op=check'
    }).then(function (response) {
        if (response.data.message.obj_tipoUsuario.id === 1) {
            sessionService.setSession(response.data.message);
            sessionService.setSessionActive(response.data.message);
            sessionService.setUserName(response.data.message.nombre + " " + response.data.message.apel);
            sessionService.setId(response.data.message.id);
            sessionService.setTypeUserID(response.data.message.obj_tipoUsuario.id);
            deferred.resolve();
        } else {
            $location.path('/home');
        }
    }, function (response) {
        sessionService.setSessionInactive();
        $location.path('/home');
    });
    return deferred.promise;
}

var autenticacionUsuario = function ($q, $location, $http, sessionService) {...23 lines...}

var autenticacionCualquieraLogueado = function ($q, $location, $http, sessionService) {...23 lines...}

var autenticacionCualquiera = function ($q, $location, $http, sessionService) {...26 lines...}

cercahuerta.config(['$routeProvider', function ($routeProvider) {
    //HOME
    $routeProvider.when('/', {templateUrl: 'js/app/common/home.html', controller: 'homeController', resolve: {auth: autenticacionCualquiera}});
}])

```

En todos los casos, la contraseña de usuario se envía y almacena en la base de datos encriptada, por medio del script 'forge-sha256.min.js':

```

    pass: forge_sha256($scope.pass)
}

```

La contraseña nunca es enviada del servidor a la aplicación cliente; es enviada encriptada y se compara en la base de datos con la contraseña encriptada que está almacenada.

Proceso de petición de datos al servidor

Mediante una petición AJAX, se hace una llamada al servidor, indicando el método, la ruta y la información solicitada. Con la respuesta del servidor, se construye la vista con la información, o bien se envía un mensaje de error. Como la petición AJAX es asíncrona, el código restante se puede seguir ejecutando mientras se recibe la respuesta del servidor.

En el caso del listado de usuarios que se vio anteriormente, se solicita primero el número de registros, para preparar la paginación en caso de ser necesaria (la paginación solo aparece si hay más de una página). Después se solicitan los datos propiamente dichos.

Cada vez que utilizamos la paginación, solo la información pertinente y los controles de paginación se actualizan en la vista:

```
$http({
  method: 'GET',
  url: 'json?ob=' + $scope.ob + '&op=getcount'
}).then(function (response) {
  $scope.status = response.status;
  $scope.ajaxDataUsuariosNumber = response.data.message;
  $scope.totalPages = Math.ceil($scope.ajaxDataUsuariosNumber / $scope.rpp);
  if ($scope.page > $scope.totalPages) {
    $scope.page = $scope.totalPages;
    $scope.update();
  }
  pagination2();
}, function (response) {
  $scope.ajaxDataUsuariosNumber = response.data.message || 'Request failed';
  $scope.status = response.status;
});

$http({
  method: 'GET',
  url: 'json?ob=' + $scope.ob + '&op=getpage&rpp=' + $scope.rpp + '&page=' + $scope.page + $scope.orderURLServidor
}).then(function (response) {
  $scope.status = response.status;
  $scope.ajaxDataUsuarios = response.data.message;
}, function (response) {
  $scope.status = response.status;
  $scope.ajaxDataUsuarios = response.data.message || 'Request failed';
});

$scope.update = function () {
  $location.url($scope.ob + '/plist/' + $scope.rpp + '/' + $scope.page + '/' + $scope.orderURLCliente);
};
```

En el caso de envío de datos, como los de un formulario, se construye un 'JSON' con la información del formulario y se envía con la petición:

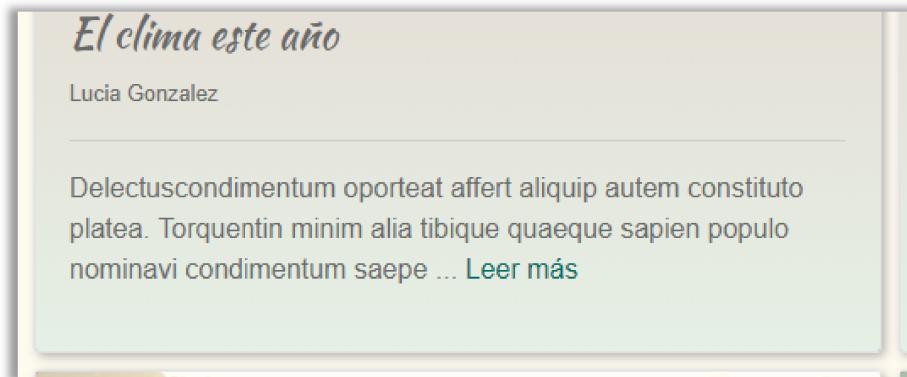
```
$scope.update = function () {

  var json = {
    id: null,
    dni: $scope.dni,
    nombre: $scope.nombre,
    apel: $scope.apel,
    ape2: $scope.ape2,
    email: $scope.email,
    login: $scope.login2,
    pass: forge_sha256($scope.pass),
    id_tipoUsuario: $scope.obj_tipousuario.id
  }
  $http({
    method: 'GET',
    header: {
      'Content-Type': 'application/json;charset=utf-8'
    },
    url: 'json?ob=usuario&op=create',
    params: {json: JSON.stringify(json)}
  }).then(function () {
    $scope.edited = false;
  });
};
```

Otras funciones de cliente

Filtro clipString

Creado para limitar el texto que aparece en las noticias cuando son listadas; para leer la noticia entera hay que entrar en ella.

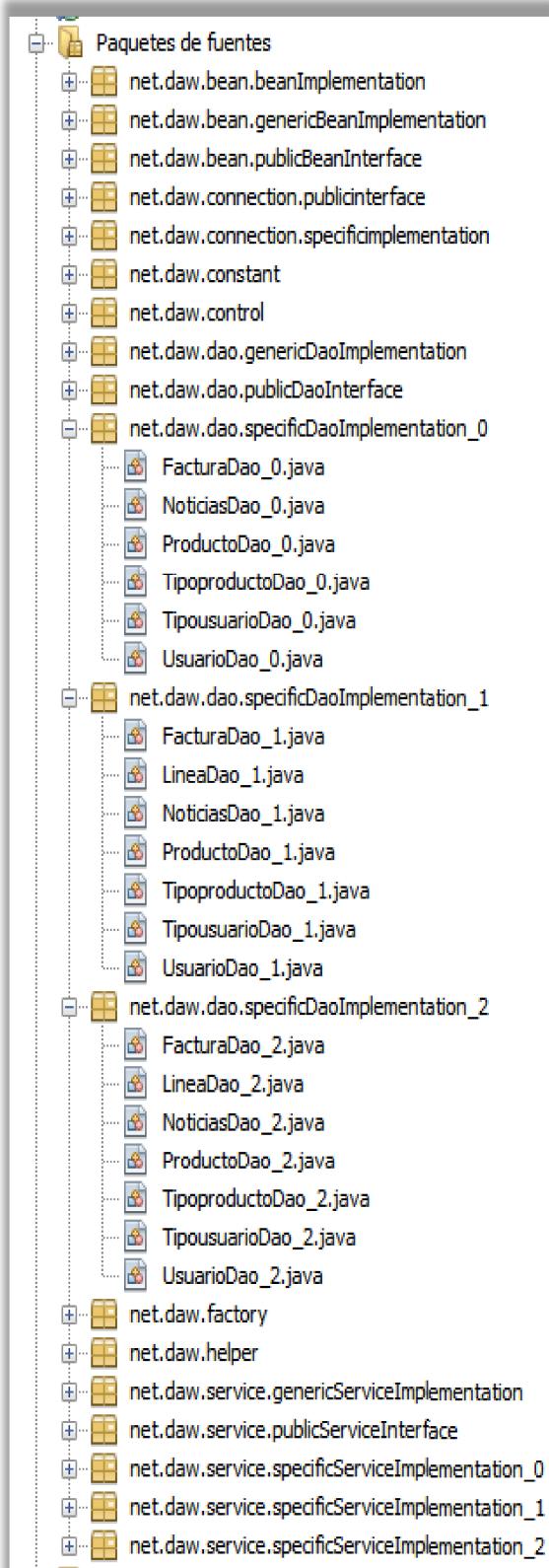


jsPDF (<https://github.com/MrRio/jsPDF>)

Se utiliza en la creación de PDFs para las facturas.

<h2>Factura 14</h2> <p>Cliente: Manuela Perez Gonzalez DNI: 98741236L Fecha: may 21, 2019</p>	<p>Cerca de la Huerta CIF. D29892858 555-125-563 info@cercahuerta.com</p> <p>Calle 11 Sector 2 PI Salper Ciudad Nueva</p>		
Codigo Nombre Cantidad Precio			
tom48	tomates de temporada	3	91.33
cal29	esparagos grandes	2	13.15
tom48	calabazas de temporada	4	70.89

Aplicación Servidor



Java EE

En el lado servidor, se ha utilizado Java EE (Java Enterprise Edition). Es un conjunto de estándares de tecnologías dedicadas al desarrollo de Java del lado del servidor. La plataforma Java EE consta de un conjunto de servicios, API y protocolos que proporcionan la funcionalidad necesaria para desarrollar aplicaciones basadas en web de varios niveles. Permite el desarrollo de aplicaciones distribuidas, con una arquitectura multicapa, escritas en Java y que se ejecutan en un servidor de aplicaciones.

Maven

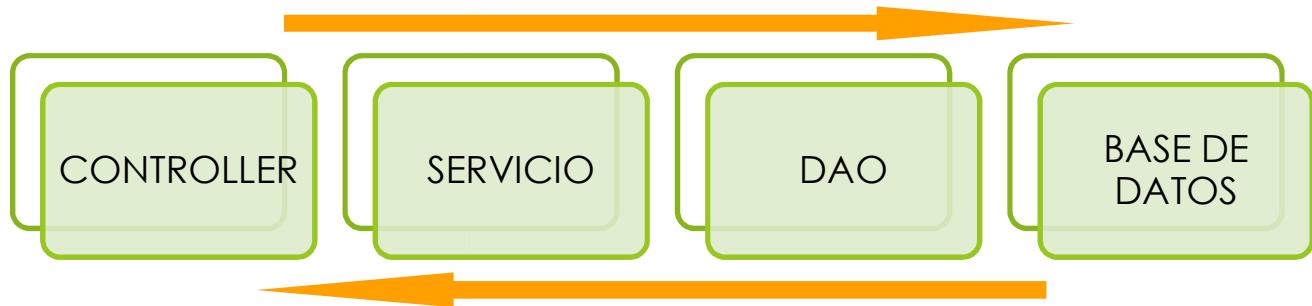
Se ha utilizado la herramienta de gestión de proyectos Maven. Se basa en un fichero central, pom.xml, donde se define todo lo que se necesita. Maneja las dependencias del proyecto, compila, empaqueta y ejecuta los test. Se definen tres ciclos de build del software con una serie de etapas diferenciadas:

- Validación (validate): Validar el proyecto.
- Compilación (compile).
- Test (test): Probar el código fuente.
- Empaquetar (package): Empaquetar el código compilado en algún formato tipo .jar o .war.
- Pruebas de integración (integration-test): Procesar y desplegar el código en algún entorno para ejecutar las pruebas de integración.
- Verificar que el código empaquetado es válido.
- Desplegar el código a un entorno (deploy).

Funcionalidad de la aplicación servidor

Estructura por capas

La aplicación de servidor, está dividida en las siguientes capas:

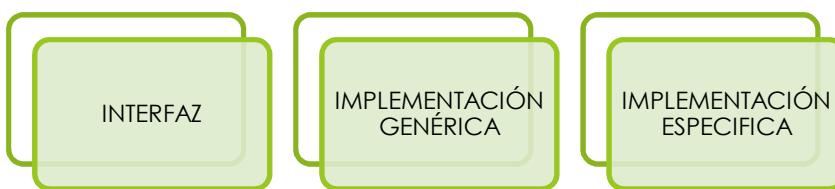


La capa controller, recibe el objeto request (petición) y envía el request a la capa service. Después construye el objeto response (respuesta) con la contestación de service (un objeto replayBean) y lo envía al cliente.

La capa service crea la conexión a la base de datos y envía la petición a la capa DAO. La respuesta de DAO es un bean (pojo). Éste es transformado en un objeto json e incluido en la respuesta a la capa controller junto con el status (código de estado).

La capa DAO ejecuta la consulta a la base de datos haciendo uso de la conexión creada en la capa service. Recibe el resultado de la query a la base de datos y devuelve en un bean o el array de beans como respuesta a la capa service o, en su caso, la excepción si no se ha podido llevar a cabo la operación.

Generalización



Se han generalizado todas las clases de la capa service, de la capa DAO y de los pojos o beans, de modo que los métodos y variables comunes, no es necesario implementarlos y/o modificarlos en todas ellas.

En primer lugar, se crea la interfaz que contiene las firmas de los métodos que se deben incluir en las clases que la implementen.

Fragmento de la interfaz de la capa service:

```
public interface ServiceInterface {
    public ReplyBean get() throws Exception;
    public ReplyBean remove() throws Exception;
    public ReplyBean getcount() throws Exception;
```

Después se crean las clases genéricas que implementan la interfaz, y por tanto sus métodos:

Fragmento de la clase genérica de la capa service:

```
public class GenericServiceImplementation implements ServiceInterface {
    protected HttpServletRequest oRequest;
    protected String ob = null;
    protected UsuarioBean oUsuarioBeanSession;

    public GenericServiceImplementation(HttpServletRequest oRequest) {
        super();
        this.oRequest = oRequest;
        ob = oRequest.getParameter("ob");
        //comprobar si falta el if para evitar NPE:
        oUsuarioBeanSession = (UsuarioBean) oRequest.getSession().getAttribute("user");
    }

    @Override
    public ReplyBean get() throws Exception {
        ReplyBean oReplyBean;
        ConnectionInterface oConnectionPool = null;
        Connection oConnection;
        try {
            Integer id = Integer.parseInt(oRequest.getParameter("id"));
            oConnectionPool = ConnectionFactory.getConnection(ConnectionConstants.connectionPool);
            oConnection = oConnectionPool.newConnection();
            DaoInterface oDao = DaoFactory.getDao(oConnection, ob, oUsuarioBeanSession);
            BeanInterface oBean = oDao.get(id, 2);
            Gson oGson = (new GsonBuilder()).excludeFieldsWithoutExposeAnnotation().create();
            oReplyBean = new ReplyBean(200, oGson.toJson(oBean));
        } catch (Exception ex) {
            throw new Exception("ERROR: Service level: get method: " + ob + " object", ex);
        } finally {
            oConnectionPool.disposeConnection();
        }
        return oReplyBean;
    }
}
```

Por último, se crean las clases específicas necesarias, que extienden la clase genérica e implementan la interfaz. En estas clases, solo se desarrollan los métodos que no son comunes a todas ellas o aquellos que deben ser modificados en esa clase en concreto.

Fragmento de la clase específica de usuario de la capa service:

```
public class UsuarioService_1 extends GenericServiceImplementation implements ServiceInterface {

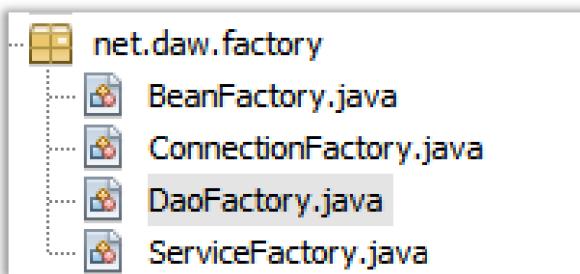
    public UsuarioService_1(HttpServletRequest oRequest) {
        super(oRequest);
        ob = oRequest.getParameter("ob");
    }

    public ReplyBean logout() throws Exception {
        oRequest.getSession().invalidate();
        return new ReplyBean(200, EncodingHelper.quoteat("OK"));
    }

    public ReplyBean check() throws Exception {
        ReplyBean oReplyBean;
        //Aquí no haría falta el usuarioBean de session, ya lo cogemos del generic, pero lo dejo por
        UsuarioBean oUsuarioBean;
        oUsuarioBean = (UsuarioBean) oRequest.getSession().getAttribute("user");
        if (oUsuarioBean != null) {
            Gson oGson = (new GsonBuilder()).excludeFieldsWithoutExposeAnnotation().create();
            oReplyBean = new ReplyBean(200, oGson.toJson(oUsuarioBean));
        } else {
            oReplyBean = new ReplyBean(401, "No active session");
        }
        return oReplyBean;
    }

    public ReplyBean updatePass() throws Exception {
        int iRes = 0;
        ReplyBean oReplyBean;
```

Factorías



Desde las factorías se instancian los nuevos objetos (los news de todas las clases), de modo que al llamarlos, se llama a la factoría y se le pasan los parámetros necesarios que indican el tipo de objeto y la operación a realizar.

Llamada a las factorías de beans y DAOs desde service:

```
DaoInterface oDao = DaoFactory.getDao(oConnection, ob, oUsuarioBeanSession);
BeanInterface oBean = oDao.get(id, 2);
```

Factoría de la capa DAO:

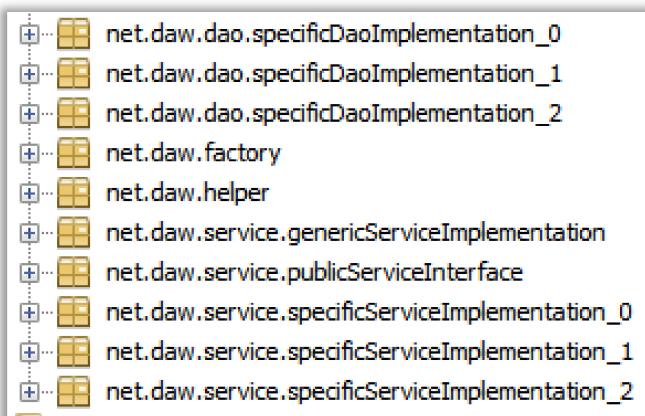
```
public class DaoFactory {
    public static DaoInterface getDao(Connection oConnection, String ob, UsuarioBean oUsuarioBeanSession) throws Exception {
        DaoInterface oDao = null;
        int idSessionUserType;
        if (oUsuarioBeanSession != null) {
            idSessionUserType = oUsuarioBeanSession.getObj_tipoUsuario().getId();
        } else {
            idSessionUserType = 0;
        }

        switch (idSessionUserType) {
            case 1:
                switch (ob) {
                    case "usuario":
                        oDao = new UsuarioDao_1(oConnection, ob, oUsuarioBeanSession);
                        break;
                    case "tipousuario":
                        oDao = new TipousuarioDao_1(oConnection, ob, oUsuarioBeanSession);
                        break;
                    case "tipopropuesto":
                        oDao = new TipopropuestoDao_1(oConnection, ob, oUsuarioBeanSession);
                        break;
                    case "producto":
                        oDao = new ProductoDao_1(oConnection, ob, oUsuarioBeanSession);
                        break;
                    case "factura":
                        oDao = new FacturaDao_1(oConnection, ob, oUsuarioBeanSession);
                        break;
                    case "linea":
                        oDao = new LineaDao_1(oConnection, ob, oUsuarioBeanSession);
                        break;
                    case "noticias":
                        oDao = new NoticiasDao_1(oConnection, ob, oUsuarioBeanSession);
                        break;
                }
                break;
        }
    }
}
```

Seguridad

Perfiles de usuario

Tanto la generalización como las factorías, tienen una utilidad más; facilitar el manejo de perfiles de usuario. Al igual que en la parte de cliente, en el servidor los usuarios tendrán permitidas una serie de acciones, según el tipo de usuario que sean. Los usuarios son administradores (1), usuarios (2), y no registrados/activados (0).



Para este fin, se han creado diferentes implementaciones de las clases genéricas para cada tipo de usuario, en las que se restringe el uso de según qué métodos dependiendo de lo que tenga permitido hacer el usuario que hace la petición.

El usuario administrador (1) tiene todos los permisos, por lo que sus clases solo contienen los métodos específicos, sin tener que modificar los de la clase genérica.

Método 'remove' en la clase genérica:

```
@Override
public int remove(int id) throws Exception {
    int iRes = 0;
    PreparedStatement oPreparedStatement = null;
    try {
        oPreparedStatement = oConnection.prepareStatement(strSQL_remove);
        oPreparedStatement.setInt(1, id);
        iRes = oPreparedStatement.executeUpdate();
    } catch (SQLException e) {
        throw new Exception("Error en Dao remove de " + ob, e);
    } finally {
        if (oPreparedStatement != null) {
            oPreparedStatement.close();
        }
    }
    return iRes;
}
```

Método 'remove' en la clase usuarioDAO_2 (usuario no administrador):

```
@Override
public int remove(int id) throws Exception {
    throw new Exception("Error en Dao remove de " + ob + ": No autorizado");
}
```

Al hacer cualquier petición desde el cliente, en el request viajan los datos de sesión y, en ellos, los datos del usuario activo.

Lo primero que hacen las factorías, tanto de DAO como de service, es comprobar si hay un usuario en sesión y qué tipo de usuario es. Dependiendo del tipo de usuario, tendrá acceso a las clases correspondientes:

```
public class DaoFactory {

    public static DaoInterface getDao(Connection oConnection, String ob, UsuarioBean oUsuarioBeanSession) throws Exception {
        DaoInterface oDao = null;
        int idSessionUserType;
        if (oUsuarioBeanSession != null) {
            idSessionUserType = oUsuarioBeanSession.getObj_tipoUsuario().getId();
        } else {
            idSessionUserType = 0;
        }

        switch (idSessionUserType) {
            case 1:
                switch (ob) {
                    case "usuario":
                        oDao = new UsuarioDao_1(oConnection, ob, oUsuarioBeanSession);
                        break;
                }
        }
    }
}
```

Petición a la base de datos

Los datos que llegan del cliente, generalmente de formularios, son recogidos en las clases especiales para este propósito, los beans o pojos, por lo que esto supone un primer filtro, ya que los datos deben coincidir con el tipo de variables de estos pojos:

```
public class NoticiasBean extends GenericBeanImplementation implements BeanInterface {

    @Expose
    private String titulo;
    @Expose
    private String mensaje;
    @Expose
    private String foto;
    @Expose(serialize = false)
    private int id_usuario;
    @Expose(deserialize = false)
    private UsuarioBean obj_usuario;

    public String getTitulo() {
```

Las 'queries' por su parte, se construyen mediante métodos en los propios beans y sentencias preparadas (prerestedStatements), lo que ayuda a evitar inyecciones SQL.

Algunos métodos de los beans:

```
@Override
public String getPairs() {
    String strPairs = "";
    strPairs += "titulo=" + EncodingHelper.quoteatetitulo) + ",";
    strPairs += "mensaje=" + EncodingHelper.quoteatemensaje) + ",";
    strPairs += "foto=" + EncodingHelper.quoteatefoto) + ",";
    strPairs += "id_usuario=" + id_usuario;
    strPairs += " WHERE id=" + id;
    return strPairs;
}
```

```
@Override
public String getColumns() {
    String strColumns = "";
    strColumns += "id,";
    strColumns += "titulo,";
    strColumns += "mensaje,";
    strColumns += "foto,";
    strColumns += "id_usuario";
    return strColumns;
}
```

Consulta a base de datos con preparedStatement:

```
try {
    oPreparedStatement = oConnection.prepareStatement(strSQL_get);
    oPreparedStatement.setInt(1, id);
    oResultSet = oPreparedStatement.executeQuery();
```

Proceso de alta

En esta sección de la memoria, se explica el proceso desde que un usuario entra por primera vez a la aplicación hasta que se da de alta.

Cuando un usuario envía el formulario de alta a la aplicación servidor, la primera operación es comprobar el tipo de usuario (como se vio anteriormente). Solo un usuario sin sesión activa (de tipo '0') puede utilizar este proceso (los administradores, desde luego, pueden dar de alta usuarios, pero utilizando otro proceso).

En el método de creación de usuario de la clase usuarioDAO_0, se comprueba que el tipo de usuario a crear es '0', de lo contrario, se rechaza la operación. Si se crea correctamente, se crea un 'token' que servirá para identificar al usuario y se le envía un email de activación con dicho token y el enlace de activación.

Método 'create' en la clase usuarioDAO_0:

```

@Override
public BeanInterface create(BeanInterface oBean) throws Exception {
    UsuarioBean oUsuarioBeancontrol=(UsuarioBean) oBean;
    UsuarioBean oBeanActivation;
    int idtipousuario = oUsuarioBeancontrol.getId_tipoUsuario();
    if (idtipousuario == 0) {
        try {
            oBeanActivation = (UsuarioBean) super.create(oBean);
            String email = oBeanActivation.getEmail();
            String nombre = oBeanActivation.getNombre();
            UsuarioBean oBeanToken = (UsuarioBean) super.get(oBeanActivation.getId(), 1);
            String token = oBeanToken.getToken();
            UserActivationEmail.sendActivationEmail(email, nombre, token);
        } catch (Exception ex) {
            throw new Exception("Error en Dao create de " + ob + ": " + ex.getMessage(), ex);
        }
    }
    return oBeanActivation;
} else {
    throw new Exception("Error en Dao update de " + ob + ": No autorizado");
}
}

```

Generador de tokens:

```

public class TokenGenerator {
    public static final String CHARACTERS = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    public static final int SECURE_TOKEN_LENGTH = 256;
    private static final SecureRandom random = new SecureRandom();
    private static final char[] symbols = CHARACTERS.toCharArray();
    private static final char[] buf = new char[SECURE_TOKEN_LENGTH];

    public static String nextToken() {
        for (int idx = 0; idx < buf.length; ++idx) {
            buf[idx] = symbols[random.nextInt(symbols.length)];
        }
        return new String(buf);
    }
}

```

Método para enviar email de activación de la clase UserActivationEmail.java:

```

public static String sendActivationEmail(String email, String nombre, String token) throws Exception {
    String to = email;
    String from = "info.cercahuerta@gmail.com";
    String pass = "cercahuerta2019";
    String link = "http://localhost:8081/cercahuerta/json?ob=usuarioc&op=activar&token=" + token;
    String linkTexto = "http://localhost:8081/cercahuerta/json?ob=usuarioc&op=activar&token=" + token;
    Properties properties = new Properties();
    properties.setProperty("mail.smtp.host", "smtp.gmail.com");
    properties.setProperty("mail.smtp.starttls.enable", "true");
    properties.setProperty("mail.smtp.port", "587");
    properties.setProperty("mail.smtp.user", from);
    properties.setProperty("mail.smtp.auth", "true");
    Session session = Session.getDefaultInstance(properties);

    try {
        MimeMessage message = new MimeMessage(session);
        message.setFrom(new InternetAddress(from));
        message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));

        message.setSubject("confirme su alta en Cerca de la Huerta");
        message.setText("Bienvenido a Cerca de la Huerta " + nombre
                + ",<br> Haz click en <a href='" + link + "'>este enlace</a> para confirmar tu cuenta"
                + "<br> Si el link no funciona o si lo prefiere, pegue esto en su navegador:"
                + "<br>" + linkTexto
                + "<br><br><small>Mensaje automático; por favor, no responda este correo</small>",
                "utf-8", "html");
        Transport t = session.getTransport("smtp");
        t.connect(from, pass);
        t.sendMessage(message, message.getAllRecipients());
        t.close();
        respuesta = "Mensaje enviado con éxito";
    } catch (MessagingException mex) {
        respuesta = "Ha habido un error al enviar el mensaje: " + mex.getMessage();
        throw new Exception("Error en UserActivationEmail sendActivationEmail: " + mex.getMessage(), mex);
    }
    return respuesta;
}

```

En ese momento, el usuario está en la base de datos, aunque con los campos 'activo' e 'id_tipousuario' con el valor '0'.

pass	id_tipousuario	token	activo
8d969eefbe cad3c29a3a629280e686cf0c3f5d5a86aff3ca12...	0	MeaOByS80rGA0rV6K5tVoWujbc7nhB9HzM5CAoevUsjRIUnHVq...	0

Una vez el usuario recibe el email y accede al enlace de activación, se ejecuta el método 'activar' y desde este los métodos que comprueban que el token está en la base de datos y que finalmente activan al usuario, dejando el campo 'activo' en '1' y el tipo de usuario en '2'.

Método activar:

```
public Integer activar(String token) throws Exception {
    int resultadoActivacion = 0;
    UsuarioBean oUsuarioBean = (UsuarioBean) this.comprobarToken(token, 2);
    if (oUsuarioBean.getToken().equals(token)) {
        resultadoActivacion = this.activarUsuario(token);
        UserActivationEmail.sendConfirmationEmail(oUsuarioBean.getEmail(), oUsuarioBean.getNombre());
    }
    return resultadoActivacion;
}
```

Método de comprobación de token:

```
public BeanInterface comprobarToken(String token, Integer expand) throws Exception {
    strSQL_get = "SELECT * FROM " + ob + " WHERE token=?";
    BeanInterface oBean;
    ResultSet oResultSet = null;
    PreparedStatement oPreparedStatement = null;
    try {
        oPreparedStatement = oConnection.prepareStatement(strSQL_get);
        oPreparedStatement.setString(1, token);
        oResultSet = oPreparedStatement.executeQuery();
        if (oResultSet.next()) {
            oBean = BeanFactory.getBean(ob);
            oBean.fill(oResultSet, oConnection, expand, oUsuarioBeanSession);
        } else {
            oBean = null;
        }
    } catch (SQLException e) {
        throw new Exception("Error en Dao get de " + ob, e);
    } finally {
        if (oResultSet != null) {
            oResultSet.close();
        }
        if (oPreparedStatement != null) {
            oPreparedStatement.close();
        }
    }
    return oBean;
}
```

Método de activación de usuario:

```
public int activarUsuario(String token) throws Exception {
    int iResult = 0;
    strSQL_update = "UPDATE " + ob + " SET ";
    strSQL_update += "id_tipoUsuario=2,activo=1 ";
    strSQL_update += "WHERE token=?";
    PreparedStatement oPreparedStatement = null;
    try {
        oPreparedStatement = oConnection.prepareStatement(strSQL_update);
        oPreparedStatement.setString(1, token);
        iResult = oPreparedStatement.executeUpdate();

    } catch (SQLException e) {
        throw new Exception("Error en Dao activarUsuario de " + ob + ": " + e.getMessage(), e);
    } finally {
        if (oPreparedStatement != null) {
            oPreparedStatement.close();
        }
    }
    return iResult;
}
```

Si todo va bien, el método activar envía un segundo email al usuario, informándole de que su alta se ha hecho efectiva:

```
public static String sendConfirmationEmail(String email, String nombre) throws Exception {
    String to = email;
    String from = "info.cercahuerta@gmail.com";
    String pass = "cercahuerta2019";

    Properties properties = new Properties();
    properties.setProperty("mail.smtp.host", "smtp.gmail.com");
    properties.setProperty("mail.smtp.starttls.enable", "true");
    properties.setProperty("mail.smtp.port", "587");
    properties.setProperty("mail.smtp.user", from);
    properties.setProperty("mail.smtp.auth", "true");
    Session session = Session.getDefaultInstance(properties);
    try {
        MimeMessage message = new MimeMessage(session);
        message.setFrom(new InternetAddress(from));
        message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
        message.setSubject("Su cuenta en Cerca de la Huerta ha sido confirmada");
        message.setText("Enhorabuena " + nombre
                + "<br> Haz click en <a href='http://localhost:8081/cercahuerta/'>este enlace</a> para entrar en Cerca de la Huerta"
                + "<br> Si el link no funciona o si lo prefiere, pegue esto en su navegador:"
                + "<br>http://localhost:8081/cercahuerta/"
                + "<br>Ya puedes empezar a disfrutar de tus ventajas como usuario registrado"
                + "<br><br><small>Mensaje automático; por favor, no responda este correo</small>",
                "utf-8", "html");
        Transport t = session.getTransport("smtp");
        t.connect(from, pass);
        t.sendMessage(message, message.getAllRecipients());
        t.close();
        respuesta = "Mensaje enviado con éxito";
    } catch (MessagingException mex) {
        respuesta = "Ha habido un error al enviar el mensaje: " + mex.getMessage();
        throw new Exception("Error en UserActivationEmail sendConfirmationEmail: " + mex.getMessage(), mex);
    }
    return respuesta;
}
```

Otras funciones de servidor

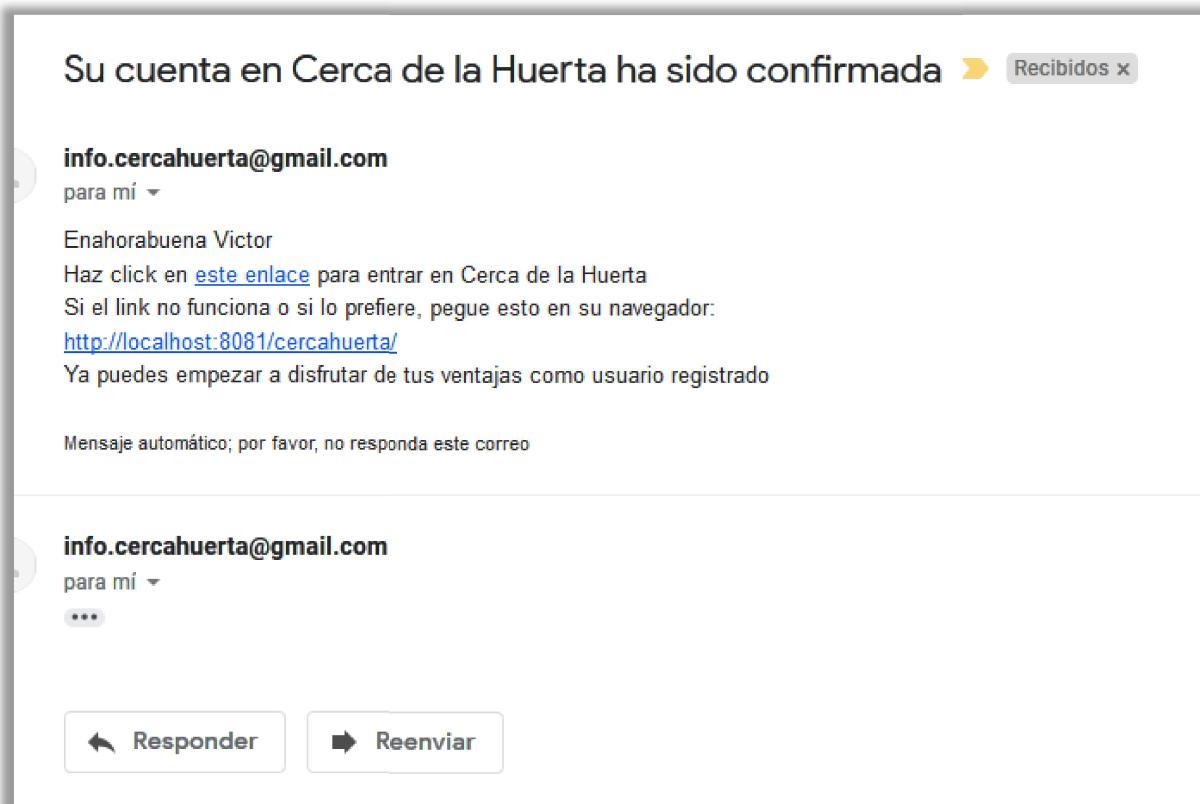
Lorem Ipsum generator (<https://github.com/mdeanda/lorem>)

Se ha utilizado para generar textos aleatorios para las noticias creadas automáticamente para el desarrollo de la aplicación. Dependiendo de su configuración, crea un texto de longitud o número de párrafos aleatorios:

```
String[] titulos = {"la huerta", "el campo",  
String[] titulo2 = {"de hoy", "este año", "mejor", "novedad", "destacado"};  
String mensaje = lorem.getParagraphs(3, 4);  
String[] foto = {"noticias1.jpg", "noticias2.jpg", "noticias3.jpg", "noticias4.jpg"};
```

JavaMail (<https://javaee.github.io/javamail/>)

Utilizado para el envío de emails en el proceso de activación de una cuenta de usuario.



Futuras ampliaciones y mejoras

Cerca de la Huerta, a pesar de tener la funcionalidad completa para el objetivo principal, que es dar a conocer la huerta y sus productos, así como posibilitar su comercialización, presenta la oportunidad de las siguientes ampliaciones y mejoras:

Sistema para visitas a la huerta

En un principio se pensó en crear un sistema para publicitar visitas a la huerta, con posibilidad de comer allí, recoger uno mismo los productos, charlas... Pero hay un impedimento; Los posibles clientes y sobre todo, los productores, no estaban muy dispuestos a participar en esos eventos. Tal vez iniciada la actividad y con el tiempo, se pueda sondear de nuevo a los participantes y ver si se puede llevar a cabo.

Mantenimiento

La aplicación ha sido probada, no obstante, con el uso real puede que surjan tareas para mejorar o ampliar características, por lo que se hará un seguimiento del funcionamiento y se recopilarán datos de los usuarios para este fin.

Bibliografía y enlaces

<https://stackoverflow.com/>

<https://developer.mozilla.org/en-US/>

<https://css-tricks.com/>

<https://www.youtube.com/channel/UCF6G2sF4DKIWvgfEGcm50FQ/playlists>

<https://www.youtube.com/user/0utKast/playlists>

<https://www.tutorialspoint.com/index.htm>

<https://www.codecademy.com/>

<http://www.javahispano.org/>

<https://openlibra.com/es>

<https://uniwebsidad.com/>

Licencia

"Los autores de los proyectos tienen plena disposición y derecho exclusivo a la explotación del proyecto presentado, sin más limitaciones que las contenidas en el Real Decreto Legislativo 1/1996, de 12 de abril en materia de propiedad industrial e intelectual, sin perjuicio de lo cual deberá ceder al centro el proyecto a efectos académicos"

(Extraído de http://www.docv.gva.es/datos/2012/07/26/pdf/2012_7455.pdf)

La licencia del proyecto es MIT, que traducida viene a decir lo siguiente:

Se concede permiso por la presente, libre de cargos, a cualquier persona que obtenga una copia de este software y de los archivos de documentación asociados (el "Software"), a utilizar el Software sin restricción, incluyendo sin limitación los derechos a usar, copiar, modificar, fusionar, publicar, distribuir, sublicenciar, y/o vender copias del Software, y a permitir a las personas a las que se les proporcione el Software a hacer lo mismo, sujeto a las siguientes condiciones:

El aviso de copyright anterior y este aviso de permiso se incluirán en todas las copias o partes sustanciales del Software.

EL SOFTWARE SE PROPORCIONA "COMO ESTÁ", SIN GARANTÍA DE NINGÚN TIPO, EXPRESA O IMPLÍCITA, INCLUYENDO PERO NO LIMITADO A GARANTÍAS DE COMERCIALIZACIÓN, IDONEIDAD PARA UN PROPÓSITO PARTICULAR E INCUMPLIMIENTO. EN NINGÚN CASO LOS AUTORES O PROPIETARIOS DE LOS DERECHOS DE AUTOR SERÁN RESPONSABLES DE NINGUNA RECLAMACIÓN, DAÑOS U OTRAS RESPONSABILIDADES, YA SEA EN UNA ACCIÓN DE CONTRATO, AGRAVIO O CUALQUIER OTRO MOTIVO, DERIVADAS DE, FUERA DE O EN CONEXIÓN CON EL SOFTWARE O SU USO U OTRO TIPO DE ACCIONES EN EL SOFTWARE.

<https://opensource.org/licenses/MIT>

Agradecimientos

Este proyecto, no hubiera sido posible sin contar con la dedicación profesional y personal de mis profesores/as, la colaboración sincera y desinteresada de mis compañeros, y la organización y esfuerzo del CIPFP Ausiàs March en general.