

SMARTORDER: Digitalización de pedidos y pagos en la hostelería



SmartOrder

IVÁN MARTÍNEZ OLMEDO

Desarrollo de Aplicaciones Web - 2º DAW

IES Juan Bosco

Francisco Perez Sanchez

1. Introducción

- a. Presentación del proyecto
- b. Objetivos del proyecto
- c. Justificación del proyecto

2. Análisis de requerimientos

- a. Identificación de necesidades y requerimientos
- b. Identificación de público
- c. Estudio de mercado y competencia

3. Diseño y planificación

- a. Definición de la arquitectura del proyecto
- b. Diseño de la interfaz de usuario
- c. Planificación de las tareas y los recursos necesarios

4. Implementación y pruebas

- a. Desarrollo de las funcionalidades del proyecto
- b. Pruebas unitarias y de integración
- c. Corrección de errores y optimización del rendimiento

5. Documentación

- a. Documentación técnica
- b. Documentación de usuario
- c. Manual de instalación y configuración

6. Mantenimiento y evolución

- a. Plan de mantenimiento y soporte
- b. Identificación de posibles mejoras y evolución del proyecto
- c. Actualizaciones y mejoras futuras

7. Conclusiones

- a. Evaluación del proyecto
- b. Cumplimiento de objetivos y requisitos
- c. Lecciones aprendidas y recomendaciones para futuros proyectos

8. Bibliografía y referencias

- a. Fuentes utilizadas en el proyecto
- b. Referencias y enlaces de interés

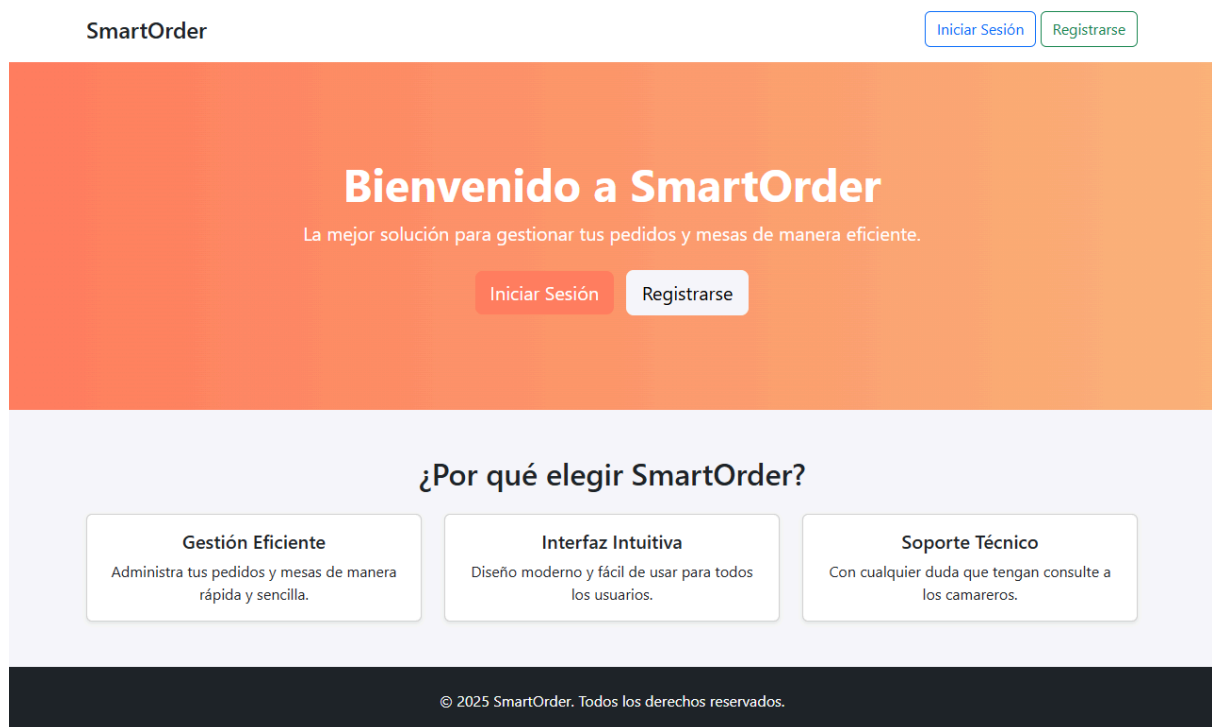
1. Introducción

1.1 Presentación del proyecto

SmartOrder es una aplicación web desarrollada para facilitar la gestión de pedidos, comandas y pagos en el ámbito de la hostelería como bares y restaurantes. Su diseño está orientado a mejorar la experiencia tanto del cliente como del personal del local, optimizando tiempos de atención y reduciendo errores en la toma de pedidos.

La plataforma permite a los clientes seleccionar una mesa, realizar su pedido desde un menú digital dividido por categorías y pagar utilizando el método que les resulte más cómodo, ya sea de forma completa o parcial según lo deseen. Por otro lado, los administradores pueden gestionar productos y mesas desde las respectivas páginas destinadas a ello.

El sistema cuenta con control de roles para garantizar la seguridad y el acceso limitado según el tipo de usuario (cliente o administrador). La interfaz es moderna, responsiva y fácil de usar, lo que permite una adopción rápida por parte de los usuarios.



1.2 Objetivos del proyecto

- Objetivo general
 - El propósito de este proyecto es el desarrollo de una aplicación moderna, intuitiva y funcional que permita gestionar de forma integral los procesos relacionados con las comandas, pedidos y pagos dentro del ámbito de la hostelería como son los bares, cafeterías o restaurantes. Esta solución busca mejorar la eficiencia operativa del establecimiento al mismo tiempo

que se optimiza la experiencia del cliente final, garantizando rapidez, comodidad y claridad durante el proceso de atención.

- La aplicación tiene como propósito digitalizar y simplificar tareas que tradicionalmente se realizan de forma manual, como la asignación de mesas, el registro de productos consumidos, la emisión de tickets y la gestión de cobros. Gracias a un sistema de roles de usuario (cliente y administrador), el sistema es capaz de limitar y personalizar las funciones visibles según el tipo de usuario, asegurando así la seguridad y el control sobre la gestión del negocio.
 - Además, se pretende que el sistema sea escalable, modular y adaptable a distintos entornos, con vistas a futuras mejoras como el control de inventario, la integración de pasarelas de pago o incluso su uso en dispositivos móviles. En definitiva, el proyecto aspira a proporcionar una herramienta útil y práctica que combine tecnología, usabilidad y productividad en el ámbito hostelero.
- **Objetivos específicos**
 - Implementar un sistema de autenticación mediante el control de roles para diferenciar entre usuarios administradores y clientes.
 - Diseñar un menú digital interactivo que permita a los clientes seleccionar los productos que quieran separados por categorías y agregarlos a su pedido.
 - Permitir a los clientes seleccionar una mesa, realizar un pedido y pagarlo de forma total o por partes.
 - Desarrollar un panel de administración en el que puedas gestionar los productos que se verán reflejados en el menú y las mesas del establecimiento.
 - Garantizar la persistencia de los datos mediante una base de datos relacional, aplicando buenas prácticas de validación y seguridad.

1.3 Justificación del proyecto

En el sector de la hostelería, la gestión de pedidos y pagos suele generar cuellos de botella, especialmente en momentos de gran cantidad de clientes. Muchos bares y restaurantes todavía dependen de métodos manuales o poco optimizados para tomar comandas y procesar pagos, lo que puede derivar en errores, tiempos de espera prolongados y una experiencia menos satisfactoria tanto para el personal como para los clientes.

SmartOrder surge como una solución tecnológica moderna que busca automatizar este proceso, facilitando la creación y gestión de pedidos desde una interfaz web accesible tanto para los camareros como para los clientes. La aplicación permite incluso dividir pagos, una funcionalidad clave que responde a una necesidad observada en entornos reales.

De hecho, **la idea del proyecto nace de una experiencia personal**: cuando mis amigos y yo íbamos a los bares a tomar algo, siempre pagábamos por separado y muchas veces comentábamos lo útil que sería tener una forma de pagar nuestra parte individual sin necesidad de esperar o desplazarnos a caja. Al estudiar desarrollo web y tener la oportunidad de aplicar estos conocimientos en un Trabajo Fin de Grado, decidí desarrollar una aplicación que hiciera posible precisamente eso. De esta forma, el proyecto no solo

responde a un problema real y cotidiano, sino que además representa un reto técnico que me ha permitido consolidar y aplicar los aprendizajes adquiridos durante la formación.

2. Análisis de requerimientos

2.1 Identificación de necesidades y requerimientos

- Antes de comenzar el desarrollo de la aplicación SmartOrder, se llevó a cabo un análisis de las necesidades básicas tanto para los clientes como los responsables del negocio. A partir de esta análisis, se identificaron los siguiente requisitos funcionales y no funcionales:
 - **Requisitos funcionales**
 - El sistema debe permitir el registro y login de usuarios.
 - El usuario debe poder seleccionar una mesa disponible.
 - El cliente debe poder añadir productos a su pedido.
 - El cliente debe poder ver el resumen de su pedido.
 - El cliente debe poder pagar productos ya sea por separado o todos a la vez.
 - El administrador debe poder gestionar productos (CRUD)
 - El administrador debe poder gestionar mesas (CRUD + liberar)
 - El sistema debe llevar un control de pedidos pagados.
 - **Requisitos no funcionales**
 - El sistema debe ser accesible desde dispositivos móviles y ordenadores.
 - La interfaz debe ser intuitiva y fácil de usar.
 - El acceso a funciones debe estar restringido según el rol del usuario.
 - El sistema debe validar entradas y mostrar mensajes claros de error.
 - El rendimiento debe ser óptimo en operaciones comunes (hacer pedido, pagar).

2.2 Identificación del público

- La aplicación SmartOrder está orientada a cubrir las necesidades específicas de dos tipos principales de usuarios, cuyas características y niveles de acceso han sido cuidadosamente diferenciados para garantizar una experiencia adecuada y segura dentro del sistema:
- **Usuario cliente**, corresponde al perfil de los clientes habituales en bares, cafeterías o restaurantes, que hacen uso del sistema desde el punto de vista del consumidor. Este tipo de usuario no requiere de conocimientos técnicos ni formación previa, ya que la interfaz está diseñada para ser intuitiva, accesible y orientada a la simplicidad de uso.
- Las acciones que puede realizar un cliente incluyen:
 - Selección de mesa de entre las disponibles
 - Visualizar el menú completo, dividido por categorías.
 - Agregar productos al pedido actual de forma individual.
 - Realizar pedido completos o pagos parciales (por productos)
 - Acceder al detalle del pedido y a su factura digital.

- Visualizar el total pagado y confirmar el cierre del pedido.
- La experiencia de uso para el cliente ha sido optimizada visualmente con botones claros, mensajes informativos y un flujo de navegación que guía paso a paso.
- **Usuario administrador**, está reservado para los empleados o gestores del establecimiento, como camareros, encargados o propietarios. A diferencia del cliente, el usuario administrador tiene acceso total al sistema y puede realizar tareas administrativas clave para la operación del negocio.
- Sus funcionalidades principales incluyen:
 - **Gestión completa de productos**: añadir, editar o eliminar artículos del menú.
 - **Gestión de mesas**: añadir, editar, eliminar o liberación de mesas ocupadas.
 - **Supervisión de pedidos**: revisar qué mesas han hecho pedidos, cuál es su estado y qué productos han sido seleccionados.
 - **Acceso a un dashboard personalizado**: desde donde puede controlar todos los elementos del sistema de forma centralizada.
- Esta separación de roles permite mantener la seguridad, reducir errores y evitar que usuarios no autorizados modifiquen datos sensibles.

2.3 Estudio de mercado y competencia

- En la actualidad existen numerosas aplicaciones móviles y web destinadas a gestionar pedidos en el sector hostelero, especialmente enfocadas al reparto de comida a domicilio. Entre las más conocidas se encuentran **Glovo, Uber Eats y Just Eat**, plataformas ampliamente utilizadas por negocios que desean extender sus ventas a través del delivery.
- No obstante, estas herramientas no están orientadas a la gestión interna de un local físico, y presentan limitaciones importantes cuando se trata de pedidos realizados en mesa o consumición en el local.

Funcionalidad	SmartOrder	Glovo	Uber Eats
Pedido desde mesa	Sí	No	No
Gestión interna de productos/mesas	Sí	No	No
Pago por separado	Sí	No	No
Interfaz personalizada por negocio	Sí	No	No
Reparto a domicilio	No	Sí	Sí

- Diferencias clave:
 - Diseño a medida para locales físicos, sin necesidad de conexión con plataformas externas.
 - Mayor control por parte del restaurante, sin depender de terceros ni pagar comisiones.
 - Funcionalidad específica de división de pagos, ideal para grupos que consumen en conjunto.
 - Flujo de uso rápido y sin complicaciones, pensado para clientes que quieren pedir y pagar sin esperas.

- En definitiva, SmartOrder está pensado para un tipo de necesidad o grupos de usuarios que no está bien atendido por las aplicaciones más grandes y populares dichas anteriormente, aportando una solución económica, autónoma y enfocada a mejorar la atención al cliente dentro del propio establecimientos.
-

3. Diseño y planificación

3.1 Definición de la arquitectura del proyecto

- La arquitectura del sistema está basada en el patrón MVC (Modelo - Vista - Controlador) proporcionado por el framework Laravel. Esta estructura permite una separación clara entre la lógica del negocio, la gestión de datos y la presentación visual de la información, lo cual mejora el mantenimiento y escalabilidad del proyecto
 - Estructura general del sistema:
 - **Modelo (app/Models):** Contiene toda la lógica relacionada con la base de datos. Aquí se definen modelos como User, Pedido, Mesa, Productos, etc.
 - **Controladores (app/Http/Controllers):** Se encarga de recibir las peticiones del usuario, procesarlas(validaciones, llamadas al modelo, etc) y retornar la vista correspondiente.
 - **Vistas (resources/views):** Plantillas Blade que representan el contenido visual que se muestra al usuario (interfaz).
 - Componentes principales:
 - **Gestión de usuario:** Registro, login, asignación de roles (cliente, administrador) y control de acceso.
 - **Gestión de productos:** CRUD de productos accesibles solo para administradores.
 - **Gestión de mesas:** Visualización y selección de mesas por parte de los clientes, y administración por parte de los administradores.
 - **Gestión de pedido:** Permite a los clientes crear pedidos asociados a su mesa, añadir productos y pagarlos.
 - **Pagos:** Sistema de pago parcial o completo, con control de productos ya abonados y generación de facturas.

3.2 Diseño de la interfaz de usuario

- El diseño de la interfaz se ha realizado con un enfoque intuitivo, simple y responsive, utilizando principalmente Bootstrap y Tailwind CSS. Se ha evitado el uso excesivo de animaciones o componentes complejos para no dificultar la experiencia del usuario, especialmente en dispositivos móviles.
 - **Dashboard:** Muestra el acceso rápido a las funciones según el rol del usuario (administrador o cliente).



Daniel ▾

Bienvenido, Daniel

Selecciona una de las opciones para gestionar tu aplicación de manera eficiente.



Ver Productos

Gestiona los productos disponibles en tu sistema.

[Ir a Productos](#)



Gestionar Mesas

Administra las mesas del restaurante.

[Ir a Mesas](#)



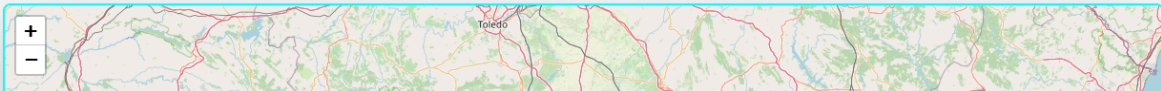
Ver Menú

Consulta el menú disponible para los clientes.

[Ir al Menú](#)

Ubicaciones del Restaurante

Consulta nuestras ubicaciones en Tomelloso y Ciudad Real:



Ivan ▾

Bienvenido, Ivan

Selecciona una de las opciones para gestionar tu aplicación de manera eficiente.



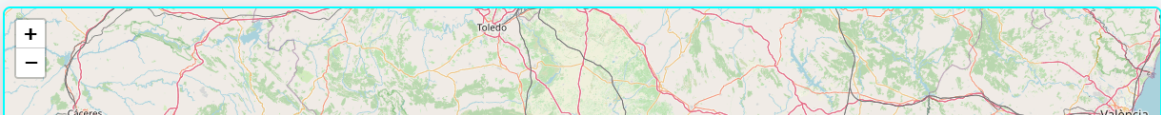
Ver Menú

Consulta el menú disponible para los clientes.

[Ir al Menú](#)

Ubicaciones del Restaurante

Consulta nuestras ubicaciones en Tomelloso y Ciudad Real:



- **Menú del restaurante:** Presenta los productos en diferentes categorías (desayuno, bebidas, bocadillos, etc), organizados en tarjetas con botones claros para añadir al pedido.

[🍽️ Seleccionar Mesa](#)

☕ Desayunos

Café con Leche - 1.80€	Agregar
Tostada con Tomate - 2.00€	Agregar
Croissant - 1.60€	Agregar
Zumo de Naranja - 2.20€	Agregar
Churros - 2.60€	Agregar

🥪 Bocadillos

Bocadillo Bacon con Queso - 4.20€	Agregar
Bocadillo de Lomo al Ajillo - 3.20€	Agregar
Bocadillo de Jamón - 4.20€	Agregar
Bocadillo de Calamares - 4.80€	Agregar
Bocadillo de Atún - 4.50€	Agregar
Bocadillo Vegetal - 4.30€	Agregar

🥙 Sándwiches


Sándwich de Pollo y Queso - 4.80€	Agregar
Sándwich Vegetal con Huevo - 4.50€	Agregar
Sándwich Mixto - 3.90€	Agregar
Sándwich de Atún y Tomate - 4.20€	Agregar

🍰 Postres

Tarta de Queso - 3.50€	Agregar
Flan Casero - 2.80€	Agregar
Helado de Vainilla - 2.50€	Agregar
Brownie con Helado - 3.90€	Agregar

[← Anterior](#)[Siguiente →](#)


- **Selector de mesa:** Permite al usuario elegir una mesa disponible antes de poder interactuar con el sistema.



Daniel ▾

Seleccionar Mesa


Por favor, selecciona una mesa para continuar con tu pedido.



Mesa 1

Disponible


Seleccionar



Mesa 2

Disponible


Seleccionar



Mesa 3

Disponible


Seleccionar



Mesa 4

Disponible


Seleccionar



Mesa 5

Disponible


Seleccionar



Mesa 6

Disponible


Seleccionar



Mesa 7

Disponible

Seleccionar



Mesa 8

Disponible

Seleccionar

← Volver al Menú

- **Pedido actual:** Vista que muestra los productos seleccionados, opción para eliminar, hacer pedido y cuenta atrás antes del envío.

Mi Pedido

Revisa tus productos y confirma tu pedido

Volver al Menú

Productos seleccionados

Bocadillo de Jamón 4.20€

Eliminar

Zumo de Naranja 2.20€

Eliminar

Sándwich de Atún y Tomate 4.20€

Eliminar

Sándwich Vegetal con Huevo 4.50€

Eliminar

Resumen del pedido

Total: 15.10€

Confirmar Pedido

- **Factura:** Muestra todos los productos pagados, el total y permite volver al menú o seguir pagando si quedan productos pendientes.

FACTURA DE PAGO

Gracias por tu visita

Mesa: 5

Fecha: 02/06/2025

Hora: 16:21

Productos Pagados

Bocadillo de Jamón (x1)	4.20 €
Zumo de Naranja (x1)	2.20 €
Sándwich de Atún y Tomate (x1)	4.20 €
Sándwich Vegetal con Huevo (x1)	4.50 €

Total Pagado: 15.10 €

✓ Finalizar

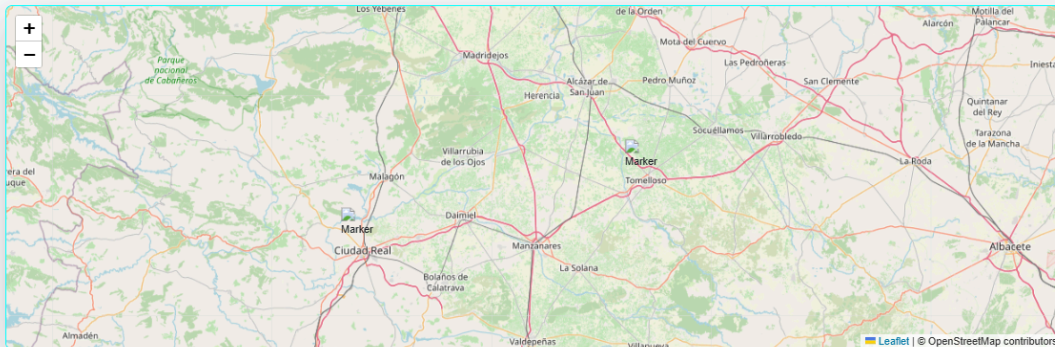
- Además de las vistas funcionales ya descritas, se ha incorporado al dashboard principal un componente interactivo de geolocalización. Este mapa, desarrollado con la librería **LeafletJS**, permite visualizar múltiples ubicaciones físicas del restaurante en tiempo real.
- Su integración ha sido diseñada de forma que no interfiera con las funcionalidades principales del dashboard, conservando el estilo visual moderno de la interfaz. El mapa se adapta correctamente tanto a pantallas grandes como móviles, y está incrustado como un contenedor responsive y accesible. Actualmente, el mapa muestra 2 ubicaciones del restaurante:
 - “Av. La Mancha, esquina 150, Av. Juan Carlos I, 13700 Tomelloso, Cuidad Real”

- “Av.Lagunas de Ruidera, 8, 13004 Ciudad Real”
- Estas ubicaciones se muestran mediante marcadores personalizados, con mensajes emergentes para mejorar la orientación del cliente. Esta funcionalidad añade valor visual al sistema y refuerza la idea de que SmartOrder es una solución aplicable a negocios reales y con varias sucursales.



Ubicaciones del Restaurante

Consulta nuestras ubicaciones en Tomelloso y Ciudad Real:



3.3 Planificación de tareas y recursos

- Durante la fase de desarrollo se siguió una planificación flexible basada en metodología incremental, donde cada funcionalidad se desarrolló y probó antes de avanzar a la siguiente. A continuación se muestra una aproximación a la secuencia de tareas realizadas:
 - **Análisis y definición de requisitos**
 - Identificación de funcionalidades clave
 - Diseño inicial del flujo de uso
 - **Configuración del entorno de desarrollo**
 - Instalación de **Laravel**, **XAMPP**, **Composer**, [Node.js](#)
 - Estructura del proyecto base.
 - **Diseño de la base de datos**
 - Migraciones de tablas (**users**, **mesas**, **productos**, **pedidos**, **detalle_pedidos**, **tipo_to_users**)
 - **Autenticación y roles**
 - Registro e inicio de sesión
 - Añadir campo **tipo** al modelo **User**
 - Middleware personalizado para restringir acceso
 - **Desarrollo de vistas cliente**

- Menú, pedido, ver pedido, pagar parcial o completo
 - **Desarrollo de vistas administrador**
 - **CRUD** de productos y mesas
 - **Liberación** de mesas y control del sistema
 - **Pruebas y validaciones**
 - Validación de formularios
 - Pruebas de errores (sin mesa, sin productos, sin selección, etc)
 - **Corrección de errores y optimización**
 - Manejo de sesiones
 - Mejoras de seguridad
 - Eliminación de navegación no deseada
 - Recursos utilizados
 - **Editor.** Visual Studio Code
 - **Base de datos:** MySQL (con phpMyAdmin)
 - **Framework:** Laravel 10
 - **Front-end:** Bootstrap, Tailwind CSS
 - **Control de versiones:** Git-Hub
 - **Navegador:** Google Chrome
 - **Sistema de logs:** Laravel Log
-

4. Implementación y pruebas

4.1 Desarrollo de funcionalidades

- Durante la fase de implementación se desarrollaron las funcionalidades principales del sistema SmartOrder. Estas funcionalidades se organizaron y estructuraron en controladores específicos, utilizando el patrón MVC de Laravel.
 - **Autenticación y registro de roles.** El sistema de autenticación se desarrolló utilizando los mecanismos de **Laravel Breeze**, permitiendo el registro, login y logout de usuario. Una funcionalidad clave añadida fue el campo “Tipo de usuario”, el cual permite distinguir entre administradores y clientes durante el registro.
 - Dependiendo del valor de este campo (admin o cliente), el usuario tendrá acceso a funcionalidades diferentes. Esto se gestiona mediante:
 - Middleware personalizado (EsAdmin) para proteger rutas exclusivas del administrador
 - Métodos en el modelo User (esAdmin() y esCliente()) para validar el rol actual.

Crear Cuenta

Nombre

Correo Electrónico

Tipo de Usuario

Cliente ▼

Contraseña

Confirmar Contraseña

[¿Ya tienes una cuenta?](#)

Registrarse



pilar ▼

Bienvenido, pilar

Selecciona una de las opciones para gestionar tu aplicación de manera eficiente.



Ver Productos

Gestiona los productos disponibles en tu sistema.

[Ir a Productos](#)



Gestionar Mesas

Administra las mesas del restaurante.

[Ir a Mesas](#)



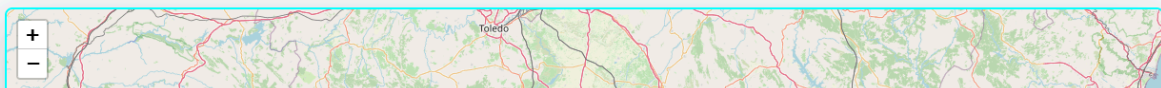
Ver Menú

Consulta el menú disponible para los clientes.

[Ir al Menú](#)

Ubicaciones del Restaurante

Consulta nuestras ubicaciones en Tomelloso y Ciudad Real:





Jose ▾

Bienvenido, Jose

Selecciona una de las opciones para gestionar tu aplicación de manera eficiente.



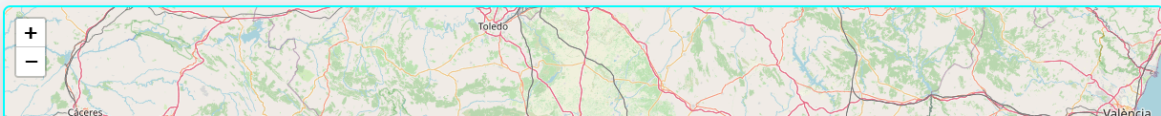
Ver Menú

Consulta el menú disponible para los clientes.

[Ir al Menú](#)

Ubicaciones del Restaurante

Consulta nuestras ubicaciones en Tomelloso y Ciudad Real:



- **Gestión de pedidos.** Los clientes pueden consultar el menú, seleccionar productos y realizar pedidos. El flujo de estas funcionalidades:
 - El cliente selecciona una mesa disponible.
 - Añade productos al pedido desde el menú.
 - Confirma el pedido (con cuenta regresiva)
 - El sistema crea un nuevo pedido en la base de datos y lo asocia a la mesa y al usuario.
- **Pagos.** Una vez realizado el pedido, se habilita la vista de Confirmación de Pedido, donde el usuario puede:
 - Pagar productos seleccionados (uno a uno o varios).
 - Pagar todo el pedido de una vez
- El sistema registra cada pago en sesión para evitar dobles pagos y actualizar el estado del pedido a Pagado cuando los productos han sido abonados.

Confirmación del Pedido

Productos Pendientes de Pago

<input type="checkbox"/> Helado de Vainilla	2.50 €
<input type="checkbox"/> Brownie con Helado	3.90 €
<input type="checkbox"/> Bocadillo de Tortilla	4.30 €
<input type="checkbox"/> Sándwich Mixto	3.90 €
<input type="checkbox"/> Tostada con Tomate	2.00 €
<input type="checkbox"/> Croissant	1.60 €


Pagar Seleccionados

Pagar Todo

- **Liberación de mesa.** Al pagarse el pedido por completo, la mesa se libera automáticamente. También se proporciona un botón de “Liberar mesa” para el administrador
- Entre las funcionalidades añadidas recientemente destaca la integración de un sistema de localización en el dashboard, que permite mostrar visualmente las ubicaciones físicas del restaurante.
- Esta funcionalidad se ha desarrollado usando **Leaflet** y se inicializa automáticamente con coordenadas predefinidas, mostrando dos puntos de interés sobre el mapa.
- La lógica del mapa se carga junto con la vista y está optimizada para no interferir con las demás tarjetas funcionales del dashboard. Este componente se ha diseñado de forma que se adapte correctamente tanto a pantallas de escritorio como a dispositivos móviles.

4.2 Pruebas unitarias y de integración

- Durante la fase de desarrollo se realizaron pruebas manuales para validar los distintos flujos de uso, tanto del cliente como del administrador.
 - Métodos de prueba utilizados
 - Pruebas funcionales con interacción directa en el navegador.
 - Comprobación de mensajes de errores y alertas.
 - Revisión de sesiones, redirecciones y middleware.
 - Casos de prueba destacados
 - Agregar productos sin seleccionar mesa. El sistema muestra una alerta informando que es obligatorio seleccionar una mesa antes de realizar un pedido.




pilar ▾

Nuestra Carta


Descubre nuestras deliciosas opciones

Seleccionar Mesa


Por favor, selecciona una mesa antes de agregar productos.

 Desayunos

Café con Leche 1.80€	+
Tostada con Tomate 2.00€	+
Croissant 1.60€	+
Zumo de Naranja 2.20€	+
Churros 2.60€	+


 Bocadillos

Bocadillo Bacon con Queso 4.20€	+
Bocadillo de Lomo al Ajillo 3.20€	+
Bocadillo de Jamón 4.20€	+
Bocadillo de Calamares 4.80€	+
Bocadillo de Atún 4.50€	+

 Sándwiches

Sándwich de Pollo y Queso 4.80€	+
Sándwich Vegetal con Huevo 4.50€	+
Sándwich Mixto 3.90€	+
Sándwich de Atún y Tomate 4.20€	+
Sandwinch de Bacon con Queso 4.35€	+

- **Ver pedido sin seleccionar mesa.** Similar al caso anterior, se bloquea el acceso a la vista de pedido.




pilar ▾

Nuestra Carta


Descubre nuestras deliciosas opciones

Seleccionar Mesa


Por favor, selecciona una mesa antes de ver el pedido.

 Desayunos

Café con Leche 1.80€	+
Tostada con Tomate 2.00€	+
Croissant 1.60€	+
Zumo de Naranja 2.20€	+
Churros 2.60€	+

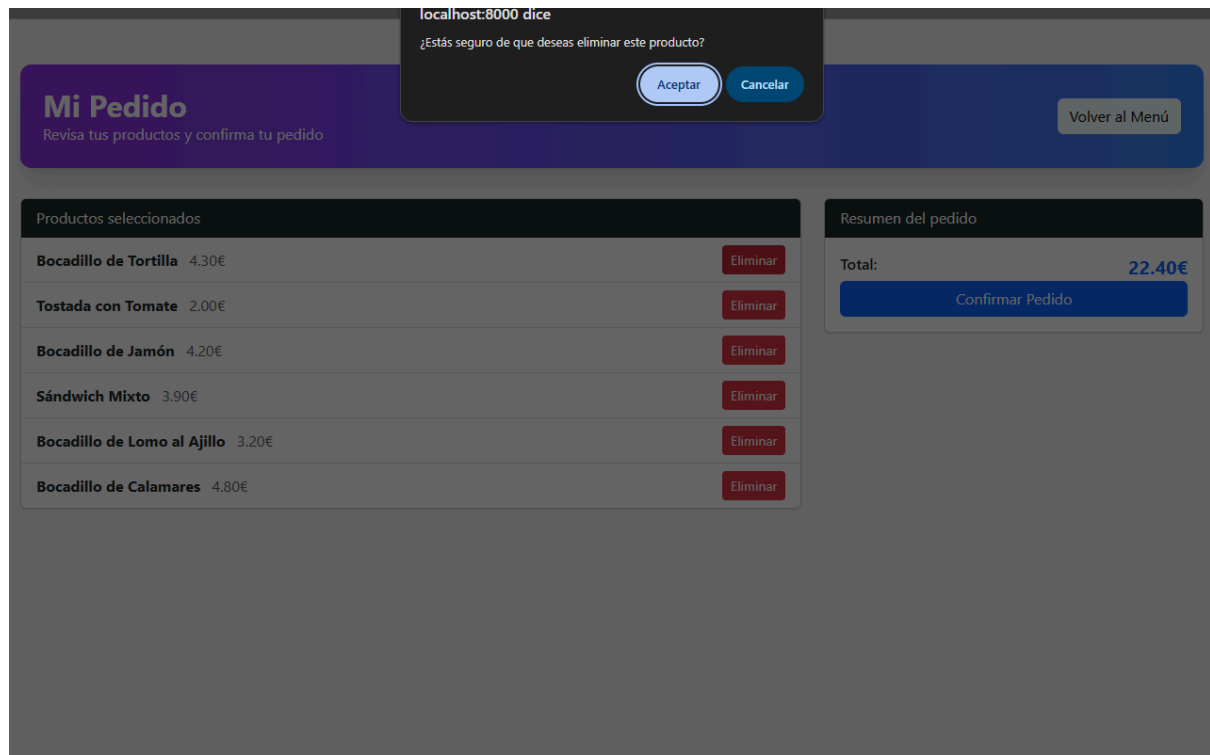
 Bocadillos

Bocadillo Bacon con Queso 4.20€	+
Bocadillo de Lomo al Ajillo 3.20€	+
Bocadillo de Jamón 4.20€	+
Bocadillo de Calamares 4.80€	+
Bocadillo de Atún 4.50€	+

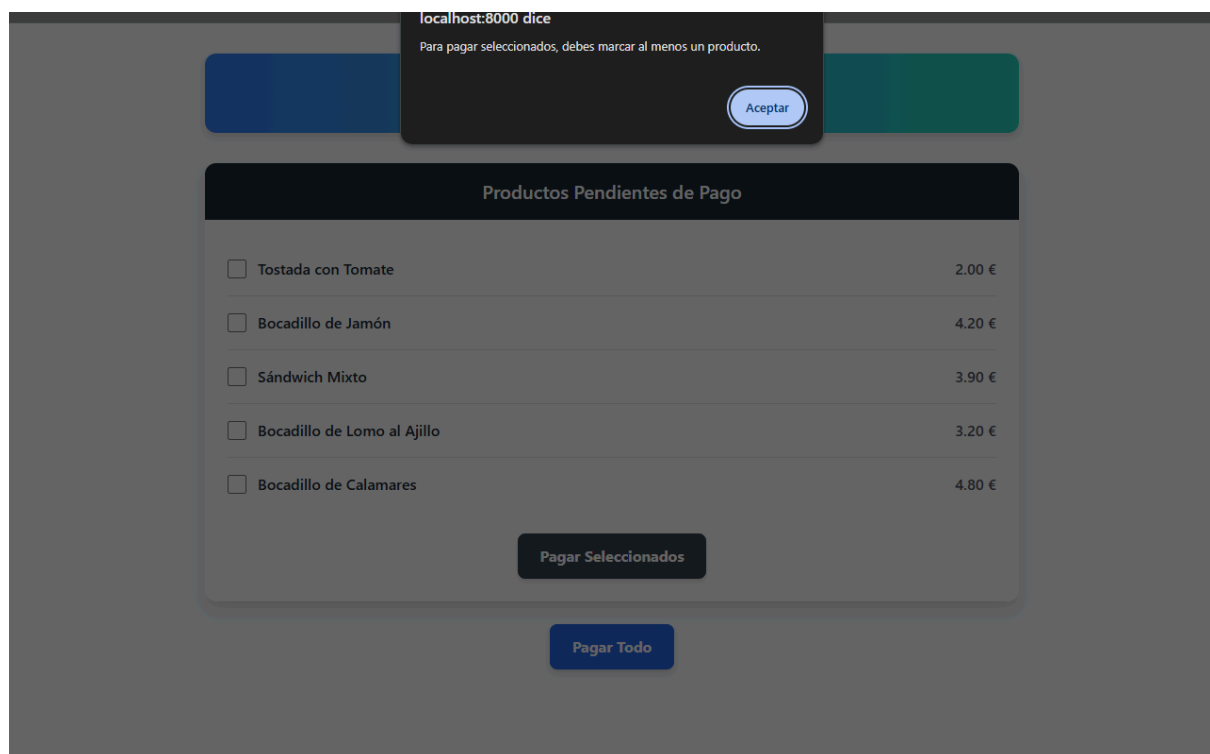
 Sándwiches

Sándwich de Pollo y Queso 4.80€	+
Sándwich Vegetal con Huevo 4.50€	+
Sándwich Mixto 3.90€	+
Sándwich de Atún y Tomate 4.20€	+
Sandwinch de Bacon con Queso 4.35€	+

- **Eliminar producto del pedido.** Al intentar eliminar un producto del pedido, aparece un mensaje de confirmación.



- **Intentar pagar sin productos seleccionados.** El sistema no permite enviar el formulario y muestra una alerta.



- **Pagos correctos.** Una vez seleccionados los productos, el sistema permite pagar y redirige a la vista de factura.

FACTURA DE PAGO

Gracias por tu visita

Mesa: 5

Fecha: 02/06/2025

Hora: 16:36

Productos Pagados

Tostada con Tomate (x1)	2.00 €
Bocadillo de Jamón (x1)	4.20 €
Sándwich Mixto (x1)	3.90 €
Bocadillo de Lomo al Ajillo (x1)	3.20 €
Bocadillo de Calamares (x1)	4.80 €

Total Pagado: 18.10 €

✓ Finalizar

4.3 Corrección de errores y optimización del rendimiento

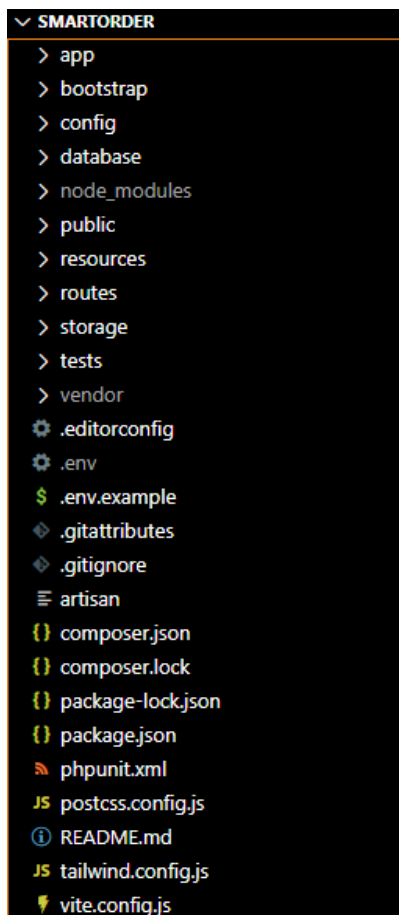
- Durante el desarrollo se detectaron varios errores que se resolvieron conforme se iba avanzando en el proyecto. A continuación voy a listar los errores más frecuentes que he tenido:
 - El botón “**Volver al Menú**” no se desactivaba tras hacer pedido, esto permitía que el usuario interrumpiera el flujo de creación del pedido y causara inconsistencias en sesión.
 - **Selección múltiple de mesas**, el sistema permitía seleccionar varias mesas seguidas, lo que provocaba errores en el pedido.

- **El pedido no se registraba en la base de datos**, aunque se mostraba el mensaje ¡Tu pedido está listo!, no se guardaba en la base de datos porque el método no estaba bien definido.
- **El id de la mesa se establecía por defecto como 1**, esto se debía a que el valor de la mesa no se recuperaba correctamente de la sesión.
- **Cómo los resolviste**
 - En el script hice que el botón se eliminase y que se volviera a mostrar una vez completado el proceso.
 - Se implementó una validación para que si una mesa ya estaba seleccionada, no se pudiera volver a seleccionar otra hasta finalizar el pedido.
 - Se corrigió el método HTTP del formulario para que enviara los datos correctamente al backend.
 - Se corrigió la recuperación del **mesa_id** usando **session('mesa_id')** en vez de valores fijos.
- **Mejoras de rendimiento.** Además de resolver errores funcionales, se aplicaron varias medidas destinadas a mejorar el rendimiento y la eficiencia general de la aplicación:
 - **Eliminación de peticiones innecesarias:** Se optimizó el flujo de interacción entre cliente y servidor evitando recargas innecesarias y validaciones duplicadas. Por ejemplo, se implementaron validaciones en el fronted mediante JavaScript que impiden el envío de formulario incompletos, como intentar pagar sin haber seleccionado productos. Esto reduce la carga sobre el servidor y mejora la experiencia del usuario.
 - **Optimización del uso de sesiones:** Se revisó el uso de variables de sesión con el fin de evitar acumulaciones innecesarias de datos. Al finalizar un pedido o liberar una mesa, se eliminan correctamente las variables de sesión asociadas (pedido_x, total_x, mesa_id, etc), asegurando así un uso eficiente de la memoria del servidor y previniendo errores como la persistencia de pedidos antiguos.
 - **Minimización del tiempo de carga con recursos CDN:** Para mejorar la velocidad de carga de la aplicación, se optó por utilizar recursos desde redes de distribución de contenido (CDN), como Bootstrap o jQuery. Esto reduce el tiempo de descarga de archivos estáticos y permite al navegador reutilizar versiones cacheadas de dichos recursos, como por ejemplo: **<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">**
 - **Desactivación de componentes innecesarios en vistas específicas:** Con el objetivo de reducir la carga visual y técnica en algunas vistas específicas (como la de factura, confirmación de pedido o vista de pago), se ocultaron elementos estructurales como la barra de navegación superior. Esta medida mejora la legibilidad y enfoca la atención del usuario únicamente en la información relevante.

5. Documentación

5.1 Documentación técnica

- Estructura general del proyecto
 - La aplicación ha sido construida sobre el framework **Laravel**, una de las herramientas más utilizadas para el desarrollo de aplicaciones web modernas en PHP. Laravel proporciona una estructura de directorios modular que separa claramente cada responsabilidad del sistema, lo que facilita su mantenimiento, escalabilidad y entendimiento por parte de otros desarrolladores. Cuando se crea un proyecto Laravel, automáticamente se genera una arquitectura con las siguientes carpetas.



- Sus carpetas más destacadas son:
 - **app/Http/Controllers**
 - Aquí se encuentran los controladores de la aplicación. Cada controlador es responsable de manejar la lógica de negocio relacionada con una funcionalidad específica, como gestionar productos, pedidos o mesas. En este proyecto, hay controladores diferenciados para usuarios administradores (por ejemplo, **AdminProductoController**, **MesaController**) y para la lógica general del cliente (**PedidoController**, **ProductoController**). Además dentro de este directorio también se encuentran los middlewares personalizados como **EsAdmin**, encargado de restringir el acceso a ciertos recursos.



Acceso denegado

Lo sentimos, esta sección está reservada solo para administradores.

[Volver al panel](#)

■ resources/views

- Esta carpeta contiene todas las vistas Blade, que son plantillas HTML con sintaxis específica de Laravel para mostrar información dinámica.
- Las vistas están organizadas en subdirectorios, como admin/ para los paneles de administración (gestión de productos y mesas), y otros como layouts/, donde está el diseño base que se reutiliza (navegación, encabezado, etc).

■ routes/web.php

- Este archivo define todas las rutas web de la aplicación vinculando cada URL a un controlador o vista. Además, se han protegido rutas específicas con middleware como **auth** o **esadmin** para garantizar el acceso solo a usuarios autenticados y con permisos.

```
Route::middleware(['auth', 'esadmin'])->prefix('admin')->name('admin.')->group(function () {
    // Mesas
    Route::get('/mesas', [MesaController::class, 'index'])->name('mesas.index');
    Route::get('/mesas/create', [MesaController::class, 'create'])->name('mesas.create');
    Route::post('/mesas', [MesaController::class, 'store'])->name('mesas.store');
    Route::get('/mesas/{mesa}/edit', [MesaController::class, 'edit'])->name('mesas.edit');
    Route::put('/mesas/{mesa}', [MesaController::class, 'update'])->name('mesas.update');
    Route::delete('/mesas/{mesa}', [MesaController::class, 'destroy'])->name('mesas.destroy');
});

Route::middleware(['auth', 'esadmin'])->prefix('admin')->name('admin.')->group(function () {
    // Productos
    Route::get('/productos', [AdminProductoController::class, 'index'])->name('productos.index');
    Route::get('/productos/create', [AdminProductoController::class, 'create'])->name('productos.create');
    Route::post('/productos', [AdminProductoController::class, 'store'])->name('productos.store');
    Route::get('/productos/{producto}/edit', [AdminProductoController::class, 'edit'])->name('productos.edit');
    Route::put('/productos/{producto}', [AdminProductoController::class, 'update'])->name('productos.update');
    Route::delete('/productos/{producto}', [AdminProductoController::class, 'destroy'])->name('productos.destroy');
});
```

■ .env y config/

- El archivo **.env** guarda las variables de entorno como la conexión a base de datos, claves secretas o puertos.

- **config/** centraliza todos los ajustes globales del sistema (bases de datos, caché, mail).
 - **Base de datos y migraciones**
 - En **database/migrations** se encuentran los archivos que definen la estructura de cada tabla (usuarios, productos, pedidos, etc).
- Archivos importantes y su función. En el desarrollo del proyecto SmartOrder, se han identificado varios archivos y funciones clave que forman el núcleo del sistema. A Continuación se describen dos de los más relevantes por su papel fundamental en la lógica de negocio de la aplicación.
- Esta función es la encargada de procesar el pedido que el usuario ha configurado desde el menú. Su propósito es crear un nuevo pedido y almacenarlo en la base de datos junto con los productos seleccionados.
- Explicación del flujo de esta función.
 - **Validación de la sesión:** Se comprueba si el usuario ha seleccionado una mesa previamente (usando la sesión mesa_id). Si no lo ha hecho, se le redirige al menú con un mensaje de error.
 - **Validación de productos:** Se verifica que el usuario ha añadido al menos un producto. Si no hay productos, también se muestra un error.
 - **Cálculo del total:** Se calcula el total del pedido sumando los precios de los productos seleccionados.
 - **Creación del pedido:** Si todo es correcto, se crea una nueva entrada en la tabla pedido y se asocia al usuario y a la mesa.
 - **Inserción de productos:** Se recorre el array de productos para guardarlos en la tabla **detalle_pedidos** con su producto_id, cantidad y subtotal.
 - **Manejo de errores:** Si ocurre un error en el proceso, se captura la excepción y se muestra un mensaje indicando el problema.
- Esta función representa uno de los pilares del proyecto, al ser la encargada de convertir la interacción del usuario en una operación persistente y coherente en la base de datos.

```

public function hacerPedido(Request $request)
{
    if (!session()->has('mesa_id')) {
        return redirect()->route('menu')->with('error', 'Debe seleccionar una mesa antes de hacer un pedido.');
```

```
    }

    $mesa_id = session('mesa_id'); // Solo desde sesión

    $productos = json_decode($request->input('productos'), true);

    if (empty($productos)) {
        return redirect()->back()->with('error', 'No hay productos en el pedido.');
```

```
    }

    $total = array_sum(array_column($productos, 'precio'));

    try {
        Log::info('Iniciando creación de pedido para mesa ' . $mesa_id);

        $nuevoPedido = Pedido::create([
            'mesa_id' => $mesa_id,
            'user_id' => auth()->id(),
            'total' => $total,
            'estado' => 'Pendiente',
        ]);

        Log::info('Pedido creado con ID: ' . $nuevoPedido->id);

        foreach ($productos as $producto) {
            $nuevoPedido->detalles()->create([
                'producto_id' => $producto['id'],
                'cantidad' => 1,
                'subtotal' => $producto['precio'],
            ]);
        }

        return redirect()->route('confirmar.pedido', $nuevoPedido->id);
    } catch (\Exception $e) {
        Log::error('Error al crear el pedido: ' . $e->getMessage());
        return redirect()->back()->with('error', 'Hubo un problema al guardar el pedido. Seleccione una mesa.');
```

```
    }
}

```

- Esta función se ejecuta cuando un cliente selecciona una mesa al iniciar su experiencia en la app. Esto es obligatorio antes de realizar cualquier pedido.
- Explicación del flujo de esta función:
 - **Búsqueda de la mesa:** A través de findOrFail, se recupera la mesa seleccionada según el id enviado en el formulario.
 - **Comprobación de estado:** Si la mesa ya está ocupada (estado === 'Ocupada'), se impide continuar y se muestra un mensaje de advertencia al usuario.
 - **Cambio de estado:** Si la mesa está libre, se marca como "Ocupada" y se guarda el cambio en la base de datos.
 - **Guardado en sesión:** El id de la mesa se guarda en la sesión del cliente para poder vincularlo a futuros pedidos.
 - **Redirección al menú:** Finalmente, se redirige al menú y se notifica que la selección fue exitosa.
- Esta función permite controlar que un cliente solo pueda usar una mesa a la vez , evitando solapamientos o errores en el sistema.


```
//FUNCION QUE NOS DEJA SELECCIONAR UNA MESA ANTES DE HACER EL PEDIDO
public function seleccionarMesa(Request $request)
{
    $mesa = Mesa::findOrFail($request->mesa_id);

    if ($mesa->estado === 'Ocupada') {
        | return redirect()->back()->with('mensaje', 'Esa mesa ya está ocupada. Por favor, elige otra.');
    }

    // Marcar la mesa como ocupada
    $mesa->estado = 'Ocupada';
    $mesa->save();

    // Guardar la mesa en la sesión del cliente
    session(['mesa_id' => $mesa->id]);

    return redirect()->route('menu')->with('mensaje', 'Mesa seleccionada correctamente.');
}
}
```

- Esta función permite al cliente pagar únicamente una parte de los productos que ha pedido, es decir, realizar un pago parcial o “dividir cuenta”. Es ideal para cuando varios comensales desean abonar una parte individual.
- Explicación del flujo de esta función:
 - **Validación de productos seleccionados:** Se recuperan los productos que el cliente ha marcado para pagar mediante input('items'). Si no hay ningún producto seleccionado, el sistema redirige al usuario a un mensaje de error: **“Para pagar por separado, debe seleccionar al menos un producto.”**
 - **Recuperación y filtrado de detalle:** Se consultan en base de datos los DetallePedido correspondiente a los IDs seleccionados. Se construye un array pagados con nombre, precio y cantidad de cada productos.
 - **Cálculo del total parcial:** Se suman los subtotales de los productos seleccionados para mostrar la cantidad a pagar.
 - **Gestión de sesión de productos pagados:** Se almacena en la sesión del usuario qué productos ya han sido pagados para excitar duplicidades (session('pagados_{pedido_id}')). La sesión se actualiza cada vez que el cliente paga una nueva parte.
 - **Finalización automática del pedido (si aplica):** Si se detecta que todos los productos del pedido han sido pagados, se marca el pedido como Pagado. En ese caso también se libera automáticamente la mesa y se limpian las variables de sesión asociadas al pedido.
 - **Renderizado de la factura parcial:** Se devuelve la vista pagar_seleccionado mostrando el desglose y el total de los productos pagados en esa operación.
- Esta función representa una característica avanzada del sistema, al permitir un grado de flexibilidad que normalmente no está disponible en sistemas simples de pedido. Mejora la experiencia del cliente permitiendo pagos individuales.

```

public function pagarSeparado(Request $request)
{
    $detalleIds = $request->input('items', []);

    if (empty($detalleIds)) {
        return redirect()->back()->with('error', 'Para pagar por separado, debe seleccionar al menos un producto.');
```

```

    }

    $detalles = DetallePedido::with(['producto', 'pedido'])->whereIn('id', $detalleIds)->get();

    $pagados = $detalles->map(function ($detalle) {
        return [
            'nombre' => $detalle->producto->nombre,
            'precio' => $detalle->subtotal,
            'cantidad' => $detalle->cantidad,
        ];
    });

    $total = $pagados->sum('precio');
    $pedido = $detalles->first()->pedido;
    $mesa_id = $pedido->mesa_id;

    // Guardar los IDs pagados en sesión
    $yaPagados = session('pagados_' . $pedido->id, []);
    $yaPagados = array_merge($yaPagados, $detalleIds);
    session(['pagados_' . $pedido->id => $yaPagados]);

    // Verificar si ya se pagó todo el pedido
    $totalDetalles = $pedido->detalles()->count();
    if (count($yaPagados) >= $totalDetalles) {
        $pedido->estado = 'Pagado';
        $pedido->save();

        // Liberar la mesa
        $mesa = $pedido->mesa;
        $mesa->estado = 'Disponible';
        $mesa->save();

        // Limpiar la sesión del pedido
        session()->forget("pedido_" . $mesa->id);
        session()->forget("total_" . $mesa->id);
        session()->forget("pagados_" . $pedido->id);
        session()->forget("mesa_id");
    }

    return view('pagar_seleccionados', [
        'pagados' => $pagados,
        'total' => $total,
        'pedido_id' => $pedido->id,
        'mesa_id' => $mesa_id
    ]);
}

```

- Esta función permite al usuario pagar todos los productos pendientes del pedido en una única acción, es decir, realizar el pago total de toda la cuenta.
- Explicación del flujo de esta función:
 - **Recuperación del pedido:** Se obtiene el pedido por su pedido_id y se cargan sus relaciones (productos y detalles).
 - **Filtrado de productos aún no pagados:** A través de la sesión pagados_{pedido_id}, se detecta qué productos aún no han sido pagados (útil si hubo pagos parciales anteriores). Si todos los productos ya estaban pagados, se redirige al menú con el mensaje: **"Todos los productos ya están pagados."**

- **Preparación de la factura completa:** Se construye un array con los productos pendientes, mostrando su nombre, precio y cantidad. Se calcula el total sumando los subtotales de todos los productos.
- **Actualización de estado del pedido:** Si tras este pago se completan todos los productos, el estado del pedido se actualiza como Pagado.
- **Liberación de la mesa:** La mesa asociada al pedido se marca como disponible automáticamente. Se eliminan las variables de sesión asociadas al pedido.
- **Vista de factura general:** Se muestra al cliente una factura completa con el resumen de todos los productos pagados mediante la vista pagar_todo.
- Esta función garantiza que el sistema cubra el flujo tradicional de pago completo, permitiendo una gestión eficiente de los pedidos y cerrando correctamente la operación tanto en base de datos como a nivel de interfaz.

```

public function pagarTodo(Request $request)
{
    $pedido = Pedido::with('detalles.producto')->findOrFail($request->pedido_id);
    // Obtener detalles ya pagados desde sesión
    $yaPagados = session('pagados_' . $pedido->id, []);
    // Filtrar los detalles no pagados
    $pendientes = $pedido->detalles->whereNotIn('id', $yaPagados);

    if ($pendientes->isEmpty()) {
        return redirect()->route('menu')->with('mensaje', 'Todos los productos ya están pagados.');
```

5.2 Documentación de usuario

- Qué ve y hace un usuario cliente
 - El cliente representa el perfil de usuario común del restaurante. Su experiencia ha sido diseñada para ser lo más fluida, visual y sencilla posible, minimizando pasos y evitando sobrecarga de información.
 - Acciones permitidas al cliente:
 - Acceder a la vista de Menú, donde verá los productos disponibles divididos en categorías.

- Seleccionar una mesa disponible antes de empezar el pedido. Esto garantiza que los pedidos se asocien a una mesa concreta.
- Añadir productos al pedido en curso pulsando un botón. Cada producto incluye nombre, precio y categoría.
- Cambiar entre páginas de menú si hay muchos productos.
- Visualizar el pedido actual, donde podrá:
 - Ver productos agregados con su precio.
 - Eliminar productos.
 - Ver el total acumulado en tiempo real



Jose ▾

Bienvenido, Jose

Selecciona una de las opciones para gestionar tu aplicación de manera eficiente.



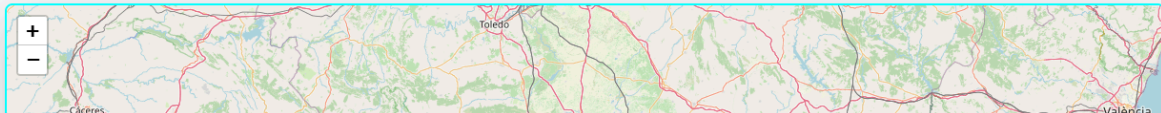
Ver Menú

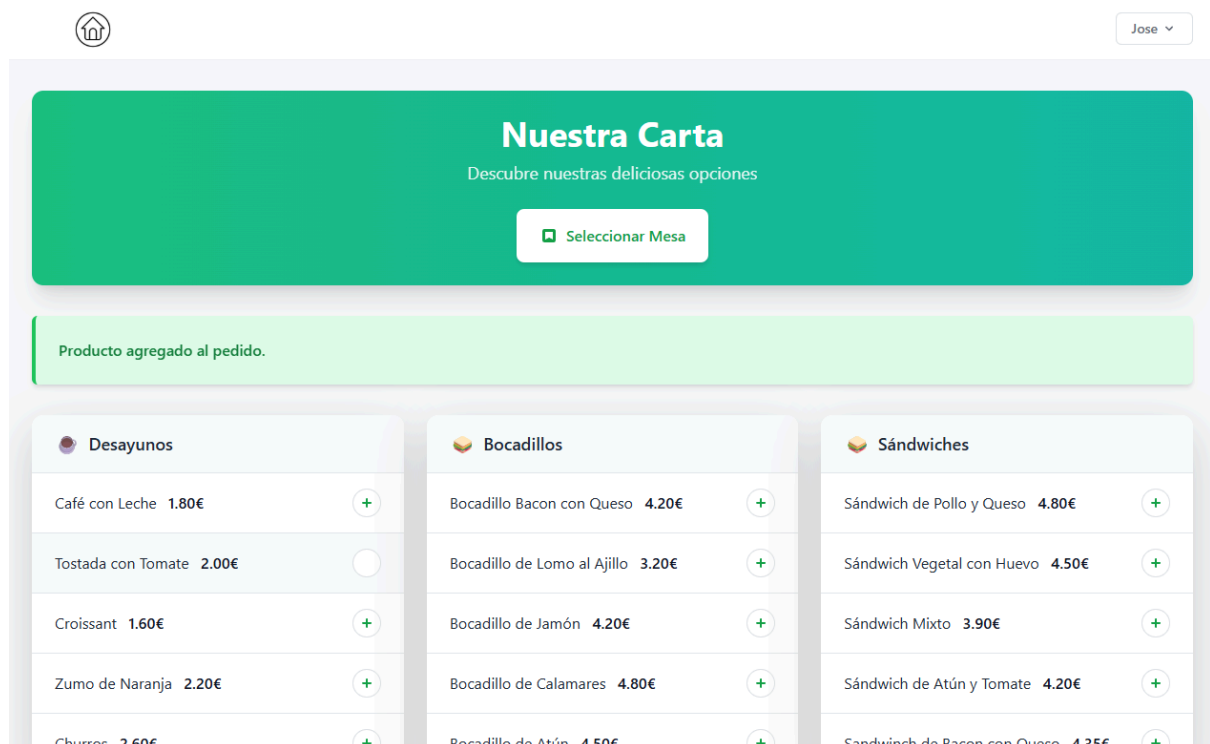
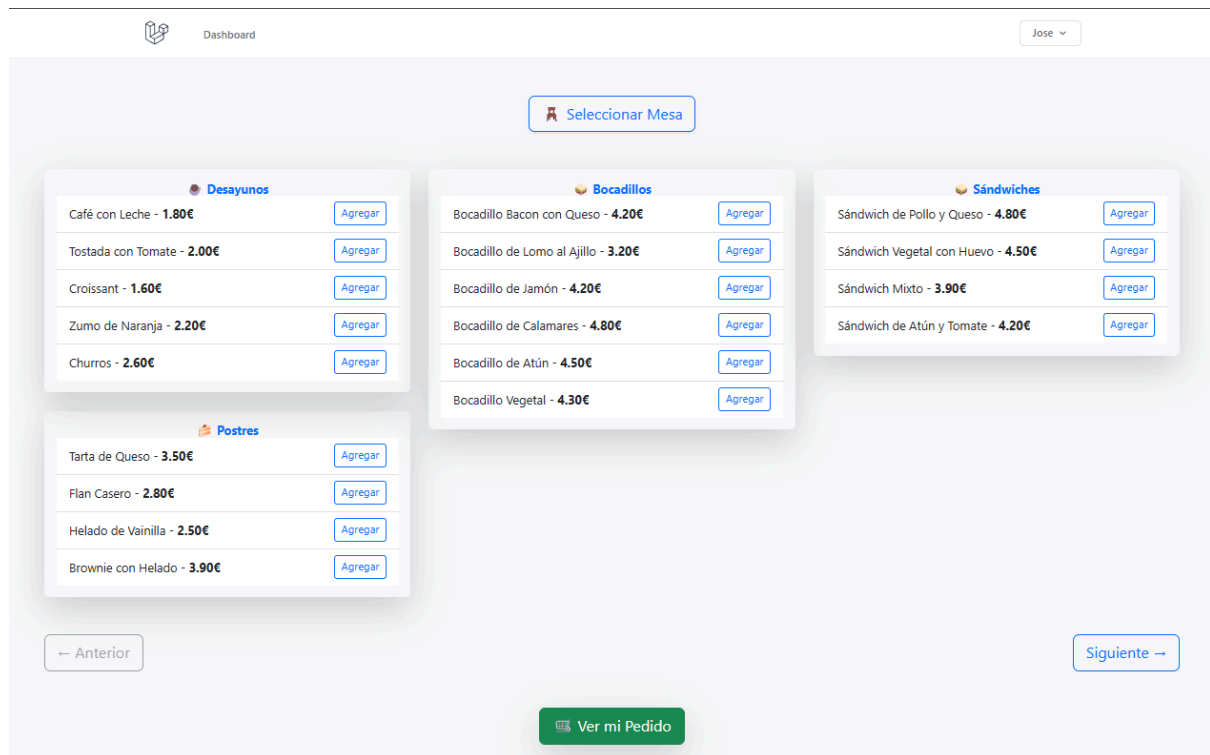
Consulta el menú disponible para los clientes.

[Ir al Menú](#)

Ubicaciones del Restaurante

Consulta nuestras ubicaciones en Tomelloso y Ciudad Real:





- Realizar el pedido, que se guarda en la base de datos.
- Pagar el pedido completo o solo productos seleccionados.
- Ver factura, donde podrá ver:
 - Fecha y hora.
 - ID de la mesa.
 - Lista de productos.
 - Precio unitario y total pagado.

FACTURA DE PAGO

Gracias por tu visita

Mesa: 3

Fecha: 02/06/2025

Hora: 16:44

Productos Pagados

Bocadillo de Lomo al Ajillo (x1)	3.20 €
Bocadillo Bacon con Queso (x1)	4.20 €
Bocadillo de Lomo al Ajillo (x1)	3.20 €
Sándwich Mixto (x1)	3.90 €

Total Pagado: 14.50 €

➡ Continuar pagando

- Diseño centrado en el usuario:
 - Botones grandes y diferenciados.
 - Mensajes claros de validación o advertencia.
 - Confirmaciones visuales tras acciones como pago, eliminación o errores.

Mi Pedido

Revisa tus productos y confirma tu pedido

[Volver al Menú](#)

Productos seleccionados

Sándwich Mixto	3.90€	Eliminar
Sándwich Vegetal con Huevo	4.50€	Eliminar
Sándwich de Pollo y Queso	4.80€	Eliminar
Bocadillo de Calamares	4.80€	Eliminar
Bocadillo de Jamón	4.20€	Eliminar
Bocadillo de Lomo al Ajillo	3.20€	Eliminar

Resumen del pedido

Total: **25.40€**

Confirmar Pedido

Confirmación del Pedido

¡Gracias! Este pedido ya ha sido pagado completamente.

Finalizar

- Que ve y hace un usuario administrador
 - El administrador tiene un conjunto de vistas adicionales accesibles desde el dashboard, solo si su rol es admin.
 - Acciones exclusivas del administrador:
 - Gestión de productos: ver, añadir, editar y eliminar.
 - Gestión de mesas: ver, añadir, editar, eliminar y liberar.
 - Acceso a rutas protegidas mediante middleware esadmin.



Daniel ▾

Gestión de Productos

Administra el catálogo de productos disponibles

[+ Crear Producto](#)

#	NOMBRE	PRECIO	CATEGORÍA	ACCIONES
1	Hamburguesa con Huevo	4.35€	Hamburguesas	Editar Eliminar
2	Coca-Cola	2.80€	Bebidas	Editar Eliminar
3	Bocadillo Bacon con Queso	4.20€	Bocadillos	Editar Eliminar
4	Bocadillo de Lomo al Ajillo	3.20€	Bocadillos	Editar Eliminar
5	Cerveza	3.20€	Bebidas	Editar Eliminar



Daniel ▾

Gestión de Mesas

Administra las mesas de tu restaurante

[+ Crear Mesa](#)

#	NOMBRE	ESTADO	ACCIONES
1	Mesa 1	Disponible	Editar Eliminar
2	Mesa 2	Disponible	Editar Eliminar
3	Mesa 3	Disponible	Editar Eliminar
4	Mesa 4	Ocupada	Editar Eliminar Liberar
5	Mesa 5	Disponible	Editar Eliminar

- **Visualización de ubicación del restaurante**

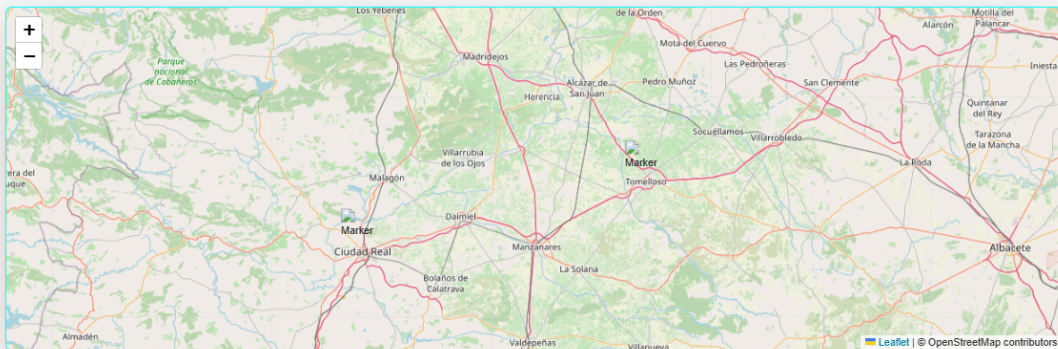
- En la parte inferior del dashboard, tanto los cliente como los administradores tienen acceso a una nueva funcionalidad visual: un mapa interactivo que permite visualizar las ubicaciones físicas del restaurante.
- Esta vista se integra directamente dentro del flujo de navegación del sistema y no requiere permisos especiales para acceder, por lo que está disponible para todos los roles (cliente o administrador).

- Cada local está señalado con un marcador interactivo y un mensaje emergente, facilitando la orientación y reduciendo la necesidad de preguntar o buscar manualmente la dirección.
- El mapa funciona de manera responsiva y puede visualizarse correctamente en cualquier tipo de dispositivos.



Ubicaciones del Restaurante

Consulta nuestras ubicaciones en Tomelloso y Ciudad Real:



5.3 Manual de instalación y configuración

- Herramientas necesarias para poder ejecutar el proyecto de forma local y sin errores, es necesario contar con el siguiente entorno y herramientas instaladas en el equipo:
 - **XAMPP** (o equivalente con Apache y MySQL)
 - **PHP 8.1 o superior**
 - **Composer** (gestor de dependencias de PHP)
 - **Node.js y NPM** (para compilar recursos front-end)
 - **Visual Studio Code** u otro editor de código
 - Navegador web (preferiblemente Google Chrome)

```

PS C:\xampp\htdocs\smartorder> npm -v
10.9.2
PS C:\xampp\htdocs\smartorder> php -v
PHP 8.2.12 (cli) (built: Oct 24 2023 21:15:15) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.2.12, Copyright (c) Zend Technologies
PS C:\xampp\htdocs\smartorder> composer -v

-----
/ / / / / V V V V V / / / / /
/ / / / / / / / / / / ( _ ) / / / / /
\ \ \ \ \ / / / / / . \ \ \ \ \ / / / / /
/ /

Composer version 2.8.6 2025-02-25 13:03:50

```

Es importante verificar que todos estos elementos estén correctamente configurados en las variables de entorno del sistema.

- Pasos para la instalación
 - Clonar o copiar el proyecto en la carpeta htdocs de XAMPP.
 - Acceder al terminal y situarse en el directorio del proyecto.
 - Ejecutar el comando: `composer install`
 - Configurar el archivo `.env` con los datos de nuestra base de datos local
 - Crear la base de datos manualmente en phpMyAdmin o con el siguiente comando si se usa CLI: `CREATE DATABASE smartorder;`
 - Ejecutar las migraciones: `php artisan migrate`
 - Instalar las dependencias fronted
 - `npm install`
 - `npm run dev`
 - `php artisan serve`
- La aplicación quedará disponible en **`http://localhost:8000`**

6. Mantenimiento y evolución

6.1 Plan de mantenimiento y soporte

- Una aplicación en producción requiere un plan de mantenimiento robusto y sostenible para garantizar su correcto funcionamiento a largo plazo. Este mantenimiento debe cubrir tanto el aspecto técnico como el soporte al usuario final. En el caso de SmartOrder, se propone una estrategia dividida en varias acciones concretas:
 - Corrección de errores detectados por los usuarios tras la puesta en marcha.
 - Los errores reportados por los usuarios serán documentados, clasificados por gravedad y corregidos en el menor tiempo posible.

- Actualización del entorno de Laravel, PHP o dependencias como Bootstrap y Tailwind, en caso de vulnerabilidades.
- Seguimiento del funcionamiento en producción, comprobando que tanto la gestión de pedido como los métodos de pago se ejecutan correctamente.
- Backup de base de datos, para evitar pérdidas de información en caso de errores, caídas del servidor o manipulaciones erróneas, se debe establecer una política de copias de seguridad de la base de datos MySQL.
 - Copias automáticas diarias de la base de datos.
 - Almacenamiento externo o en la nube.
 - Recuperación simple mediante phpMyAdmin o comandos de consola.

6.2 Identificación de mejoras

- Durante el desarrollo se han identificado funcionalidades útiles que no han sido implementadas por falta de tiempo, pero que podrían considerarse en futuras versiones para mejorar la experiencia del usuario y del personal del restaurante:
 - **Lectura de código QR**
 - Permitir que el cliente escanee un código QR presente en su mesa para acceder directamente al menú y comenzar el pedido. Esto elimina la necesidad de seleccionar manualmente la mesa, mejora la experiencia de usuario y reduce error..
 - **Nuevos roles de usuarios**
 - Añadir un rol de camarero, con acceso a los pedidos activos, posibilidad de gestionarlos y cambiar su estado.
 - Añadir un rol de cocinero, que solo vea los pedidos confirmados para empezar a preparar los platos.
 - Cada rol tendría permisos específicos y pantallas adaptadas.
 - **Control de inventario**
 - Almacenar la cantidad de stock disponible por producto o ingrediente.
 - Deshabilitar productos cuando no estén disponibles.
 - Emitir alertas cuando el stock esté por debajo de un umbral mínimo.
 - **Notificaciones en tiempo real**
 - Laravel Echo + WebSockets o Pusher para emitir notificaciones automáticas cuando se realiza un pedido o se confirma un pago.
 - Estas notificaciones aparecerán automáticamente en el panel de camareros o cocina.

6.3 Actualizaciones y mejoras futuras

- Aunque el sistema SmartOrder ya cumple con su propósito principal de gestionar pedidos y pagos en un entorno de hostelería, existen múltiples posibilidades para expandir y mejorar su funcionalidad. Estas futuras actualizaciones no solo optimizarán la experiencia del usuario, sino que también ampliarían el alcance del sistema hacia nuevas necesidades y perfiles de negocio.
 - **Versión móvil independiente**, si bien actualmente el sistema cuenta con un diseño responsive gracias a Bootstrap y Tailwind CSS, el objetivo a largo plazo sería desarrollar una aplicación móvil nativa tanto para Android como

para iOS. Esta aplicación podría conectarse con el backend de Laravel a través de una API REST, ofreciendo las siguientes ventajas.

- Interfaz totalmente adaptada a pantallas pequeñas, mejorando la usabilidad.
- Acceso directo mediante escaneo de código QR desde la cámara del móvil, lo cual eliminaría pasos intermedios como seleccionar mesa manualmente.
- Posibilidad de recibir notificaciones push cuando el pedido esté listo o se realicen actualizaciones.
- **Integración de pasarelas de pago**, hasta ahora, SmartOrder simula el proceso de pago para centrarse en la validación del flujo y la experiencia del usuario. En una versión futura se pretende integrar métodos de pagos reales, de forma que los clientes puedan abonar su consumo desde el propio dispositivo, sin necesidad de desplazarse a la caja, utilizando pasarelas como lo son:
 - Stripe
 - Bizum
 - PayPal
- **Panel de estadísticas**, uno de los módulos más solicitados por los negocios es el de análisis de datos. Se planea desarrollar un panel visual accesible desde el dashboard del administrador, con información en tiempo real sobre el rendimiento del local. Incluiríamos:
 - **Productos más vendidos**: identificación rápida del topo de ventas.
 - **Ingresos diarios, semanales o mensuales**: gráficos comparativos.
 - **Mesas más utilizadas**: para detectar patrones de uso del espacio.
 - **Tasa de ocupación por horas**: útil para planificación de turnos.
 - **Comparativas entre meses o temporadas**: seguimiento del crecimiento.
- **Traducción a otros idiomas**, con el objetivo de hacer el sistema accesible para turistas o negocios en zonas multilingües, se prevé añadir soporte para idiomas adicionales:
 - Inglés
 - Francés
 - Alemán.
- Implementación propuesta:
 - Detección automática del idioma del navegador del cliente.
 - Mejora la accesibilidad para turistas y clientes extranjeros.
 - Posibilidad de selección manual del idioma desde la interfaz.
- **Gestión avanzada de mesas**, actualmente, el sistema permite seleccionar una mesa disponible y liberar la misma tras el pago. Sin embargo, en una versión extendida se plantean las siguientes mejoras:
 - **Reservas anticipadas**: que el cliente pueda reservar mesa desde el móvil o página web, seleccionando día y hora.
 - **Combinación de mesas**: opción para juntar dos o más mesas en caso de grupos grandes.
 - **Registrar número de comensales por mesa**: para adaptar el menú, hacer sugerencias personalizadas o incluso calcular tiempos estimados de atención.

- En relación con la funcionalidad del mapa, se propone:
 - **Detección automática de ubicación del usuarios:** permitiendo centrar el mapa en su ubicación actual.
 - **Gestión de sucursales:** añadiendo desde el panel de administración nuevas ubicaciones, editarlas o eliminarlas
 - En resumen, SmartOrder tiene el potencial de evolucionar desde una aplicación funcional a una plataforma integral de gestión de restaurante, incorporando herramientas avanzadas de análisis, personalización y conectividad. Estas mejoras propuestas sientan las bases para una evolución sostenible y escalable en el tiempo.
-

7. Conclusiones

7.1 Evaluación del proyecto

- El desarrollo de la aplicación **SmartOrder** ha permitido cumplir con la mayor parte de las funcionalidades previstas al inicio del proyecto. Se ha logrado implementar un sistema de gestión de pedidos en entornos del sector hostelero con control de mesas, roles de usuario, separación de pagos y seguimiento del estado de cada comanda.
- A lo largo del desarrollo se ha conseguido implementar un sistema que permite a los usuario:
 - Seleccionar una mesa disponible.
 - Agregar productos desde un menú dividido por categorías.
 - Confirmar pedido, pagar total imparcial y visualizar facturas detalladas.
- Además, se ha incorporado un sistema de control de acceso por roles (cliente y administrador), lo cual permite segmentar la experiencia de uso y restringir funcionalidades críticas, como la gestión de productos o mesas.
- El resultado final ha superado las expectativas iniciales, ya que se han implementado funcionalidades adicionales no previstas en un primer momento, como la validación de flujo de acciones (por ejemplo, impedir pagar sin selección productos o sin mesas), mensaje de ayuda en los formularios, y una experiencia visual cuidada con alertas, colores intuitivos y una distribución clara de la información.
- A pesar de que algunas funcionalidades como el escaneo mediante QR o el control de inventario quedaron fuera del alcance del desarrollo inicial, la base del sistema se encuentra consolidada y preparada para una evolución posterior.

7.2 Cumplimiento de objetivos y requisitos

- En la fase de planificación inicial del proyecto se definieron una serie de objetivos generales y específicos que debían cumplirse. Tras el desarrollo completo del sistema, se puede afirmar que se han alcanzado prácticamente todos ellos:
 - Se ha desarrollado una interfaz limpia y amigable para el usuario, pensando tanto para administradores como para clientes.

- El sistema permite una gestión eficaz de mesas, impidiendo que dos clientes puedan seleccionar la misma mesa simultáneamente y liberándola automáticamente tras el pago.
- Se ha implementado la creación de pedido y gestión dinámica de productos, incluyendo la posibilidad de pagar por separado.
- Se ha establecido un control de roles, permitiendo que solo los administradores accedan a la gestión de productos y mesas.
- Se han validado los flujos de usuario para evitar errores lógicos o inconsistencias, como intentar pagar sin tener productos o sin haber seleccionado mesa.
- Los únicos elementos que quedaron como posibles mejoras futuras (y no como requisitos incumplidos) fueron:
 - Lectura de códigos QR para identificar automáticamente la mesa.
 - Creación de más roles intermedios (camareros, cocina).
 - Implementación de un sistema de gestión de stock o inventario.
- Cabe destacar que ninguno de los requisitos principales quedó sin cubrir, y las funcionalidades pendientes no afectan al funcionamiento principal del sistema.

7.3 Lecciones aprendidas y recomendaciones

- Este proyecto ha sido una oportunidad ideal para afianzar conocimientos teóricos mediante una aplicación práctica de principio a fin, desde el análisis inicial hasta la documentación final. Lecciones técnicas aprendidas:
 - Laravel: mayor dominio del framework, incluyendo rutas, controladores, middleware personalizados y validaciones.
 - Base de datos relacional: se reforzaron los conceptos de diseño de tablas, claves foráneas y migraciones.
 - Integración frontend: uso práctico de Bootstrap para un diseño responsivo y JavaScript para mejorar la interactividad.
 - Flujo de sesión: uso correcto de sesiones en Laravel para mantener el estado de usuario y pedidos.
- Lecciones metodológicas:
 - La importancia de validar constantemente las funcionalidades desde el punto de vista del usuario final.
 - La necesidad de estructurar el proyecto desde el inicio con claridad para evitar duplicación de lógica o rutas poco claras.
 - La documentación no debe dejarse para el final: explicar el código es parte fundamental del desarrollo.
 - La utilidad de herramientas como Git para tener versiones seguras y controladas.
- Después de todo lo que he tenido que hacer en el desarrollo y documentación de este proyecto y mi recomendación es la siguiente:
 - Planificar bien la base de datos: una estructura clara desde el principio facilita todo el desarrollo.
 - Haz pruebas reales durante el desarrollo, simulando errores intencionados y flujos incorrectos.
 - No dejar de lado la documentación, ya que es clave para el mantenimiento en el futuro.

- Utiliza buenas prácticas de codificación: nombres claros, separación de lógica y reutilización de código.
 - Por último, subestimes el diseño de la experiencia de usuario: un sistema funcional pero poco intuitivo es difícil de usar.
-

8. Bibliografía y referencias

8.1 Fuentes utilizadas

- Laravel docs
- Bootstrap docs
- Stack Overflow
- ChatGPT
- YouTube

8.2 Referencias y enlaces de interés

- <https://laravel.com>
- <https://getbootstrap.com>
- <https://chat.openai.com>