

INSTITUTO DE EDUCACIÓN SECUNDARIA JOSÉ PLANES

Departamento de Informática y Comunicaciones
Técnico Superior en Desarrollo de aplicaciones Web

C/ Maestro Pérez Abadía, 2
30100 Espinardo – Murcia
T. 968 834 605
30010577@murciaeduca.es
www.iesjoseplanes.es

Memoria del proyecto

Desarrollo de aplicaciones web

<EasyTPV>

Autores/as:

<Pablo Esturillo Lorente>

<Marcos Lopez Almela>

<Raúl Guirao Blázquez>

Profesor/a-coordinador/a:

<Javier Andrés Gutiérrez>

Murcia, Junio de 2024



Esta obra está bajo una [licencia de Creative Commons Reconocimiento-CompartirIgual 4.0 Internacional](#).

En primer lugar, nos gustaría dar las gracias a nuestros profesores por los consejos que nos han ofrecido durante el desarrollo de esta aplicación(Javier Andres y Luis Miguel).

También nos gustaría agradecer a nuestro compañero Antonio Cano por ayudarnos en el despliegue de la aplicación.

Y también a los compañeros de GETIC por ayudar con algunas parte de la estructura de nuestra base de datos, a nuestros demás tutores de las prácticas. A familia que ha aportado su granito de ayuda para proporcionar ideas. A Natalia por consejos y aportes para darle nuevos enfoques a la aplicación, gracias por tanto en tan poco tiempo que nos conocemos <3.

Contenido

1	Resumen extendido	1
2	Palabras clave	1
3	Introducción	2
4	Estado del arte/trabajos relacionados	2
5	Análisis de objetivos y metodología.	3
	5.2 Metodología	4
6	Diseño y resolución del trabajo realizado	5
	6.1 Diseño de la base de datos	6
	6.2 Login y registro de usuarios con Laravel	7
	6.2.1 Registro	8
	6.2.2 Inicio de sesión	8
	6.3 Sección TPV	9
	6.3.1 Columna izquierda	9
	6.3.2 Columna central (TPV)	9
	6.3.3 Columna central(Ventas)	10
	6.3.4 Columna central(Sala)	11
	6.3.5 Columna central(Arqueo)	11
	6.3.6 Sección venta	12
	6.4 Sección gestión	13
	6.5 BackEnd	15
	6.5.1 TPV	15
	6.5.2 Gestión Inventario	25
	6.6 Diseño	29
	6.6.1 Diseño Prototipos de alta fidelidad(Figma)	29
	6.6.2 Elección de colores y tipografía	29

7	Presupuesto	31
8	Conclusiones y vías futuras	32
9	Bibliografía/Webgrafía.	36

1 Resumen extendido

EasyTPV es una aplicación web desarrollada con Laravel y Alpine.js, pensada para ayudar en la organización y la asignación de tareas de proyectos buscando siempre el máximo potencial a las características necesarias para el mundo de la hostelería.

En esta aplicación los usuarios que son empleados de cualquier tienda o bar pueden vender todos los productos que tienen en stocks a los clientes y a partir de ello crear tickets de venta a los usuarios con todo lo que han comprado.

Todos los productos pueden llegar a agotarse haciendo que no sea posible su venta, a parte el empleado puede buscar fácilmente los artículos ya sea por nombre o por el tipo de producto, se pueden visualizar las ventas del día el conteo de la caja (el usuario asigna la caja en la que está), las salas que tiene el negocio para ver la venta, además el empleado puede iniciar la jornada de trabajo a partir del sistema de fichaje.

Por otro lado tenemos la parte de la vista del Manager/Encargado que puede ver el stock de todos los productos el estado el mismo puede insertar nuevos productos, editarlos y borrarlos. Por parte del apartado de los empleados se puede ver toda la información relevante a ellos y filtrar para encontrar al empleado querido. Los tipos de iva también son modificables y se pueden crear nuevas cajas para el negocio

2 Palabras clave

Hostelería, gestión, JavaScript, Alpine.js, PHP, Laravel, Tailwind.css, HTML5.

3 Introducción

El proyecto surgió a raíz de que uno de nuestros compañeros tiene experiencia en el mundo de la hostelería y veía que todas las TPVs que se utilizaban eran iguales y genéricas. Por ese motivo quisimos crear una nosotros mismos con más funcionalidades y un diseño más moderno e intuitivo. Por ello para ponernos a empezar con el desarrollo de la aplicación se utilizaron nuevas tecnologías no vistas en clase como Laravel y Alpine.js.

Para el login se decidió utilizar Laravel con Bootstrap para usar algo conocido y se viera como nos seguimos desenvolviendo.

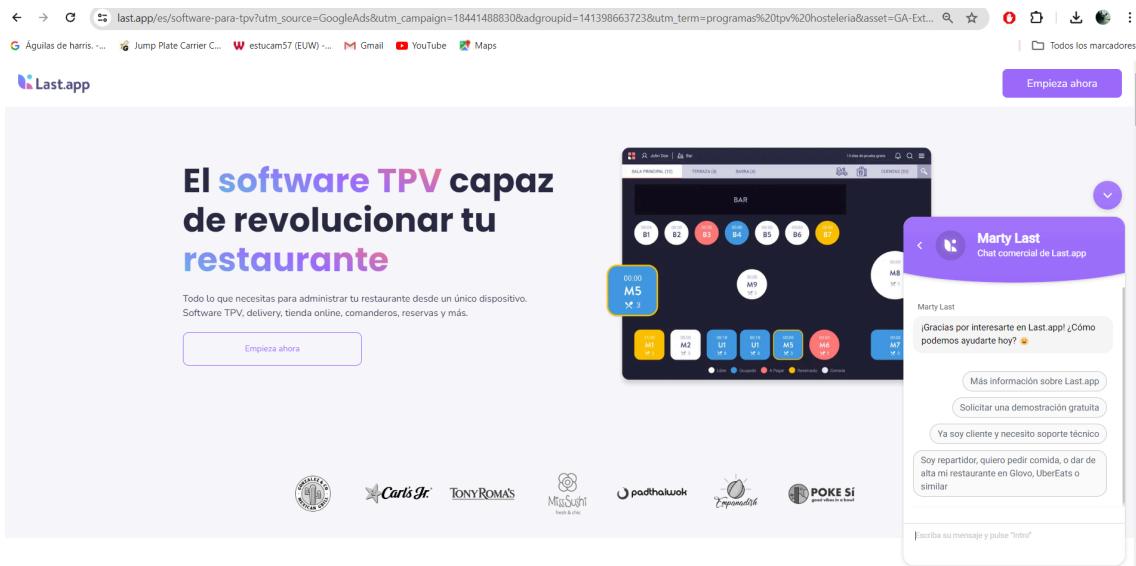
Para utilizar EasyTpv es sencillo, el usuario accede a la TPV con su usuario y contraseña, el empleado pone en la caja que se encuentra, la TPV tienes varios productos todo escogibles (Si no están es que no hay stock) pueden hacer diferentes modos de pago (Efectivo y tarjeta). Se crea la factura y se guarda el dinero en la caja.

4 Estado del arte/trabajos relacionados

En la mayoría de TPV que se ejecuten en web no suelen ser directas, son programas que abarcan varios campos que no tienen porqué estar relacionadas a una TPV ni a la hostelería y añaden una sección de TPV simple para que abarque todo los tipos de negocio posibles. Es difícil conseguir capturas de esto porque la mayoría de TPV son de pago y no te dejan tener una previsualización del producto.

Queremos que nuestra aplicación se centre al 100% en los aspectos que puede necesitar un local de hostelería tales como el control de stock. Lo que hemos buscado con EasyTpv es agilizar los procesos lo más rápido posible

para hacer que el producto llegue lo antes posible al consumidor



5 Análisis de objetivos y metodología.

Creación de base de datos adecuada.

- Login y registro de usuarios.
- Creación, edición y eliminación de productos por parte del manager.
- Creación, edición y eliminación de usuarios de tareas por parte del manager.
- Creación, edición y eliminación de ivas por parte del manager.
- Creación, edición y eliminación de cajas por parte del manager.
- Creación, edición y eliminación de categorías por parte del manager.
- Creación, edición y eliminación del carrito por parte del empleado.
- Sistema de fichaje para el usuario.
- Sistema de Arqueo para el control de caja
- Añadir y eliminar empleados.
- Diseño adaptativo.
- Web accesible.

- Despliegue de la aplicación.

5.2 Metodología.

Para llevar a cabo los diferentes objetivos propuestos, hemos seguido la siguiente metodología de trabajo.

Metodología "CICLO" para Proyectos de Programación

CICLO: Colaboración, Iteración, Control de calidad, Lanzamiento, Optimización

1. Colaboración

- Formación del Equipo: Asignar roles (líder de proyecto, desarrolladores, diseñadores, testers, etc.).
- Herramientas de Colaboración: Seleccionar herramientas como GitHub, Jira, Slack, Trello.
- Kick-off Meeting: Reunión inicial para definir objetivos, alcance y cronograma del proyecto.

2. Iteración

- Metodología Ágil: Adoptar Scrum

Scrum: Dividir el proyecto en sprints (1 semana), con reuniones diarias de las tareas realizadas a lo largo del día.

Planificación: Definir tareas y objetivos para cada iteración.

3. Control de Calidad

- Revisión de Código: Implementar revisiones de código sistemáticas (pull requests en GitHub).

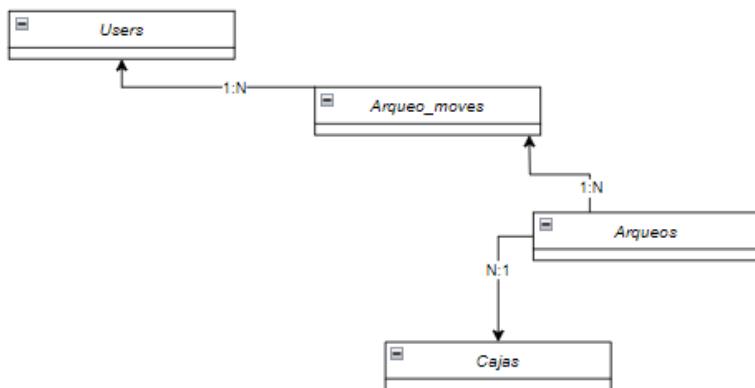
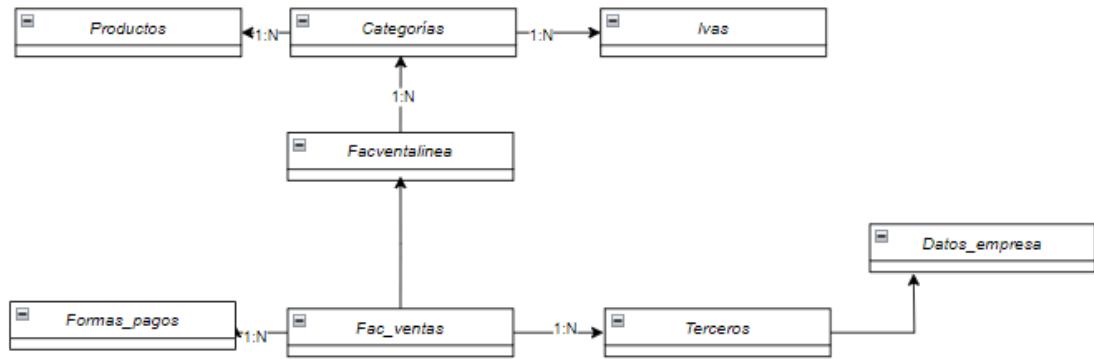
4. Uso de Figma para la creación de prototipos del proyecto de una manera sencilla para intentar tomar decisiones sobre las interacciones sobre los elementos de la página.
5. Uso de git para el control total de las versiones con diferentes ramas en el proyecto para poder ir poco a poco tocando nuevas funcionalidades de la web.
6. Uso de Whatsapp para una comunicación más rápida por posibles problemas que pudiesen suceder por implementaciones recientes en el código.

6 Diseño y resolución del trabajo realizado

Partes de la aplicación. Partes principales

6.1 Diseño de la base de datos

Una vez decidido todo lo que necesitamos en la base de datos inicialmente nos pusimos en marcha para crear el diseño de la base de datos. Así que realizamos un modelo entidad-relación utilizando Draw.io.



6.2 Login y registro de usuarios con Laravel

Utilizamos una de la tabla de la base de datos users (para el registro y el usuario), en nuestro caso para poder registrar un usuario solo un administrador/manager puede hacerlo a través de la ventana de gestión de usuarios.

Imagen	Nombre	Correo	Privilegios	Puesto
	Marcos	markosstone@gmail.com	0	
	Marcos			
	Manuel			
	Marcos			
	Manolagor			
	Salmonete			
	Ramon	ramon@gmail.com	1	Salteador
	rafita@gmail.com	marcos@gmail.com	0	Pescador

Las contraseñas e imágenes aportadas en el formulario post que se manda se encripta en base 64. Simplemente conectamos el frontend Alpine con Laravel. Luego simplemente para logearnos dentro del sistema introducimos el usuario y la contraseña para verificarse.



6.2.1 Registro

Para el registro utilizamos la función “registrarEmpleado”, que hace las comprobaciones necesarias para ver que todo lo introducido en el front cumple con el tipo, además verifica que el correo no este duplicado por lo que tiene que ser único. Una vez son validados los campos se encriptan tanto la contraseña como las fotos y se guardan en la base de datos. Para acceder a este apartado de la creación hay que dirigirse a la siguiente ruta <http://127.0.0.1:8000/gestion-inventario> dentro de esta url ir al apartado Empleados y darle al botón de creación de empleado.

6.2.2 Inicio de sesión

Para el inicio de sesión se ha creado la función “login” se trae a un usuario con el correo introducido y una vez se ha hace la comprobación del correo y de la contraseña tenemos la autentificación superada y nos redirigirá directamente a la url tpv.blade.php o no tirará una excepción.

```
public function login()
{
    $user = User::where('email', $this->email)->first();

    if ($user && password_verify($this->password, $user->password)) {
        // Autenticación exitosa
        // Aquí puedes realizar las acciones necesarias, como establecer una sesión o redirigir al usuario a una página específica
        return redirect()->route('tpv.blade.php');
    } else {
        // Autenticación fallida
        // Aquí puedes mostrar un mensaje de error o realizar otras acciones necesarias
        session()->flash('error', 'Usuario o contraseña incorrectos');
        return redirect()->back();
    }
}
```

6.3 Sección TPV

Esta es la vista principal de nuestra web, esta vista se centra en realizar ventas de productos y organización de sala.

The screenshot shows the TPV interface. On the left is a vertical sidebar with icons for TPV, Ventas, Sala, Arqueo, and Salir. The main area has a search bar at the top. Below it is a navigation bar with categories: aperitivos, entrantes, plato principal, postre, and bebidas. Underneath these categories are product cards with images and names like prueba 21, prueba 4, PRUEBA 5, prueba 9, prueba 8, producto 7, and asdasas. To the right of the products is a table of sales with columns for ID, Product, Price, and Total. At the bottom right are buttons for VENDER, Borrar, and Guardar.

6.3.1 Columna izquierda

Esta vista se divide en tres columnas. Empezando por la **izquierda** tenemos la zona de iconos que nos servirá para navegar por las distintas secciones de nuestra TPV(TPV, Ventas, Sala y Arqueo). Esto solo cambiará nuestra **columna central**.

6.3.2 Columna central (TPV)

La columna **central** será la que más cambios tendrá. Empezando por la vista de TPV, que será la principal. Se divide en tres partes:

- Un input para buscar productos por nombre.
- Categorías para agrupar varios productos.
- Sección productos la cual tendrá los productos los cuales al pulsarlos se añadirán al carrito para realizar ventas.

Al estar la columna **derecha** y la sección TPV de la columna **central** tan compaginadas haré una pequeña incisión para explicar esta.

La columna **derecha** se compone del listado de productos(carrito), donde nos mostrará las unidades, nombre, sub.total, total y un botón para eliminar esa línea.

Más abajo tendremos un desglose de los valores del carrito teniendo base imponible(precio sin iva), iva y total(base imponible + iva).

Por último tenemos los botones los cuales, por orden de izquierda a derecha, nos borraran todo el carrito, guardar el carrito en una mesa de sala y abrir la ventana de venta.

6.3.3 Columna central (Ventas)

Retomando la columna **central**, la siguiente sección será la de ventas la cual tiene un buscador por referencia y nos muestra un listado de las facturas que se han realizado en esa caja, pudiendo separar las facturas de las distintas cajas.

Filtrar por referencia						
REFERENCIA	TERCERO	FECHA	FORMA DE PAGO	BASE IMPONIBLE	TOTAL IVA	TOTAL
FS24-0011	Anonimo	15/06/2024	Efectivo	34.00€	0.00€	40.26€
FS24-0010	Anonimo	15/06/2024	Efectivo	32.00€	3.80€	35.80€
FS24-0009	Anonimo	15/06/2024	Efectivo	12.00€	2.52€	14.52€
FS24-0008	Anonimo	15/06/2024	Efectivo	12.00€	2.52€	14.52€
FS24-0007	Anonimo	15/06/2024	Tarjeta	5.00€	1.05€	6.05€
FS24-0006	Anonimo	15/06/2024	Efectivo	12.00€	2.52€	14.52€
FS24-0005	Anonimo	15/06/2024	Efectivo	5.00€	1.05€	6.05€
FS24-0004	Anonimo	15/06/2024	Efectivo	12.00€	2.52€	14.52€
FS24-0003	Anonimo	15/06/2024	Efectivo	19.00€	3.99€	22.99€
FS24-0002	Anonimo	15/06/2024	Tarjeta	12.00€	2.52€	14.52€
FS24-0001	Anonimo	15/06/2024	Efectivo	5.00€	1.05€	6.05€

6.3.4 Columna central (Sala)

La siguiente sección es Sala. Aquí tenemos la idea de mostrar las mesas en las que se organizaría el local. Al guardar una mesa se indica quien lleva esa mesa y que mesa se está utilizando. Al pulsar en una mesa se te llevarán los productos guardados al carrito para poder hacer la venta.

Mesa 1 X	Mesa 2 X	Mesa 3 X
Vendedor:	Vendedor:	Vendedor:
Mesa 4 X	Mesa 5 X	Mesa 6 X
Vendedor: Marcos	Vendedor:	Vendedor:
Mesa 7 X	Mesa 8 X	Mesa 9 X
Vendedor:	Vendedor:	Vendedor:
Mesa 10 X	Mesa 11 X	Mesa 12 X
Vendedor:	Vendedor:	Vendedor:

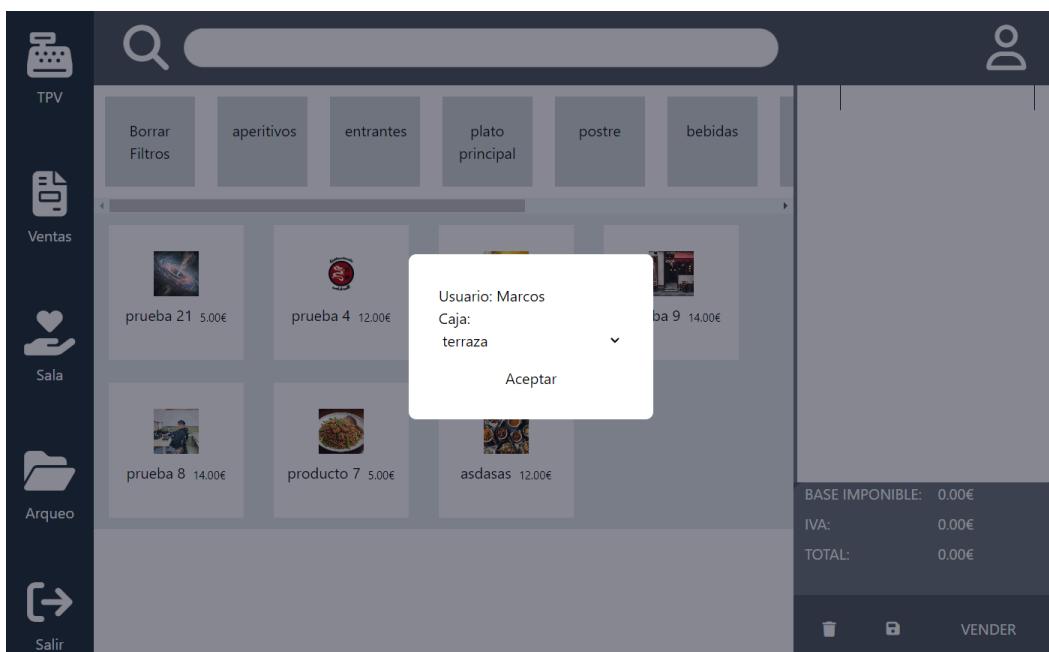
6.3.5 Columna central (Arqueo)

Por último tenemos la vista Arqueo la cual es parecida a Ventas pero en esta nos mostrará los diferentes Arqueos que se hagan la caja en la que se esté.

Filtrar por fecha						
Fecha	Saldo Inicial	Pagos Tarjeta	Pagos Efectivo	Cantidad Total	Cantidad Final	Comprobante
15-06-2024	100.00€	20.57€	128.97€	149.54€	128.97€	\$
14-06-2024	100.00€	0.00€	100.00€	100.00€	100.00€	\$
13-06-2024	0.00€	0.00€	0.00€	80.00€	100.00€	\$

El botón comprobante nos abre una ventana emergente para realizar el arqueo de cierre.

Para seguir con los Arqueos, al empezar el día tenemos que realizar el arqueo inicial para mantener un control del cambio que hay en caja al empezar el día. Tenemos que indicar en qué caja vamos a hacer el servicio.

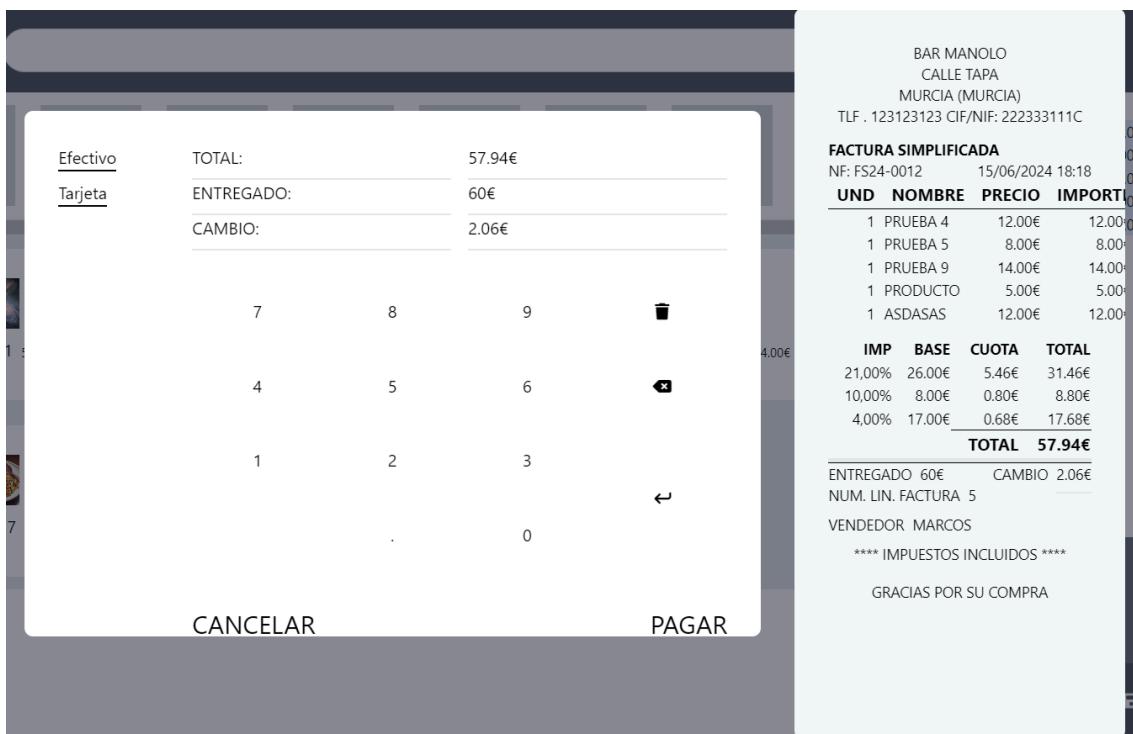


Una vez indicada la caja se nos abre otra pestaña para hacer el conteo del cambio. Tras hacer eso ya se puede usar la TPV con normalidad.

6.3.6 Sección venta

Para terminar tenemos la ventana de venta la cual nos muestra una interfaz simple para introducir el método de pago y en caso de ser necesario el dinero entregado. También se puede apreciar una previsualización del ticket que se

imprimirá.



6.4 Sección Gestión

Esta vista está reservada a los usuarios con privilegios de manager. En ella realizamos la gestión de los productos, empleados, categorías, ivas, etc.

Imagen	Nombre	Precio	IVA	Categoría	Stock	Estado	Editar	Borrar
	prueba 21	5.00€	21,00%	plato principal	-47	●	Editar	Borrar
	prueba 4	12.00€	21,00%	plato principal	-51	●	Editar	Borrar
	PRUEBA 5	8.00€	10,00%	entrantes	-9	●	Editar	Borrar
	prueba 9	14.00€	21,00%	plato principal	-31	●	Editar	Borrar
	prueba 8	14.00€	21,00%	plato principal	-8	●	Editar	Borrar
	producto 7	5.00€	4,00%	entrantes	-17	●	Editar	Borrar

La primera vista y principal es la de Inventario en la que podremos crear, editar y borrar los productos que tengamos.

Ambos formularios de creación y edición tienen la misma estructura:

NUEVO PRODUCTO

Nombre* Precio*

IVA
0,01% Categoría
aperitivos

Stock Estado
Seleccione Estado

Imagen
Seleccionar archivo Ningún archivo seleccionado

Guardar

En este podremos indicar todos los campos para crear un producto y asignarle una foto.

Luego tenemos la sección de categorías para crear, editar y borrarlas también.

Nombre	Editar	Borrar
aperitivos	Editar	Borrar
entrantes	Editar	Borrar
plato principal	Editar	Borrar
postre	Editar	Borrar
bebidas	Editar	Borrar
bebidas	Editar	Borrar
tartas	Editar	Borrar
pro	Editar	Borrar
asdsaaas	Editar	Borrar

El mismo caso para los ivas.

Inventario	Empleados	Categorías	Ivas	Cajas
QTY				
0,01%	Editar	Borrar		
4,00%	Editar	Borrar		
21,00%	Editar	Borrar		
10,00%	Editar	Borrar		

Y también las cajas.

Inventario	Empleados	Categorías	Ivas	Cajas
Nombre				
sala 11	Editar	Borrar		
terraza	Editar	Borrar		

6.5 BackEnd

6.5.1 TPV

función obtenerSiguienteNúmeroFactura()

```
public function obtenerSiguienteNúmeroFactura()
{
    // Obtener el año actual en formato de dos dígitos (YY)
    $year = now()->format('y');

    // Construir el prefijo de la factura (FSYY-)
    $prefix = 'FS' . $year . '-';

    // Obtener la última factura que cumpla con el prefijo ordenándola por ID descendente
    $lastInvoice = Factura::where('name', 'like', $prefix . '%')->orderBy('id', 'desc')->first();

    // Obtener el número secuencial (XXXX) sumando 1 al último número encontrado
    $sequence = $lastInvoice ? intval(substr($lastInvoice->name, -4)) + 1 : 1;

    // Construir el nombre completo de la próxima factura
    $this->getNextRef = $prefix . str_pad($sequence, 4, '0', STR_PAD_LEFT);

}
```

- Esta función tiene la finalidad de devolverme el próximo número de referencia para una factura. Tiene una estructura definida la cual utiliza las siglas FS(Factura simplificada), el año actual y números que van en aumento por cada factura creada (FS24-0001). Esto lo llevo a mi vista para imprimirlo en la factura.

función updateSumaTotalEfectivo()

```

public function updateSumaTotalEfectivo($caja_id)
{
    $fecha_actual = now()->format('Y-m-d');

    // Obtener la suma total de ventas pagadas en efectivo para la fecha y caja especificadas
    $sumaTotalEfectivo = FacVenta::whereDate('fecha', $fecha_actual)
        ->where('caja_id', $caja_id)
        ->where('forma_pago_id', 1) // ID 1 para efectivo
        ->sum('total');

    // Actualizar el registro de arqueo existente
    $arqueoExistente = Arqueo::where('fecha', $fecha_actual)
        ->where('caja_id', $caja_id)
        ->first();

    if ($arqueoExistente) {
        $arqueoExistente->update([
            'saldo_efectivo' => $sumaTotalEfectivo,
            'saldo_total' => $arqueoExistente->saldo_tarjeta + $sumaTotalEfectivo,
        ]);
    }
    $this->sumaTotalEfectivo = $sumaTotalEfectivo;
}

```

- Esta función recibe el id de caja y se usa para actualizar el valor de total_efectivo y saldo_total. Primero obtenemos la fecha actual y obtenemos la suma del campo total de la tabla fac_ventas cuando los campos de caja_id y fecha coincidan con los valores que les indicamos. Luego obtenemos el arqueo de este día de la tabla Arqueos cuya caja_id coincide. Tras obtener el arqueo que queremos le actualizamos los campos de saldo_efectivo con la suma de totales de la tabla fac_venta y el saldo_total sumandole \$sumaTotalEfectivo y el valor de saldo_tarjeta que actualizaremos en la siguiente función.

función updateSumaTotalTarjeta()

```
public function updateSumaTotalTarjeta($caja_id)
{
    $fecha_actual = now()->format('Y-m-d');

    // Obtener la suma total de ventas pagadas con tarjeta para la fecha y caja especificadas
    $sumaTotalTarjeta = FacVenta::whereDate('fecha', $fecha_actual)
        ->where('caja_id', $caja_id)
        ->where('forma_pago_id', 2) // ID 2 para tarjeta
        ->sum('total');

    // Actualizar el registro de arqueo existente
    $arqueoExistente = Arqueo::where('fecha', $fecha_actual)
        ->where('caja_id', $caja_id)
        ->first();

    if ($arqueoExistente) {
        $arqueoExistente->update([
            'saldo_tarjeta' => $sumaTotalTarjeta,
            'saldo_total' => $arqueoExistente->saldo_efectivo + $sumaTotalTarjeta,
        ]);
    }
    $this->sumaTotalTarjeta = $sumaTotalTarjeta;
}
```

- Sigue la misma lógica que updateSumaTotalEfectivo() pero con el campo saldo_tarjeta.

función updateTotalesArqueo()

```
public function updateTotalesArqueo($caja_id)
{
    $fecha_actual = now()->format('Y-m-d');

    // Actualizar el registro de arqueo existente
    $arqueoExistente = Arqueo::where('fecha', $fecha_actual)
        ->where('caja_id', $caja_id)
        ->first();

    if ($arqueoExistente) {
        // Actualizar el valor de saldo_final para que sea igual a saldo_total
        $arqueoExistente->update([
            'saldo_final' => $arqueoExistente->saldo_efectivo,
        ]);
    }
}
```

- Obtengo el arqueo con la fecha actual y la caja_id que le paso a la función y le actualizo el valor de saldo_final para que sea igual que el campo saldo_efectivo del mismo arqueo.

funcion listarFacturas()

```
public function listarFacturas()
{
    $facturas = FacVenta::select('caja_id', date('Y-m-d'))
        ->select('name', 'tercero_id', 'fecha', 'base_imp', 'total_iva', 'total', 'forma_pago_id')
        ->get();

    $this->listadoFacturas = $facturas;
}
```

- Esta función me devuelve un array para listar las facturas en la vista.

función comprobarArqueo()

```

public function comprobarArqueo($caja_id)
{
    $arqueos = Arqueo::whereDate('fecha', date('Y-m-d'))
        ->where('caja_id', $caja_id)
        ->exists();

    return $arqueos;
}

```

- Esta función me devuelve un valor booleano si ya existe un registro en la tabla Arqueos con la fecha actual y la caja_id indicada. Esto me permite que en mi vista no tenga que hacer el arqueo inicial cada vez que se refresca la página.

función valorSaldoInicial()

```

public function valorSaldoInicial($caja_id)
{
    $ultimoArqueo = Arqueo::where('caja_id', $caja_id)
        ->latest('id')
        ->first();

    $saldo_inicial = $ultimoArqueo ? $ultimoArqueo->saldo_final : 10;

    $this->saldoInicial = $saldo_inicial;
}

```

- En esta función obtengo el valor de saldo_inicial para usarlo luego en mi vista en la zona para hacer el arqueo inicial.

función valorBilletes()

```

public function valorBilletes($caja_id)
{
    $ultimoArqueo = ArqueoMove::where('caja_id', $caja_id)
        ->latest('id')
        ->first();

    $billetes = $ultimoArqueo ? json_decode($ultimoArqueo->billetes, true) :
        [ "1" => 0, "2" => 0, "5" => 0, "10" => 0, "20" => 0, "50" => 0, "100" => 0, "200" => 0, "500" => 0, "05" => 0, "02" => 0, "01" => 0 ];

    return $billetes;
}

```

- busca el registro más reciente en la tabla ArqueoMoves para una caja específica. Si encuentra un registro, decodifica el campo billetes de ese

registro en un array. Si no encuentra un registro, retorna un array inicializado con todas las denominaciones de billetes y monedas establecidas a 0.

función crearArqueoInicial()

```

public function crearArqueoInicio($caja_id, $billetes)
{
    $user = User::where('name', $this->user)->first();

    $ultimoArqueo = Arqueo::where('caja_id', $caja_id)
        ->latest('id')
        ->first();

    $saldo_inicial = $ultimoArqueo ? $ultimoArqueo->saldo_final : 0;

    $fecha = now()->format('Y-m-d');

    $arqueo = Arqueo::create([
        'fecha' => $fecha,
        'saldo_inicial' => $saldo_inicial,
        'saldo_efectivo' => $saldo_inicial,
        'saldo_tarjeta' => 0,
        'saldo_final' => $saldo_inicial,
        'saldo_total' => $saldo_inicial,
        'caja_id' => $caja_id,
    ]);

    $arqueo->save();

    $arqueoMove = ArqueoMove::create([
        'arqueo_id' => $arqueo->id,
        'caja_id' => $caja_id,
        'user_id' => $user->id,
        'billetes' => json_encode($billetes),
        'moves' => 'inicio',
    ]);

    $arqueoMove->save();
    $this->saldo_inicialDS = $saldo_inicial;

    $this->arqueos = Arqueo::select('id', 'fecha', 'saldo_inicial', 'saldo_efectivo', 'saldo_total', 'caja_id', 'user_id', 'moves', 'billetes')
        ->get();
    return $arqueo;
}

```

- Crea un nuevo registro de arqueo inicial para una caja específica y guarda la información de los billetes asociados. Obtiene el usuario actual, determina el saldo inicial basado en el último arqueo (si existe), y guarda tanto el nuevo arqueo como el movimiento de arqueo. Luego, actualiza las propiedades del objeto y retorna el nuevo arqueo creado.

función crearArqueoCierre()

```

public function crearArqueoCierre($user_name, $caja_id, $billetes, $saldoInicialSiguiente, $sumaBilletes)
{
    $user = User::where('name', $user_name)->first();

    $arqueoExistente = Arqueo::where('fecha', date('Y-m-d'))
        ->where('caja_id', $caja_id)
        ->first();

    $arqueoExistente->update([
        'saldo_total' => $this->sumaTotalEfectivo + $this->sumaTotalTarjeta,
        'saldo_efectivo' => $this->sumaTotalEfectivo,
        'saldo_tarjeta' => $this->sumaTotalTarjeta,
        'saldo_final' => $sumaBilletes,
    ]);

    ArqueoMove::create([
        'arqueo_id' => $arqueoExistente->id,
        'caja_id' => $caja_id,
        'user_id' => $user->id,
        'billetes' => json_encode($billetes),
        'moves' => 'cierre',
    ]);

    $this->arqueos = Arqueo::select('id', 'fecha', 'saldo_inicial', 'saldo_efectivo', 'saldo_tarjeta', 'caja_id');
    // Reiniciar sumaTotalTarjeta después del arqueo de cierre
    $this->sumaTotalTarjeta = 0;
    $this->sumaTotalEfectivo = 0;
}

```

- sigue la misma lógica que crearArqueoInicial() pero en este caso cambia moves a cierre para diferenciar cuando se inicia o cierra un arqueo.

funcion guardarFactura()

```

public function guardarFactura()
{
    $factura = new FacVenta();
    $factura->tercero_id = $this->terceroSeleccionado;
    $factura->forma_pago_id = $this->formaPagoSeleccionada;
    $factura->fecha = now();
    $factura->base_imp = array_sum(array_column($this->lineas_factura, 'base_imp'));
    $factura->total_iva = array_sum(array_column($this->lineas_factura, 'total_iva'));
    $factura->total = $factura->base_imp + $factura->total_iva;

    $factura->save();

    foreach ($this->lineas_factura as $linea) {
        $facVentaLinea = new FacVentaLinea();
        $facVentaLinea->fac_venta_id = $factura->id;
        $facVentaLinea->producto_id = $linea['producto_id'];
        $facVentaLinea->cantidad = $linea['cantidad'];
        $facVentaLinea->precio_unitario = $linea['precio_unitario'];
        $facVentaLinea->base_imp = $linea['base_imp'];
        $facVentaLinea->total_iva = $linea['total_iva'];
        $facVentaLinea->save();
    }

    return $factura;
}

```

- Crea un nuevo objeto de factura, asignándole datos como el cliente, la forma de pago, la fecha y los totales calculados, y luego guarda esta factura en la base de datos. A continuación, crea y guarda cada línea de producto asociada a la factura en la base de datos, asegurando que ambas, la factura y sus líneas de productos, sean registradas correctamente, manteniendo la integridad y consistencia de los datos. Finalmente, retorna el objeto de factura creado.
- Esta función no está terminada y para implementaciones futuras.

funcion crearTicket()

```

public function crearTicket($valores, $valorIVA, $totalSinDesglosar, $totalCarrito, $usuarioName, $metodoDePago, $caja_id)
{
    $usuario = User::where('name', $usuarioName)->first();
    $usuarioId = $usuario->id;

    //restar stock
    foreach ($valores as $value) {
        $producto = Producto::find($value['id']);
        $producto->stock -= $value['cantidad'];
        $producto->save();
    }

    // Logica obtener nombre para factura
    $year = now()->format('Y');
    $prefix = 'FS' . $year . '-';
    $lastInvoice = FacVenta::where('name', 'like', $prefix . '%')->orderBy('id', 'desc')->first();
    $sequence = $lastInvoice ? intval(substr($lastInvoice->name, -4)) + 1 : 1;
    $name = $prefix . str_pad($sequence, 4, '0', STR_PAD_LEFT);

    $fecha = now()->format('Y-m-d');

    $idFormaPago = $this->formaPagoSeleccionada;

    //pasar caja_id a int
    $caja_id = intval($caja_id);
}

```

- Esta es la función más larga asi que la dividire en partes:
 - En esta primera parte obtengo la variable \$usuarioid consiguiendo primero el objeto user cuando el campo name coincide con la variable \$usuarioName, luego obtengo su id. Luego recorro la tabla Producto para restar el stock de los productos.
 - En la siguiente parte obtengo el siguiente nombre de factura para introducirlo usando la misma logica que la funcion obtenerSiguienteNumeroFactura()

```

$fecha = now()->format('Y-m-d');

$idFormaPago = $this->formaPagoSeleccionada;

//pasar caja_id a int
$caja_id = intval($caja_id);

if($metodoDePago === 'tarjeta'){
    $fp = FormjasPago::firstOrCreate(
        ['name' => 'Tarjeta']
    );
    $idFormaPago = $fp->id;
    FacVenta::create(
        [
            'name' => $name,
            'tercero_id' => 1,
            'fecha' => $fecha,
            'forma_pago_id' => $idFormaPago,
            'base_imp' => $totalCarrito,
            'total_iva' => $valorIVA,
            'total' => $totalSinDesglosar,
            'caja_id' => $caja_id,
        ]
    );
}

if($metodoDePago === 'efectivo'){
    $fp = FormjasPago::firstOrCreate(
        ['name' => 'Efectivo']
    );
    $idFormaPago = $fp->id;
    FacVenta::create(
        [
            'name' => $name,
            'tercero_id' => 1,
            'fecha' => $fecha,
            'forma_pago_id' => $idFormaPago,
            'base_imp' => $totalCarrito,
            'total_iva' => $valorIVA,
            'total' => $totalSinDesglosar,
            'caja_id' => $caja_id,
        ]
    );
}

```

- En la siguiente parte paso el valor de caja_id a tipo numerico porque me estaba dando el balor como cadena de texto. Luego dependiendo del

tipo de operación que indique (efectivo o tarjeta), introducire en la tabla fac_ventas un nuevo objeto con los campos que le paso.

```

    $this->getNextRef = $name;
    // Actualizar el arqueo con las sumas de efectivo y tarjeta
    $this->updateSumaTotalEfectivo($caja_id);
    $this->updateSumaTotalTarjeta($caja_id);
    $this->updateTotalesArqueo($caja_id);

    // Refresca factura y arqueo
    $this->arqueos = Arqueo::select('id', 'fecha', 'saldo_inicial', 'saldo_efectivo', 'saldo_tarjeta', 'caja_id', 'saldo_total', 'saldo_final')->orderBy('fecha', 'DESC');
    $this->factura = Factura::orderBy('id', 'desc')->first();
    $this->facturas = Factura::select('id', 'name', 'tercer_id', 'fecha', 'forma_pago_id', 'base_imp', 'total_iva', 'total')->get()->keyBy('id')->toArray();
}

```

- En esta última parte llamo a mis funciones updateSumaTotalEfectivo() updateSumaTotalTarjeta() y updateTotalesArqueo() para actualizar los saldos de la tabla Arqueos.
- Por último refresco las llamadas a arqueos y facturas para actualizar las tablas que muestro en mi vista.

6.5.2 Gestión Inventario

Validaciones

```

protected $rules = [
    'nombre' => 'required|string',
    'precio' => 'required|numeric',
    'imagen' => 'required|image|max:2048', // 2MB Max
    'iva_id' => 'required|integer',
    'categoria_id' => 'required|integer',
    'stock' => 'required|integer',
    'estado' => 'required|boolean',
];

protected $rulesCategoria = [
    'nombreCategoria' => 'required|string',
    'imagenCategoria' => 'required|image|max:2048', // 2MB Max
];

protected $rulesEmpleado = [
    'name' => 'required|string',
    'email' => 'required|email',
    'password' => 'required|string',
    'privilegios' => 'required|integer',
    'imagen_empleado' => 'required|image|max:2048',
    'puesto_empresa' => 'required|string',
];

```

- Estas reglas de validación aseguran que los datos recibidos cumplan con ciertos criterios antes de ser procesados.

```
funcion crearProducto()
```

```
public function crearProducto()
{
    $this->validate();

    // Guardar la imagen
    $imagenPath = $this->imagen->store('productos', 'public');

    // Crear el producto
    Producto::create([
        'nombre' => $this->nombre,
        'precio' => $this->precio,
        'imagen_url' => $imagenPath,
        'iva_id' => $this->iva_id,
        'categoria_id' => $this->categoria_id,
        'stock' => $this->stock,
        'estado' => $this->estado,
    ]);

    $this->productos = Producto::select('id', 'nombre', 'precio', 'imagen_ur
    $this->nombre = ''; $this->precio = ''; $this->iva_id = ''; $this->categ
}
```

- En esta función insertamos un nuevo objeto en la tabla Productos rellenandola con los campos que pasamos por el formulario. Primero se validan los campos, luego creo una variable llamada \$imagenPath para insertar esa ruta en mi campo imagen_url. Esto creará, si no existe, una nueva carpeta llamada productos dentro de mi carpeta storage. En ella se introducirán las imágenes que posteriormente usaremos en nuestra webpara ello nos ayudamos de la función create. Por último recargamos los productos para que se actualice el listado en nuestra vista y volvemos a poner los valores del formulario vacíos, para así poder crear otro producto sin problemas.

función crearCategoria()

```
public function crearCategoria()
{
    $this->validate($this->rulesCategoria);

    $imagenPath = $this->imagenCategoria->store('categorias', 'public');

    Categoria::create([
        'nombre' => $this->nombreCategoria,
        'imagen_url' => $imagenPath,
    ]);

    $this->categorias = Categoria::select('id', 'nombre', 'imagen_url')->get()->keyBy('id')->toArray();
    $this->nombreCategoria = ''; $this->imagenCategoria = null;
}
```

- Sigue la misma lógica que crearProducto().

funcion crearEmpleado()

```
public function crearEmpleado()
{
    $this->validate($this->rulesEmpleado);

    $imagenPath = $this->imagen_empleado->store('empleados', 'public');

    User::create([
        'name' => $this->name,
        'email' => $this->email,
        'password' => bcrypt($this->password),
        'privilegios' => $this->privilegios,
        'imagen_url' => $imagenPath,
        'puesto_empresa' => $this->puesto_empresa,
    ]);

    $this->user = User::select('id', 'name', 'email', 'password', 'privilegios', 'imagen_url', 'puesto_empresa')->get()->keyBy('id')->toArray();
}
```

- Misma lógica que las anteriores. Valida, crea ruta imagen inserta los valores y recarga el listado

funcion crearIva()

```
public function crearIva($qty)
{
    Iva::create(['qty' => $qty]);
    $this->iva = Iva::select('id', 'qty')->get()->keyBy('id')->toArray();
}
```

- Esta es un poco distinta ya que no lleva imagen y como el campo del formulario es de tipo number no vimos necesario hacer una validación.

funciones actualizar

```
public function actualizarProducto($id, $nombre, $precio, $iva_id, $categoria_id, $stock, $se_vende, $imagen_url = null, $nueva_imagen = null)
{
    $producto = Producto::find($id);
    if ($producto) {
        $producto->nombre = $nombre;
        $producto->precio = $precio;
        $producto->iva_id = $iva_id;
        $producto->categoria_id = $categoria_id;
        $producto->stock = $stock;
        $producto->se_vende = $se_vende;

        if ($this->imagen) {
            $this->validate(['imagen' => 'image|max:2048']);
            $imagenPath = $this->imagen->store('productos', 'public');
            $producto->imagen_url = $imagenPath;
        }

        $producto->save();
    }

    $this->productos = Producto::select('id', 'nombre', 'precio', 'imagen_url', 'iva_id', 'categoria_id', 'stock', 'se_vende')->with(['categoria']->get()->keyBy('id')->toArray());
}
```

- Ya que todas las funciones de actualización hacen lo mismo lo resumire aquí. Primero busca en la tabla correspondiente el objeto por su id. Si lo encuentra actualiza los campos con los valores proporcionados (\$producto->nombre = \$nombre;). Si se proporciona una nueva imagen, la valida, almacena y actualiza la URL de dicha imagen. Para terminar guarda los cambios realizados y actualiza el listado en la vista.

funciones delete

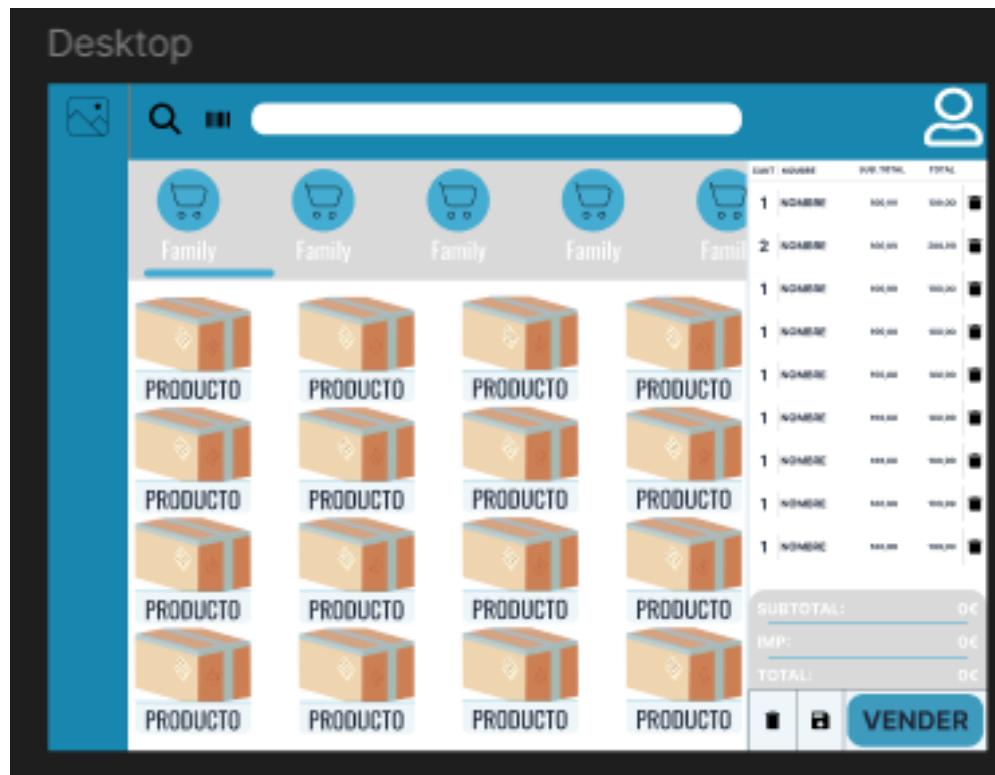
```
public function dropIva($id)
{
    $iva = Iva::find($id);
    $iva->delete();

    $this->iva = Iva::select('id', 'qty')->get()->keyBy('id')->toArray();
}
```

- Todas las funciones delete tienen la misma estructura. Busca el objeto en la tabla correspondiente ejecuta la función delete() y actualiza el listado en nuestra vista.

6.6 Diseño

6.6.1 Diseño Prototipos de alta fidelidad(Figma)



6.6.2 Elección de colores y tipografía

Para el diseño hemos optado por algo intuitivo y sencillo, para escoger los colores buscábamos algo formal y elegante, de primeras escogimos tres posibles colores, borgoña, gris y verde, la opción ganadora fue borgoña (#800020), un color bastante elegante, justo lo que teníamos en mente, pero tras aplicarlo y ver que era complicado complementar colores con éste, a pesar de haber utilizado herramientas para generar paletas de colores, nos decidimos por un gris (#4F4F4F), ya que es un color neutro y con el simple hecho de jugar

con sus diferentes tonalidades, se puede combinar muy bien. Además agregamos distintos colores por si surgía la ocasión de aplicarlos como rojo, verde, obviamente blanco... Y a partir de todos estos colores generamos distintos tonos de ellos utilizando Adobe Color. En todo momento hemos estado comprobando que el contraste sea Aa utilizando herramientas como las funciones de desarrollador de Chrome y distintas extensiones.

Además, para asegurar una experiencia óptima en diversos dispositivos y sistemas operativos, se utilizan para texto las fuentes: ui-sans-serif, system-ui, sans-serif, "Apple Color Emoji", "Segoe UI Emoji", "Segoe UI Symbol", "Noto Color Emoji" y para los iconos de font awesome 'Font Awesome 6 Brands'. Esto garantiza que el texto se muestre correctamente en todas las plataformas.

6.6.3 Accesibilidad

Vamos a explicar los criterios del primer principio del W3C (World Wide Web Consortium) que más hemos tenido en cuenta a la hora de realizar el proyecto.

6.6.3.1 Contenido no textual (1.1.1 - nivel A)

Todo contenido no textual que se presenta al usuario tiene una alternativa textual que cumple el mismo propósito, excepto en diversas situaciones. En EasyTPV todas las imágenes que hemos puesto tienen una alternativa de texto adecuada.

```
p.mx-2</b>      | 70 × 24                                                                                            |
| Color              | <input type="color"/> #FFFFFF                                                                      |
| Font               | 16px ui-sans-serif, system-ui, sans-serif, "...                                                    |
| Margin             | 0px 8px                                                                                            |
| ACCESSIBILITY      |                                                                                                    |
| Contrast           | <b>Aa</b> 12.63  |
| Name               |                                                                                                    |
| Role               | paragraph                                                                                          |
| Keyboard-focusable |                  |

## 6.7 Despliegue

Desplegado a partir de Cloudways, tanto el frontend como backend están en armonía junto con el diseño. Ha sido difícil desplegarlo con Laravel debido a que la plataforma no nos permite identificarnos como administradores en la máquina virtual.

## 7 Presupuesto

| Tarea                   | Horas | Precio |
|-------------------------|-------|--------|
| Nombre de la aplicación | 1     | 10€    |
| Diseños en FIGMA        | 10    | 100€   |
| Login y registro        | 12    | 120€   |
| Paleta de Colores       | 1     | 10€    |
| Maquetación en Alpine   | 30    | 300€   |
| Parte TPV               | 70    | 700€   |
| Parte administración    | 80    | 800€   |
| Diseño                  | 40    | 400€   |
| Documentación           | 30    | 300€   |

TOTAL: 274 horas, 2740€

## 8 Conclusiones y vías futuras

A lo largo del desarrollo del proyecto TPV, hemos logrado implementar una serie de funcionalidades clave que permiten una gestión eficiente de las ventas y el inventario. Las principales características que hemos integrado incluyen:

- Interfaz de Usuario Intuitiva: Diseñamos una interfaz amigable y fácil de usar que facilita la interacción del personal con el sistema.
- Gestión de Productos: Implementamos funcionalidades para añadir, editar y eliminar productos, así como para gestionar categorías y subcategorías.
- Procesamiento de Ventas: Desarrollamos un sistema robusto para registrar ventas, aplicar descuentos y generar recibos.
- Control de Inventario: Incluimos la capacidad de rastrear niveles de stock y generar alertas cuando los productos están a punto de agotarse.
- Reportes y Análisis: Incorporamos herramientas para generar informes de ventas, ayudando a los gerentes a tomar decisiones informadas.

### Áreas de Mejora

A pesar de los logros, identificamos varias áreas donde podemos mejorar y expandir la funcionalidad del TPV:

#### 1. Estado de las Mesas:

-Descripción: Nos gustaría añadir una funcionalidad que indique el estado de las mesas mediante colores, mostrando si están en uso, si se ha solicitado la cuenta, si el pago ha sido completado, etc.

-Estado Actual: No se implementó debido a limitaciones de tiempo.

-Plan Futuro: Implementar un sistema de gestión de mesas con codificación por colores para facilitar la gestión y aumentar la eficiencia del servicio.

## 2. Optimización de la Base de Datos:

-Descripción: Aunque la base de datos es funcional, creemos que se puede optimizar para mejorar el rendimiento.

-Estado Actual: Funciona correctamente pero con margen de mejora en la eficiencia de las consultas.

-Plan Futuro: Revisar y optimizar las consultas SQL y la estructura de la base de datos para reducir tiempos de respuesta y mejorar la escalabilidad.

## 3. Mejora en la Seguridad:

-Descripción: Aunque se han tomado medidas básicas de seguridad, siempre hay espacio para mejorar.

-Estado Actual: Seguridad estándar implementada.

-Plan Futuro: Implementar medidas adicionales como autenticación de dos factores, cifrado de datos sensibles y auditorías de seguridad regulares.

## 4. Interfaz de Usuario y Experiencia de Usuario (UI/UX):

-Descripción: Continuamente podemos mejorar la interfaz y la experiencia de usuario basándonos en el feedback recibido.

-Estado Actual: La interfaz es funcional pero puede beneficiarse de mejoras en diseño y usabilidad.

-Plan Futuro: Realizar pruebas de usabilidad y encuestas de usuario para identificar áreas de mejora y aplicar cambios basados en los resultados.

## 5- Apartado de para mayor control de horario para el usuario

## Funcionalidades Futuras

Además de las mejoras mencionadas, consideramos implementar las siguientes funcionalidades en futuras versiones del TPV:

### 1. Integración con Sistemas de Pago:

-Integrar con diversas pasarelas de pago para facilitar transacciones más rápidas y seguras de verdad.

### 2. Soporte Multilenguaje:

-Añadir soporte para múltiples idiomas, haciendo el TPV accesible a una base de usuarios más amplia.

### 3. Aplicación Móvil Complementaria:

-Desarrollar una aplicación móvil que permita a los gerentes monitorizar las ventas y el inventario en tiempo real desde cualquier lugar.

### 4. Personalización de Recibos:

-Permitir la personalización de los recibos de venta, incluyendo logotipos, mensajes personalizados y opciones de diseño.

### 5. Programa de Fidelización de Clientes:

-Implementar un sistema de fidelización que permita acumular puntos y obtener recompensas, incentivando la repetición de compras.

## **Conclusión Final**

El proyecto TPV ha avanzado significativamente, logrando la implementación de una plataforma robusta y funcional. Sin embargo, reconocemos que siempre hay margen para la mejora y la innovación. Las funcionalidades adicionales y las mejoras identificadas son esenciales para asegurar que nuestro TPV no solo cumpla con las expectativas actuales sino que también se adapte y evolucione con las necesidades futuras del mercado. Continuaremos trabajando en estas áreas para ofrecer un producto cada vez más completo y eficiente.

## **9 Bibliografía/Webgrafía.**

- Laravel Documentation. Laravel <https://laravel.com/docs/11.x/readme>
- Laravel Licencia <https://opensource.org/license/MIT>
- Tailwind Css Documentation.  
<https://tailwindcss.com/docs/installation/framework-guides>
- Tailwind Css Licencia <https://opensource.org/license/MIT>
- Alpine Documentation. Alpine.js <https://alpinejs.dev/start-here>

- Alpine Licencia Alpine.js <https://opensource.org/license/MIT>
- Bootstrap Css Documentation <https://getbootstrap.com/>
- Bootstrap Licencia <https://opensource.org/license/MIT>
- FontAwesome licencia  
[https://scripts.sil.org/cms/scripts/page.php?item\\_id=OFL\\_web](https://scripts.sil.org/cms/scripts/page.php?item_id=OFL_web)
- Imagen utilizada en el fondo de login  
<https://www.pexels.com/es-es/foto/botella-surtida-dentro-de-la-barra-941864/>
- Licencia imagen <https://www.pexels.com/es-ES/license/>  
<https://www.pexels.com/es-es/foto/botella-surtida-dentro-de-la-barra-941864/>
- Github Guide. Git <https://github.com/git-guides>
- Scrum guía. Proyectos Agiles <https://proyectosagiles.org/que-es-scrum/>