

# PROGRAMACION WEB

## PEC\_FINAL

# Lista de la compra, aplicación web

En esta práctica he programado una aplicación web desde la que se gestionara la lista de la compra. Se pueden añadir productos, guardar listas, consultar listas por fecha de creación...

Para acceder a la aplicación se deberá estar registrado, si aún no estas registrado puedes ir a la sección de registro y crear un usuario nuevo.

La programación de la página web la he llevado a cabo dividiendo las tareas en varias fases

## 1. Primera fase

Lo primero que he realizado es crear las carpetas y archivos necesarios para la web.

- **Carpeta css:** Contiene un archivo style.css
- **Carpeta img:** Contiene las imágenes de los productos usados en la web
- **Carpeta js:** Contiene los archivos .js
- **Raiz:** Archivos .html

Una vez creadas las carpetas y archivos, el primer paso es crear las clases “**usuario**”, “**producto**” y “**lista**” con sus propiedades, que serán usadas más tarde para el funcionamiento de la web.

### Clase usuario (usuario.js)

```
class Usuario {
  constructor(nombre, apellidos, direccion, poblacion, codigoPostal,
telefono, correoElectronico, usuario, contraseña) {
    this.nombre = nombre;
    this.apellidos = apellidos;
    this.direccion = direccion;
    this.poblacion = poblacion;
    this.codigoPostal = codigoPostal;
    this.telefono = telefono;
    this.correoElectronico = correoElectronico;
    this.usuario = usuario;
    this.contraseña = contraseña;
  }
}
```

### Clase producto (products.js)

```
class Producto {
  constructor(nombre, tipo, enlace){
    this.nombre = nombre;
    this.tipo = tipo;
    this.enlace = enlace;
  }
}
```

### Clase lista (list.js)

```
constructor(usuario, fecha, productos){
  this.usuario = usuario;
  this.fecha = fecha;
  this.productos = productos;
}
```

Una vez creadas las clases, se añaden también los métodos setter y getter para cada atributo. Como ejemplo:

#### Método Get

```
getUsuario() {
  return this.usuario;
}
```

#### Método Set

```
setUsuario(usuario) {
  this.usuario = usuario;
}
```

En esta fase también he implementado la estructura de todas las páginas .html, añadiendo los botones o campos de textos necesarios para una primera versión.

## 2. Segunda fase

En esta segunda fase he trabajado en la página de login y la página de registro.

**Página de login (index.html):** en el html, se ha incluido un “form” compuesto por los inputs de texto para ingresar el nombre de usuario y contraseña y dos botones, “aceptar”, para avanzar si se ha introducido bien el usuario y la contraseña y “nuevo usuario” si es necesario registrarse en la aplicación.

En la página index.html, también he añadido un <script> para gestionar el inicio de sesión. Lo que se hace en ese fragmento de código es, buscar en el localStorage el nombre de usuario, si existe se compara la contraseña ingresada con la almacenada, si toda cuadra se avanza hasta la página “products.html”. Si no hay coincidencias con usuario o contraseña se mostraran los errores correspondientes.

```

<script>
//LOGIN
document.getElementById("loginForm").addEventListener("submit",
function(event){
    event.preventDefault();

    // Obtiene userName u contraseña introducidos
    let username = document.getElementById("username").value;
    let password = document.getElementById("password").value;

    // Obtiene los datos del usuario guardados en el localStorage
    let userStored = localStorage.getItem(username);
    if(userStored){

        //Convertir los datos de JSON a un objeto JavaScript
        let user = JSON.parse(userStored);

        // Comprueba si la contraseña ingresada coincide con la guardada en el
localStorage
        if(user.contraseña === password){
            Usuario.serialize(user);

            // Si la contraseña es correcta, navegamos a las siguiente pagina
            window.location.href = "products.html";
        } else {
            // Contraseña incorrecta
            alert("Contraseña incorrecta. Por favor, inténtalo de nuevo.");
        }
    } else {
        // Si no existe el usuario
        alert("No existe una cuenta con ese nombre de usuario. Por favor,
regístrate primero.");
    }
});
</script>

```

En cuanto a user.js, llamado desde index.html, además de definir la clase “usuario” y sus métodos get y set, también se han definido los métodos serialize, deserialize, retrieveUser, guardar usuario y recuperar usuario.

```

//Coge una matriz de usuarios y la almacena en el localStorage. Convierte en
una cadena JSON
static serialize(usersArray){
    localStorage.setItem('users', JSON.stringify(usersArray));
}

//Recupera la cadena JSON de usuarios, la convierte de nuevo en una matriz
de objetos y mapea cada objeto a una
//nueva instancia de Usuario
static deserialize() {
    let usersArray = JSON.parse(localStorage.getItem('users'));

```

```

        if (usersArray) {
            return usersArray.map(userObj => new Usuario(userObj.nombre,
userObj.apellidos, userObj.direccion, userObj.poblacion, userObj.codigoPostal,
userObj.telefono, userObj.correoElectronico, userObj.usuario,
userObj.contraseña));
        }
        return [];
    }

    //Llama a deserialize
    static retrieveUser() {
        return Usuario.deserialize();
    }

    //Guardar una instancia especifica del Usuario en el localStorage
    guardarUsuario() {
        Usuario.serialize(this);
    }

    //recupera los datos de un usuario
    recuperarUsuario() {
        let usuarioGuardado = localStorage.getItem(this.usuario);
        if(usuarioGuardado) {
            let usuario = JSON.parse(usuarioGuardado);
            this.nombre = usuario.nombre;
            this.apellidos = usuario.apellidos;
            this.direccion = usuario.direccion;
            this.poblacion = usuario.poblacion;
            this.codigoPostal = usuario.codigoPostal;
            this.telefono = usuario.telefono;
            this.correoElectronico = usuario.correoElectronico;
            this.usuario = usuario.usuario;
            this.contraseña = usuario.contraseña;
        }
    }
}

```

**Página de registro en la web (register.html):** En esta página se ha incluido el <form> con todos los inputs necesarios para el registro de un usuario, también se añaden dos botones “registrarse” si todos los campos son correctos el usuario se registrará en la página y se navegará a la siguiente, y “salir” con el que regresaremos a la pantalla de login. Desde esta página se llama a “user.js”, explicado en el punto anterior y a “register.js”

Desde “register.js” lo primero que he incluido es la parte encargada de asignar un código postal a una población (*he puesto códigos postales al azar a cada población*). Funciona de tal manera que cuando seleccionas una población automáticamente se introduce su código postal. Si se modifica el código postal, a la hora de registrarse saldrá una alerta advirtiéndote que población y código postal no corresponden.

```

//Poblaciones y códigos postales
var poblacionesCodigosPostales = {
  "Seleccione una población": "",
  "Burgos": "09007",
  "Soria": "09001",
  "Barcelona": "09002",
  "Vitoria": "09003",
  "Zaragoza": "09004",
  "Susinos del Paramo": "09133",
  "Sevilla": "09006",
  "Malaga": "09010",
  "Caceres": "09008",
  "Teruel": "09009",
};

//Rellenar el select de poblaciones
var selectPoblacion = document.getElementById('poblacion');
for (var poblacion in poblacionesCodigosPostales) {
  var option = document.createElement('option');
  option.text = poblacion;
  option.value = poblacion;
  selectPoblacion.add(option);
}

// Guardo "codigoPostal" en la variable inputCodigoPostal
var inputCodigoPostal = document.getElementById('codigoPostal');

// Controlador de eventos para cuando se cambia la selección en el select de
poblaciones
selectPoblacion.addEventListener('change', function () {

  // Buscar el código postal para la población seleccionada
  var codigoPostal = poblacionesCodigosPostales[this.value];

  // Actualizar el valor del elemento input del código postal
  inputCodigoPostal.value = codigoPostal;
});

```

La siguiente parte dentro de “register.js”, se encarga de recoger los valores introducidos por el usuario en el formulario de registro, crear una nueva instancia del objeto “usuario” y convertir el objeto a formato JSON para almacenarlo en el localStorage con la clave igual al nombre de usuario.

```

//Recoger los valores introducidos por el usuario en el formulario de registro
document.getElementById("registerButton").addEventListener("click", function
(event) {
  event.preventDefault();

  let u = document.getElementById("Usuario").value;
  let contraseña = document.getElementById("contraseña").value;
  let nombre = document.getElementById("nombre").value;

```

```

let apellidos = document.getElementById("apellidos").value;
let direccion = document.getElementById("direccion").value;
let poblacion = document.getElementById("poblacion").value;
let codigoPostal = document.getElementById("codigoPostal").value;
let telefono = document.getElementById("telefono").value;
let correoElectronico =
document.getElementById("correoElectronico").value;

// Crea un objeto de usuario con los datos ingresados
let user = new Usuario(nombre, apellidos, direccion, poblacion,
codigoPostal, telefono, correoElectronico, u, contraseña);

// Guarda el objeto de usuario en el localStorage
// Los objetos deben ser convertidos a formato JSON para poder ser
guardados en el localStorage
localStorage.setItem(u, JSON.stringify(user));

```

Por último, el código encargado de las validaciones de los datos introducido por el usuario. Que no estén vacíos los campos obligatorios, formato del teléfono o email, etc.

```

// Validaciones de los campos de el formulario de registro

// Campos que son obligatorios, si alguno de ellos esta vacio se muestra
un mensaje.
if (nombre === '' || apellidos === '' || direccion === '' || u === '') {
    alert('Nombre, Apellidos, Dirección y Usuario son campos
obligatorios');
    return;
}

//Comprobar que la poblacion coincide con su codigo postal (CP simulados,
no son reales)
if (poblacionesCodigosPostales[poblacion] !== codigoPostal) {
    alert('La población y el código postal no coinciden.');
```

```

    return;
}

//Expresion regular que comprueba si el formato del numero de telefono es
correcto.
//Estos son los formatos soportados:
// +1-800-123-4567
// (1) 800 123 4567
// 18001234567
// 800.123.4567
if (!/^[\+]?([0-9]{1,4})?[-\s\.]?[0-9]{1,4}[-\s\.]?[0-9]{4,6}$/.test(telefono)) {
    alert('El teléfono proporcionado no es válido.');
```

```

        return;
    }

    //Expresion regular que comprueba si el formato del email introducido es
correcto
    if (!/\S+@\S+\.\S+/.test(correoElectronico)) {
        alert('El correo electrónico proporcionado no es válido.');
```

```

        return;
    }

    //Comprobar si el usuario ya existe en el localStorage
    let usuariosExistentes = Usuario.deserialize();

    if (usuariosExistentes.some(user => user.usuario === u)) {
        alert('El nombre de usuario ya está en uso.');
```

```

        return;
    }

    // Agrega el nuevo usuario a la lista
    usuariosExistentes.push(user);

    // Guarda la lista actualizada en localStorage
    Usuario.serialize(usuariosExistentes);

    //Expresion regular que comprueba si la contraseña introducida cumple con
las condiciones
    if (!/^(?=.*[A-Za-z])(?=.*\d)(?=.*[@$!%*#?&])[A-Za-
z\d@$!%*#?&]{8,}$/ .test(contraseña)) {
        alert('La contraseña debe contener mínimo 8 caracteres, letras,
números y al menos dos caracteres especiales.');
```

```

        return;
    }

    // Redireccionar a la página de login
    window.location.href = "index.html";

```

### 3. Tercera fase

En esta fase he construido la zona de la web en la que seleccionaremos productos para añadirlos a la lista de la compra, la podremos guardar, mostrar, imprimir, ver un listado de las listas guardadas por fecha.

**Products.html:** Lo primero que he realizado en esta página es crear los contenedores en los sé alojaran de forma dinámica los contenidos. Por ejemplo, categorías, productos y el histórico de listas. En esta página solo estar el contenedor a que estos datos se añadirán dinámicamente desde “products.js”.



También he añadido cuatro botones para gestionar las necesidades de esta página.

- **guardar:** guarda la lista con los productos añadidos.
- **mostrar:** muestra la última lista de la compra guardada
- **listas:** muestra un listado de todas las listas guardadas ordenadas por fecha
- **salir:** Regresa al menú de login.

## 4. Cuarta fase

**Products.js:** Lo primero que he añadido aquí es la clase “producto” así como los métodos set y get correspondientes.

También he creado un array de productos, para añadir cada una de las imágenes y ponerla dentro de su categoría correspondiente.

```
// Crea un array de productos
let productos = [
  //Categoría Frutas y vegetales
  new Producto("Manzana", "Frutas y vegetales", "img/manzana.jpg"),
  new Producto("Naranja", "Frutas y vegetales", "img/naranja.jpg"),
  new Producto("Pimiento rojo", "Frutas y vegetales", "img/pimiento.jpg"),
  new Producto("Calabacin", "Frutas y vegetales", "img/calabacin.jpg"),

  // Categoría Panes y pastas
  new Producto("Barra", "Panes y pastas", "img/barra.jpg"),
  new Producto("Hogaza", "Panes y pastas", "img/hogaza.jpg"),
  new Producto("Pastas de te", "Panes y pastas", "img/pastasTe.jpg"),
  new Producto("Croissant", "Panes y pastas", "img/croissant.jpg"),

  // Categoría Leche y quesos
  new Producto("Leche", "Leche y quesos", "img/leche.jpg"),
  new Producto("Rulo de queso de cabra", "Leche y quesos",
"img/quesoCabra.jpg"),
  new Producto("Queso tierno", "Leche y quesos", "img/quesoTierno.jpg"),
  new Producto("Cuajada", "Leche y quesos", "img/cuajada.jpg"),

  // Categoría Carne y pescado
  new Producto("Filete de ternera", "Carne y pescado", "img/filete.jpg"),
  new Producto("Pollo", "Carne y pescado", "img/pollo.jpg"),
  new Producto("Lubina", "Carne y pescado", "img/lubina.jpg"),
  new Producto("Pulpo", "Carne y pescado", "img/pulpo.jpg"),

  // Categoría Cereales y pastas
  new Producto("Arroz", "Cereales y pastas", "img/arroz.jpg"),
  new Producto("Pasta fresca", "Cereales y pastas", "img/pastaFresca.jpg"),
  new Producto("Avena", "Cereales y pastas", "img/avena.jpg"),
```

```
    new Producto("Quinoa", "Cereales y pastas", "img/quinoa.jpg"),
  ];
```

El siguiente paso es crear un menú de categorías para los productos. Para cada categoría se crea un botón, y al hacer clic sobre una categoría aparecen los productos dentro de esa categoría. Si se vuelve a hacer clic en la misma categoría se limpia el contenedor ocultando los productos.

```
// Obtiene una referencia al div de categorías
let divCategorias = document.getElementById("categorias");

// Crea un array para almacenar las categorías de productos únicas
let categorias = [];

// Llena el array de categorías con las categorías de productos únicas
productos.forEach(producto => {
  if (!categorias.includes(producto.getTipo())) {
    categorias.push(producto.getTipo());
  }
});

// Crea una variable para almacenar la última categoría que se mostró
let ultimaCategoriaMostrada = null;

// Crea un botón para cada categoría y lo añádelo al div de categorías
categorias.forEach(categoria => {
  let divCategoria = document.createElement("div");
  divCategoria.className = "categoria";

  let botonCategoria = document.createElement("button");
  botonCategoria.innerText = categoria;

  let divProductosCategoria = document.createElement("div");
  divProductosCategoria.className = "productosCategoria";

  botonCategoria.addEventListener("click", function() {
    if (categoria === ultimaCategoriaMostrada) {
      // Si la categoría que se hizo clic es la misma que la última que
      se mostró, limpia el div de los productos y establece la última categoría
      mostrada a null
      divProductosCategoria.innerHTML = "";
      ultimaCategoriaMostrada = null;
    } else {
      // Si no, muestra los productos como normalmente y actualiza la
      última categoría mostrada
      mostrarProductos(categoria, divProductosCategoria);
      ultimaCategoriaMostrada = categoria;
    }
  });
});
```

```

divCategoria.appendChild(botonCategoria);
divCategoria.appendChild(divProductosCategoria);
divCategorias.appendChild(divCategoria);
});

```

La siguiente parte del código es la función “mostrarProductos” encargada de mostrar las imágenes de los productos, que el usuario pueda pinchar sobre estas imágenes y aparezca un prompt para definir la cantidad. Una vez aceptado se guarda en la lista de la compra.

***\*hasta que no se dé al botón de guardar no se guarda en una lista nueva, es decir, primero se añaden todas las cantidades y una vez finalizado este proceso, al dar al botón “guardar” se guardara una nueva lista de la compra\****

```

//Muestra los productos de una categoria, permite hacer clic para añadir
cantidad y agregarlo a la lista
function mostrarProductos(categoria, divProductosCategoria) {
    divProductosCategoria.innerHTML = "";

    productos.filter(producto => producto.getTipo() ===
categoria).forEach(producto => {
        let divProducto = document.createElement("div");
        divProducto.className = "producto";

        let imgProducto = document.createElement("img");
        imgProducto.src = producto.getEnlace();
        imgProducto.alt = producto.getNombre();

        let pNombre = document.createElement('p');
        pNombre.textContent = producto.getNombre();

        imgProducto.addEventListener("click", function() {
            let cantidad = prompt("¿Cuántas unidades de este producto
quieres?", "1");

            if (cantidad !== null) {
                let productoCompra = { producto: producto.getNombre(),
cantidad: cantidad };
                listaCompra.push(productoCompra);
                localStorage.setItem('listaCompra',
JSON.stringify(listaCompra));

                // Crear el elemento de mensaje y añadirlo al body
                let message = document.createElement('div');
                message.id = 'message';
                message.textContent = 'Producto añadido correctamente';
                document.body.appendChild(message);
            }
        });
    });
}

```

```

        // Mostrar el mensaje y luego ocultarlo después de 2 segundos
        message.classList.add('show');
        setTimeout(function() {
            message.classList.remove('show');

            // Borrar el mensaje
            setTimeout(function() {
                document.body.removeChild(message);
            }, 500);
        }, 2000);
    }
});

divProducto.appendChild(pNombre);
divProducto.appendChild(imgProducto);
divProductosCategoria.appendChild(divProducto);
});
}

```

Por último, un controlador para los botones “guardar”, “mostrar” y “listas”

- **Guardar:** Al Hacer clic sobre este botón se guarda la lista de la compra en el localStorage con la fecha en formato ISO, lo utilizare como clave y los productos como valor. Al guardar la lista aparecerá un mensaje con el texto “lista guardada correctamente” durante 2 segundos.
- **Mostrar:** Al pulsar este botón se buscan todas las listas guardadas en el localStorage, se filtra por la clave de la fecha, se ordenan y se muestra la más reciente. Esta es la lista que se muestra en “list.html”
- **Listas:** Al pulsar este botón, se muestra en contenedor “divListas” con todas las listas de la compra almacenadas en el localStorage, separadas por una línea horizontal.

***\*En esta parte me ha faltado una función para recuperar una lista guardada y seguir editándola\****

## 5. Quinta fase

**List.html:** Desde esta página mostraremos la última lista de la compra guardada, ubicada en el contenedor <listaCompras>, que se rellenara dinámicamente desde list.js tendremos también la opción de imprimir la lista desde el botón “Imprimir lista” y regresar al menú anterior desde el botón “salir”

**List.js:** Lo primero que tenemos aquí es la definición de la clase “list” con sus métodos get y sets correspondientes.

El siguiente código ubicado en “list.js” se utiliza para obtener y mostrar la lista de la compra más reciente. Después de pulsar sobre el botón “mostrar”

```
document.addEventListener("DOMContentLoaded", function() {
  // Obtener todas las claves de localStorage
  let keys = Object.keys(localStorage);

  // Filtrar solo las claves que tienen formato de fecha ISO
  let listKeys = keys.filter(key => {
    return /\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}.\d{3}Z/.test(key);
  });

  // Si no hay ninguna lista guardada
  if (listKeys.length === 0) {
    console.log('No hay ninguna lista guardada.');
```

```
    return;
  }

  // Ordenar las claves en orden descendente
  listKeys.sort().reverse();

  // Obtener la última lista de la compra
  let ultimaListaCompra = JSON.parse(localStorage.getItem(listKeys[0]));

  console.log(ultimaListaCompra);
});

function getUltimaLista() {
  // Obtiene todas las claves de localStorage
  let keys = Object.keys(localStorage);

  // Filtrar solo las claves que tienen formato de fecha ISO
  let listKeys = keys.filter(key => {
    return /\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}.\d{3}Z/.test(key);
  });

  // Si no hay ninguna lista guardada
  if (listKeys.length === 0) {
    console.log('No hay ninguna lista guardada.');
```

```
    return null;
  }

  // Ordenar las claves en orden descendente
  listKeys.sort().reverse();

  // Obtener la última lista de la compra
```

```

    let ultimaListaCompra = JSON.parse(localStorage.getItem(listKeys[0]));

    return ultimaListaCompra;
}

```

En esta parte es en la que se crea una lista borrando la anterior, para mostrar únicamente la última.

```

function mostrarLista(lista) {
    let contenedorLista = document.getElementById('listaCompras');
    contenedorLista.innerHTML = '';

    // Agrega un título a la lista
    let tituloElemento = document.createElement('h1');
    tituloElemento.textContent = "¡Tu lista de la compra, para que no se te olvide nada!";
    contenedorLista.appendChild(tituloElemento);

    // Agrega la fecha a la lista
    let fechaElemento = document.createElement('p');
    let fechaLista = new Date(lista.fecha);
    fechaElemento.textContent = `Fecha: ${fechaLista.getFullYear()}-${
    ${fechaLista.getMonth()+1}.toString().padStart(2, '0')}-${
    ${fechaLista.getDate().toString().padStart(2, '0')}
    ${fechaLista.getHours().toString().padStart(2,
    '0')}:${fechaLista.getMinutes().toString().padStart(2, '0')}`;
    contenedorLista.appendChild(fechaElemento);

    // Agrega los productos a la lista
    lista.productos.forEach(producto => {
        let elementoProducto = document.createElement('p');
        elementoProducto.textContent = producto.producto + ': ' +
        producto.cantidad;
        contenedorLista.appendChild(elementoProducto);
    });
}

document.addEventListener("DOMContentLoaded", function() {
    // Código para obtener la última lista...
    let ultimaListaCompra = getUltimaLista();

    // Mostrar la última lista
    if(ultimaListaCompra) {
        mostrarLista(ultimaListaCompra);
    } else {
        console.log('No hay ninguna lista guardada.');

```

Por último, un evento para llamar a “Windows print” desde donde el usuario podrá imprimir su lista después de presionar el botón “imprimir lista”

```
// Evento para el boton de imprimir lista
document.addEventListener("DOMContentLoaded", function() {

    document.getElementById("printButton").addEventListener("click",
function() {
    window.print();
});
});
```

## 6. Sexta fase

En la última fase he trabajado en la hoja de estilos “style.css” para cambiar colores, tamaños, dar efectos hover al pasar el ratón por encima de un botón, etc.

A modo de ejemplo, el botón aceptar, de la página de login. He definido en su estado de “reposo” un color para el fondo y un color para el texto.

```
.aceptar{
    background-color: #ffe177;
    color: #1f7551;
}
```

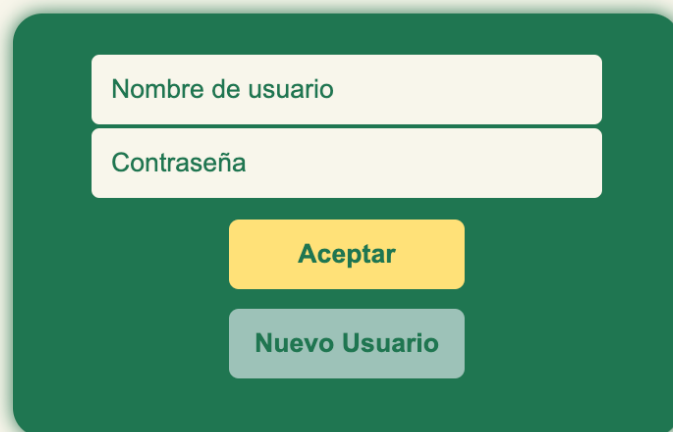
Pero cuando pasamos el ratón por encima del botón los colores se invierten. Esto esta aplicado a todos los botones de la pagina.

```
.aceptar:hover{
    background-color: #1f7551;
    color: #ffe177;
}
```

## Imágenes de la aplicación

### Página de Login.

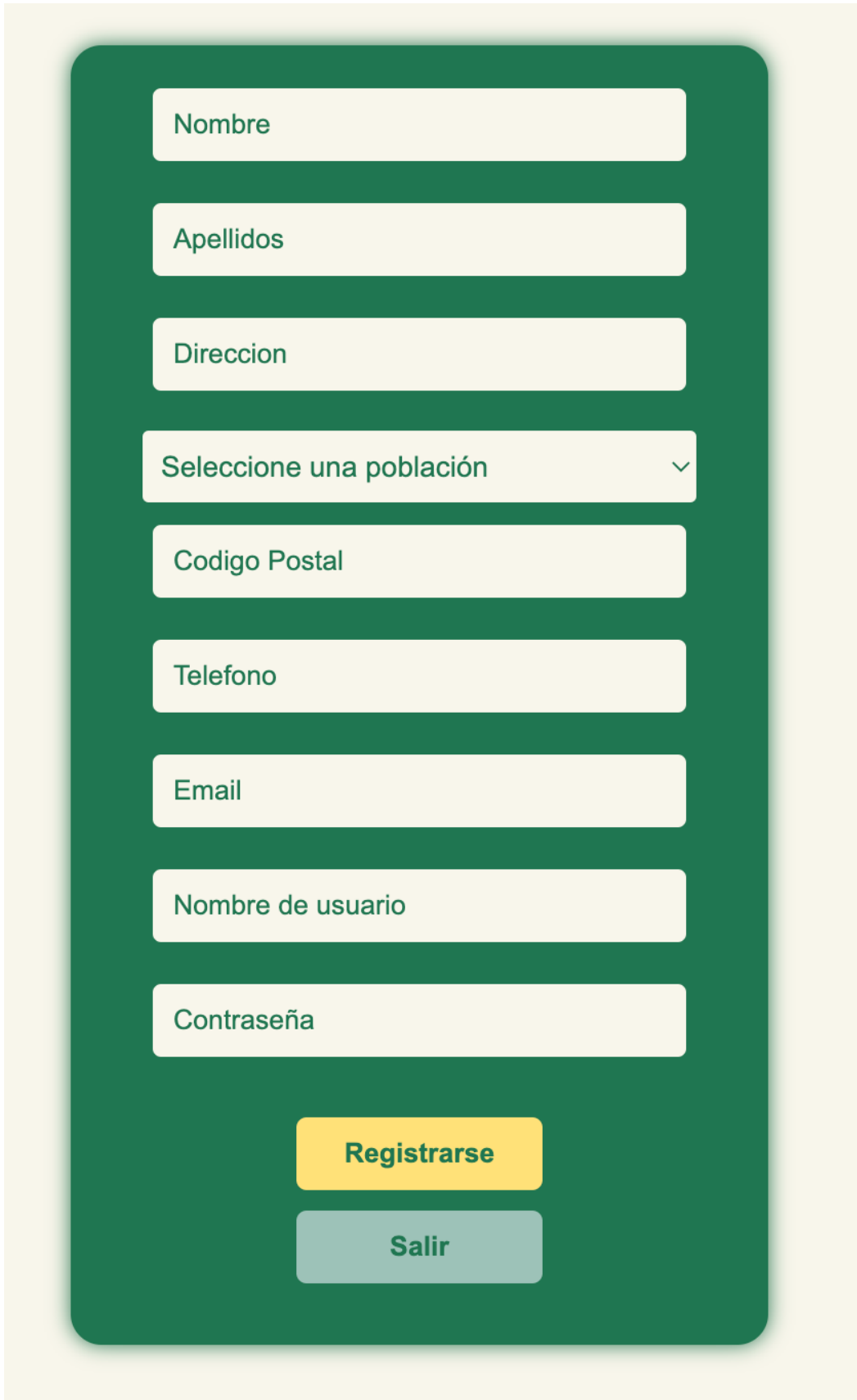
**Tu aplicación web para la lista de la Compra!**



A login form centered on a light beige background. The form is contained within a dark green rounded rectangle. It features two white input fields: the top one is labeled 'Nombre de usuario' and the bottom one is labeled 'Contraseña'. Below these fields are two buttons: a yellow 'Aceptar' button and a light blue 'Nuevo Usuario' button.



## Formulario de registro de un nuevo usuario

Un formulario de registro de un nuevo usuario con un fondo verde oscuro y un contenedor central de color crema. El formulario contiene campos de entrada para Nombre, Apellidos, Dirección, Selección de población (con una flecha hacia abajo), Código Postal, Teléfono, Email, Nombre de usuario y Contraseña. Al final del formulario hay dos botones: uno amarillo con el texto 'Registrarse' y uno gris con el texto 'Salir'.

Nombre

Apellidos

Dirección

Seleccione una población ▾

Código Postal

Teléfono

Email

Nombre de usuario

Contraseña

**Registrarse**

**Salir**

## Página de selección de productos.

Frutas y vegetales

Panes y pastas

Leche y quesos

Carne y pescado

Cereales y pastas

---

Guardar

Mostrar

Listas

Salir

---

## Detalle de una categoría seleccionada

### Frutas y vegetales

**Manzana**



**Naranja**



**Pimiento rojo**



**Calabacín**



Ultima lista de la compra guardada

# ¡Tu lista de la compra, para que no se te olvide nada!

Fecha: 2023-06-08 18:04

Naranja: 1

Calabacin: 1

Manzana: 1

Pimiento rojo: 1

Imprimir lista

Salir

Historial de listas de la compra

Compartir

Fecha: 2023-06-08 18:47

Producto: Rulo de queso de cabra, Cantidad: 3

Producto: Queso tierno, Cantidad: 3

Producto: Leche, Cantidad: 3

Fecha: 2023-06-08 18:47

Producto: Filete de ternera, Cantidad: 5

Producto: Pollo, Cantidad: 3

Producto: Pulpo, Cantidad: 3

Fecha: 2023-06-08 18:46

Producto: Pasta fresca, Cantidad: 5

Producto: Arroz, Cantidad: 4

Producto: Quinoa, Cantidad: 5